Fakulta elektrotechnická
Katedra elektrotechniky a počítačového modelování

Diplomová práce

Dynamické modely elektrických strojů

Autor práce: Bc. Martin Kadlec
Vedoucí práce: Doc. Ing. David Pánek, Ph.D.

Plzeň 2023

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Martin KADLEC**
Osobní číslo:	**E21N0050P**
Studijní program:	**N0714A060013 Elektronika a informační technologie**
Specializace:	**Výkonová elektronika**
Téma práce:	**Dynamické modely elektrických strojů**
Zadávající katedra:	**Katedra výkonové elektroniky a strojů**

Zásady pro vypracování

Vytvořte balíček v jazyce Python, který bude obsahovat nástroje pro tvorbu dynamických modelů elektrických strojů a pro syntézu řízení.

1. Proveďte podrobnou rešerši existujících balíčků pro simulaci a modelování dynamických systémů.
2. V jazyce Python implementujte balíček obsahující funkce pro tvorbu dynamických modelů a syntézu řízení.
3. Funkčnost baličku ověřte na konkrétním návrhu řízení pro zadaný stroj.

Rozsah diplomové práce: **40 – 60**
Rozsah grafických prací: **dle doporučení vedoucího**
Forma zpracování diplomové práce: **elektronická**

Seznam doporučené literatury:

J. Chiasson, *Modeling and High-Performance Control of Electric Machines*. Wiley-IEEE Press, 2005.

Vedoucí diplomové práce: **Doc. Ing. David Pánek, Ph.D.**
Katedra elektrotechniky a počítačového modelování

Datum zadání diplomové práce: **7. října 2022**
Termín odevzdání diplomové práce: **26. května 2023**

Západočeská univerzita v Plzni * Fakulta elektrotechnická *

L.S.

Prof. Ing. Zdeněk Peroutka, Ph.D.
děkan

Prof. Ing. Václav Kůs, CSc.
vedoucí katedry

V Plzni dne 7. října 2022

Abstrakt

V této diplomové práci je řešena problematika matematického modelování elektrických strojů. Jejím cílem bylo vytvořit nástroj s využitím vysokoúrovňového programovacího jazyka Python, který bude obsahovat dynamické modely elektrických strojů a na jejich základě bude schopen tyto stroje simulovat.

V první části práce je provedeno odvození modelu asynchronního stroje ve stojícím souřadném systému α, β. Je zde podrobně vysvětleno, jakým způsobem provést odvození matematického modelu a z něj vycházejícího náhradního schématu asynchronního stroje. Z tohoto základního modelu je poté odvozen stavový model tohoto stroje, který je často používán pro počítačové simulace. Tento stavový model byl také implementován do nástroje v jazyce Python, který je výstupem této diplomové práce.

Jako základ nástroje pro simulaci elektrických strojů byl použit existující balíček DynSyPy. Úpravami původního balíčku a implementací modelu asynchronního stroje se zabývá další část práce.

V závěru práce je provedeno testování vyvinutého balíčku pomocí dvou testovacích příkladů, které byly pro ověření realizovány také pomocí nástroje Matlab Simulink. Těmito testovacími příklady bylo prokázáno, že model asynchronního stroje a další třídy pro realizaci např. skalárního řízení tohoto stroje jsou funkční.

Klíčová slova

elektrický stroj, asynchronní stroj, model elektrického stroje, dynamický model, Python

Abstract

Kadlec, Martin. *Dynamical models of electric machines* [*Dynamické modely elektrických strojů*]. Pilsen, 2023. Master thesis (in Czech). University of West Bohemia. Faculty of Electrical Engineering. Department of Electrical Engineering and Computer Modeling. Supervisor: David Pánek

---

The purpose of this thesis is to solve the problem of mathematical modelling of electrical machines. Its' aim is to create a tool using the high-level programming language Python, which will contain dynamic models of electrical machines and on their basis will be able to simulate these machines.

In the first part of the work, a model of an induction machine in the standing coordinate system α, β is derived. It is explained in detail how to perform the derivation of the mathematical model and the resulting surrogate scheme of the induction machine. From this basic model, a state model of this machine is then derived, which is often used for computer simulations. This state model has also been implemented into a tool in Python language, which is the output of this thesis.

The existing DynSyPy package was used as the basis of the electrical machine simulation tool. The modifications of the original package and the implementation of the induction machine model are dealt with in the next part of the thesis.

The thesis concludes with the testing of the developed package using two test cases, which were implemented using Matlab Simulink for verification as well. Through these test examples, it was shown that the induction machine model and other classes for implementing e.g. scalar control of this machine are functional.

Keywords

electric machine, induction machine, electric machine model, dynamic model, Python

Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem svou závěrečnou práci vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 270 trestního zákona č. 40/2009 Sb.

Také prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

V Plzni dne 25. května 2023

Bc. Martin Kadlec

. . . . . .

Podpis

Poděkování

Na tomto místě bych velmi rád poděkoval doc. Ing. Davidu Pánkovi, Ph.D., vedoucímu diplomové práce, za odborné vedení práce, za podporu, věcné připomínky a trpělivost při jejím vytváření. Dále bych chtěl poděkovat doc. Ing. Jakubu Tallovi, Ph.D. za možnost konzultace k tématu odvozování modelů elektrických strojů. Děkuji také Ing. Martinu Goubejovi, Ph.D. za poskytnutou konzultaci k problematice návrhu regulátoru a linearizace systémů a za doporučení literatury k těmto tématům.

Obsah

Seznam obrázků **x**

Seznam tabulek **xi**

Seznam symbolů a zkratek **xii**

Úvod **1**

1 Matematické modelování elektrických strojů **3**
1.1 Matematický model asynchronního stroje 3
1.2 Odvození modelu asynchronního stroje pro počítačové simulace 9
1.2.1 Odvození komplexní stavové diferenciální rovnice pro rotorový tok . 10
1.2.2 Odvození komplexní stavové diferenciální rovnice pro statorový proud 11
1.2.3 Odvození pohybové stavové diferenciální rovnice 13

2 Regulace elektrických strojů **16**
2.1 PID (PI, PD) regulátory 17
2.1.1 Unášení integrační složky, anti-windup 18
2.2 Linearizace nelineárních dynamických systémů 20

3 Skalární řízení asynchronních strojů **23**
3.1 Odvození pravidel řízení ze statorové rovnice 23

4 Dostupné balíky v jazyce Python pro simulaci a modelování elektrických strojů **27**
4.1 _TINY_ACMSIMC 28
4.2 ACMSimPy 28

5 Realizace balíčku v jazyce Python **30**
5.1 Úpravy třídy System 31
5.1.1 Metoda `connect()` 31
5.1.2 Metoda `update_input()` 32
5.1.3 Metody `linear_regression()`, `input_linear_regression()` a `output_linear_regression()` 33

5.2 Nové abstraktní třídy balíčku DynSyPy 34
5.3 Abstraktní třída Machine 37
5.3.1 Metoda `clarke_transformation()` 37
5.3.2 Metoda `park_transformation()` 38
5.4 Třída SquirrelCageIM 39
5.4.1 Metody `transform()` a `update_output()` 42
5.4.2 Metoda `equation_of_state()` 43
5.4.3 Metody `non_linearity_03()` až `non_linearity_41()` 45
5.5 Třída Matrix 45
5.5.1 Metoda `eval()` 46
5.6 Třída ControlledNPhaseSine 46
5.6.1 Metoda `update_beta()` 48
5.6.2 Metoda `angle_range_adjustment()` 49
5.6.3 Metoda `source_initialize()` 50
5.7 Třída IMScalarControl 50
5.7.1 Metoda `system_function()` 52
5.8 Třída PIController 52
5.8.1 Metoda `saturation()` 53
5.8.2 Metoda `system_function()` 54

6 Ověření funkčnosti balíčku **55**
6.1 Testovací příklad – připojení ASM s kotvou nakrátko bez mechanické zátěže přímo k napěťovému zdroji 56
6.1.1 Ověření správnosti matematického modelu ASM 56
6.1.2 Simulace pomocí DynSyPy 61
6.2 Testovací příklad – skalární řízení ASM 65
6.2.1 Návrh PI regulátoru otáček ASM 65
6.2.2 Simulace pomocí Matlab Simulink 67
6.2.3 Simulace pomocí DynSyPy 70

Závěr **74**

Reference, použitá literatura **76**

Přílohy **78**

A Podklady pro simulace **78**
A.1 Model ASM v Matlab Simulink 78
A.2 Model ASM v Matlab Simulink pomocí PLECS 81
A.3 Model skalárního řízení ASM v Matlab Simulink pomocí PLECS 82

B Podklady pro realizaci experimentální identifikace **86**

B.1 Model v Matlab Simulink pro experimentální identifikaci 86

Seznam obrázků

1.1 Náhradní schéma jedné fáze statoru 3
1.2 Náhradní schéma jedné fáze rotoru 4
1.3 Náhradní schéma asynchronního stroje 9

2.1 Přímovazební regulační obvod [12] 16
2.2 Zpětnovazební regulační obvod s regulátorem 1DoF [12] 17
2.3 Unášení integrační složky při saturaci akčního členu [3] 19
2.4 PI regulátor s ochranou anti-windup založenou na principu back-calculation [13] 19

3.1 Náhradní schéma pro první harmonické veličin ASM v ustáleném stavu [17] 24
3.2 Zjednodušené náhradní schéma ASM [18] 25
3.3 Strukturní schéma pohonu s regulací otáček [17] 26

4.1 Okno real-time simulace pomocí balíku ACMSimPy 28

5.1 Zjednodušený diagram tříd balíčku DynSyPy 30
5.2 Diagram aktivit metody `connect()` 32
5.3 Diagram aktivit metody `update_input()` 33
5.4 Rozdělení systémů na state-space a non-state-space systémy 34
5.5 Třída PartiallyNonlinearSystem 35
5.6 Rozdělení systémů představujících zdroje 36
5.7 Třída Controllers 36
5.8 Třída Machine 37
5.9 Třída SquirrelCageIM 40
5.10 Znázornění činnosti metod `transform()` a `update_output()` 42
5.11 Skutečná realizace metody `update_output()` 43
5.12 Třída Matrix 45
5.13 Diagram aktivit metody `eval()` 46
5.14 Třída ControlledNPhaseSine 47
5.15 Ukázka výstupu objektů třídy ControlledNPhaseSine, nahoře třífázový zdroj, dole pětifázový zdroj 48

5.16 Realizace n-fázového zdroje, nahoře běžný způsob, dole využití rotujícího úhlu β 49
5.17 Diagram aktivit metody `source_initialize()` 50
5.18 Třída IMScalarControl 51
5.19 Třída PIController 52

6.1 ASM naprázdno přímo připojený ke zdroji napětí, realizace pomocí nástroje PLECS 56
6.2 Parametry ASM použité při simulaci pomocí PLECS 57
6.3 Připojení ASM přímo ke zdroji 3f napětí, model pomocí nástroje PLECS, průběh u_{sa}, u_{sb} a u_{sc} 57
6.4 Připojení ASM přímo ke zdroji 3f napětí, model pomocí nástroje PLECS, průběh i_{sa}, i_{sb} a i_{sc} 58
6.5 Připojení ASM přímo ke zdroji 3f napětí, model pomocí nástroje PLECS, průběh ω_{m} 58
6.6 Připojení ASM přímo ke zdroji 3f napětí, model v Matlab Simulink, průběh u_{sa}, u_{sb} a u_{sc} 59
6.7 Připojení ASM přímo ke zdroji 3f napětí, model v Matlab Simulink, průběh i_{sa}, i_{sb} a i_{sc} 60
6.8 Připojení ASM přímo ke zdroji 3f napětí, model v Matlab Simulink, průběh ω_{m} 60
6.9 Připojení ASM přímo ke zdroji 3f napětí, model v DynSyPy, průběh u_{sa}, u_{sb} a u_{sc} 63
6.10 Připojení ASM přímo ke zdroji 3f napětí, model v DynSyPy, průběh i_{sa}, i_{sb} a i_{sc} 64
6.11 Připojení ASM přímo ke zdroji 3f napětí, model v DynSyPy, průběh ω_{m} . . 64
6.12 Ukázka návrhu regulátoru pomocí nástroje Control System Designer 65
6.13 Ukázka identifikace systému pomocí nástroje System Identification Toolbox 66
6.14 Skalární řízení, model pomocí nástroje PLECS, průběh ω_{m} 68
6.15 Skalární řízení, model pomocí nástroje PLECS, průběh u_{sa}, u_{sb} a u_{sc} 68
6.16 Skalární řízení, model pomocí nástroje PLECS, průběh i_{sa}, i_{sb} a i_{sc} 69
6.17 Skalární řízení, model v DynSyPy, průběh ω_{m} 72
6.18 Skalární řízení, model v DynSyPy, průběh u_{sa}, u_{sb} a u_{sc} 72
6.19 Skalární řízení, model v DynSyPy, průběh i_{sa}, i_{sb} a i_{sc} 73

A.1 Model ASM pomocí Matlab Simulink, přímé připojení ASM ke zdroji napětí 79
A.2 Realizace bloku `ASM_model` pomocí Matlab Simulink, viz obr. A.1 a B.2 . . 80
A.3 Model ASM pomocí nástroje PLECS, obvod v bloku `ASM` je uveden na obr. 6.1 81
A.4 Realizace skalárního řízení ASM v Matlab Simulink pomocí PLECS 83

A.5 Realizace modelu ASM pro skalární řízení pomocí PLECS, blok `ASM` viz obr. A.4 84
A.6 Realizace bloku `PI_regulator_anti-windup`, viz obr. A.4 a B.1 84
A.7 Realizace bloku `scalar_control`, viz obr. A.4 a B.2 85
A.8 Realizace bloku `inverter`, viz obr. A.4 a B.2 85

B.1 Model v Matlab Simulink použitý pro experimentální identifikaci 87
B.2 Realizace bloku `ASM_subsystem`, viz obr. B.1 88

Seznam tabulek

6.1 Parametry navržených regulátorů 67

Seznam symbolů a zkratek

Značky a symboly

$\mathbf{A}$		Matice systému o rozměrech ($n \times n$).
$\overline{a}$		Operátor natočení, $\overline{a} = \mathrm{e}^{\mathrm{j} \cdot \frac{2 \cdot \pi}{3}}$.
$\mathbf{B}$		Vstupní matice o rozměrech ($n \times m$).
B_{ω_m}	$(\mathrm{N} \cdot \mathrm{m})$	Koeficient tření.
$\mathbf{C}$		Výstupní matice o rozměrech ($p \times n$).
$\mathbf{C}_A$		Transformační matice pro dopřednou transformaci Clarkeové.
$\mathbf{C}_A^{-1}$		Transformační matice pro zpětnou transformaci Clarkeové.
$\mathbf{D}$		Matice přímého působení vstupu systému na jeho výstup o rozměrech ($p \times m$).
d		Reálná osa rotujícího souřadného systému d, q.
$E(p)$		Laplaceův obraz regulační odchylky.
e		Eulerovo číslo, $\mathrm{e} \doteq 2,71828182846$.
$e(t)$		Regulační odchylka, $e(t) = w(t) - y(t)$.
$e_s(t)$		Rozdíl řídicího signálu a signálu na výstupu akčního členu, $e_s(t) = u(t) - \hat{u}(t)$.
$F_\mathrm{R}(p)$		Přenosová funkce regulátoru.
$F_\mathrm{S}(p)$		Přenosová funkce řízeného systému.
f	(Hz)	Frekvence.
f_r	(Hz)	Frekvence rotorového napětí.
f_rk	(Hz)	Kritická frekvence rotorového napětí.
f_s	(Hz)	Frekvence statorového napětí (popř. synchronní frekvence).
f_sN	(Hz)	Jmenovitá frekvence statorového napětí.
G_v		Generátor výstupní poruchy $v(t)$.
G_w		Generátor referenčního signálu $w(t)$.
$\overline{\vec{i}}_\mathrm{m}$	(A)	Prostorový vektor magnetizačního proudu.
$\overline{\vec{i}}_\mu$	(A)	Prostorový vektor magnetizačního proudu.
$\overline{\vec{i}}_\mathrm{r}$	(A)	Prostorový vektor proudu rotoru.
$i_{\mathrm{r}x}$	(A)	Proud fází x rotoru.

$\bar{i}_\mathrm{s}$	(A)	Prostorový vektor proudu statoru.
$i_{\mathrm{s}x}$	(A)	Proud fází x statoru.
J	($\mathrm{kg \cdot m^2}$)	Moment setrvačnosti.
j		Imaginární jednotka, $\mathrm{j}^2 = -1$.
K		Proporcionální zesílení regulátoru.
K_d		Derivační konstanta regulátoru.
K_{f_r}		Konstanta skalárního řízení.
K_i		Integrační konstanta regulátoru.
K_U		Konstanta skalárního řízení.
k		Konstanta transformace Clarkeové.
k_p		Konstanta transformace Clarkeové.
L_h	(H)	Hlavní (magnetizační) indukčnost.
L'_r	(H)	Indukčnost rotoru přepočítaná na stator.
L_r	(H)	Indukčnost rotorového vinutí.
L_s	(H)	Indukčnost statorového vinutí.
$L'_{\sigma\mathrm{r}}$	(H)	Rozptylová indukčnost rotoru přepočítaná na stator.
$L_{\sigma\mathrm{s}}$	(H)	Rozptylová indukčnost statoru.
M	($\mathrm{N \cdot m}$)	Moment motoru.
M_z	($\mathrm{N \cdot m}$)	Moment zátěže.
N_r		Počet závitů rotorového vinutí.
N_s		Počet závitů statorového vinutí.
p_p		Počet pólových dvojic (pólpárů) stroje.
q		Imaginární osa rotujícího souřadného systému d, q.
$R(p)$		Přenosová funkce přímovazebního regulátoru.
R_r	(Ω)	Odpor rotorového vinutí.
R'_r	(Ω)	Odpor rotorového vinutí přepočítaný na stator.
R_s	(Ω)	Odpor statorového vinutí.
s		Skluz.
T_d		Derivační časová konstanta regulátoru.
T_i		Integrační časová konstanta regulátoru.
T_r		Časová konstanta zpětné vazby od anti-windup ochrany.
t	(s)	Čas.
$U(p)$		Laplaceův obraz řídicího signálu regulátoru.
U_m	(V)	Amplituda napětí.
U_sN	(V)	Jmenovité statorové napětí.
$u(t)$		Řídicí signál.
$\boldsymbol{u}(t)$		Vektor vstupu systému.
$\hat{u}(t)$		Signál na výstupu akčního členu.
$u_{\mathrm{i}_{\mathrm{r}x}}$	(V)	Indukované napětí fáze x rotoru.
$u_{\mathrm{i}_{\mathrm{s}x}}$	(V)	Indukované napětí fáze x statoru.

$\boldsymbol{u}_{\text{konst.}}$		Konstantní vstup systému.
$\overline{u}_{\text{r}}$	(V)	Prostorový vektor napětí na rotoru.
$\overline{u}'_{\text{r}_\text{I}}$	(V)	Prostorový vektor napětí na rotoru přepočítaný na stator ve stojícím souřadném systému α, β statoru.
$u_{\text{r}x}$	(V)	Napájecí napětí fáze x rotoru.
$\overline{u}_{\text{s}}$	(V)	Prostorový vektor napětí na statoru.
$u_{\text{s}x}$	(V)	Napájecí napětí fáze x statoru.
$v(t)$		Výstupní porucha.
$w(t)$		Referenční signál.
X_{h}	(Ω)	Reaktance hlavní indukčnosti.
$X_{\text{r}\sigma}$	(Ω)	Reaktance rozptylové indukčnosti rotoru.
$X_{\text{s}\sigma}$	(Ω)	Reaktance rozptylové indukčnosti statoru.
$\boldsymbol{x}(t)$		Vektor stavu systému.
$\overline{x}$		Prostorový vektor veličiny x.
$x^{(1)}$		První harmonická veličiny x.
$\overline{x}_{\text{I}}$		Prostorový vektor veličiny x ve stojícím souřadném systému α, β statoru.
$\overline{x}_{\text{II}}$		Prostorový vektor veličiny x v souřadném systému α, β stojícím vzhledem k rotoru.
$\boldsymbol{x}_r$		Ustálený stav systému.
x_α		Složka α prostorového vektoru veličiny x ve stojícím souřadném systému α, β.
x_β		Složka β prostorového vektoru veličiny x ve stojícím souřadném systému α, β.
$y(t)$		Měřený regulovaný výstup.
$\boldsymbol{y}(t)$		Vektor výstupu systému.
$\boldsymbol{y}_r$		Ustálený výstup systému.
α		Konstanta pro zpřehlednění modelu ASM.
α		Reálná osa stojícího souřadného systému α, β.
β		Imaginární osa stojícího souřadného systému α, β.
β		Konstanta pro zpřehlednění modelu ASM.
β	(rad)	Tzv. rotující úhel.
γ		Konstanta pro zpřehlednění modelu ASM.
δ		Konstanta pro zpřehlednění modelu ASM.
ε		Konstanta pro zpřehlednění modelu ASM.
λ		Konstanta pro zpřehlednění modelu ASM.
Φ_{h}	(Wb)	Hlavní magnetický indukční tok.
Φ_{r}	(Wb)	Magnetický indukční tok rotoru.
Φ_{s}	(Wb)	Magnetický indukční tok statoru.

$\Phi_{\sigma\mathrm{r}}$	(Wb)	Rozptylový magnetický indukční tok rotoru.
$\Phi_{\sigma\mathrm{s}}$	(Wb)	Rozptylový magnetický indukční tok statoru.
φ	(rad)	Fázový posun.
Ψ_{h}	(Wb)	Hlavní magnetický indukční tok spřažený s hlavní indukčností (magnetizační).
Ψ_{hr}	(Wb)	Příspěvek od rotoru do hlavního magnetického indukčního toku.
Ψ_{hs}	(Wb)	Příspěvek od statoru do hlavního magnetického indukčního toku.
Ψ_{r}	(Wb)	Magnetický indukční tok rotoru spřažený s vinutím rotoru.
$\overline{\Psi}_{\mathrm{r}}$	(Wb)	Prostorový vektor rotorového toku spřaženého s vinutím rotoru.
$\overline{\Psi}'_{\mathrm{r_I}}$	(Wb)	Prostorový vektor rotorového toku spřaženého s vinutím rotoru přepočítaný na stator ve stojícím souřadném systému α, β statoru.
$\Psi_{\mathrm{r}x}$	(Wb)	Magnetický indukční tok fáze x rotoru.
Ψ_{s}	(Wb)	Magnetický indukční tok statoru spřažený s vinutím statoru.
$\overline{\Psi}_{\mathrm{s}}$	(Wb)	Prostorový vektor statorového toku spřaženého s vinutím statoru.
$\Psi_{\mathrm{s}x}$	(Wb)	Magnetický indukční tok fáze x statoru.
$\Psi_{\sigma\mathrm{r}}$	(Wb)	Rozptylový magnetický indukční tok rotoru spřažený s vinutím rotoru.
$\overline{\Psi}'_{\sigma\mathrm{r_I}}$	(Wb)	Prostorový vektor rozptylového toku rotoru spřaženého s vinutím rotoru přepočítaný na stator ve stojícím souřadném systému α, β statoru.
$\Psi_{\sigma\mathrm{s}}$	(Wb)	Rozptylový magnetický indukční tok statoru spřažený s vinutím statoru.
ω_{II}	$(\mathrm{rad}\cdot\mathrm{s}^{-1})$	Elektrická úhlová rychlost pohybu vzhledem ke statoru rotujícího souřadného systému.
ω_{m}	$(\mathrm{rad}\cdot\mathrm{s}^{-1})$	Mechanická úhlová rychlost rotoru.
ω_{r}	$(\mathrm{rad}\cdot\mathrm{s}^{-1})$	Úhlová rychlost rotorového napětí.
ω_{s}	$(\mathrm{rad}\cdot\mathrm{s}^{-1})$	Úhlová rychlost statorového napětí (popř. synchronní úhlová rychlost), $\omega_{\mathrm{s}} = 2\cdot\pi\cdot f_{\mathrm{s}}$.

Zkratky

1DoF	1 Degree of Freedom regulator. Regulátor s jedním stupněm volnosti.
2DoF	2 Degree of Freedom regulator. Regulátor s dvěma stupni volnosti.
ASM	Asynchronní motor.
DFIM	Doubly Fed Induction Machine. Dvojitě napájený asynchronní stroj.
GPU	Graphics Processing Unit. Grafický procesor.
GUI	Graphic User Interface. Grafické uživatelské rozhraní.
LDS	Lineární dynamický systém.
LTI	Linear Time-Invariant (system). Lineární časově nezávislý (systém).
MIMO	Multiple Input – Multiple Output. Systémy s několika vstupy a s několika výstupy.
MISO	Multiple Input – Single Output. Systémy s několika vstupy a s jedním výstupem.
NDS	Nelineární dynamický systém.
PID	Proporcionálně integračně derivační (regulátor).
PLECS	Piecewise Linear Electrical Circuit Simulation. Simulační platforma pro výkonové elektronické systémy.
PWM	Pulse Width Modulation. Pulsně-šířková modulace.
RK4	4th order Runge-Kutta numerical integration method. Numerická integrační metoda Runge-Kutta 4. řádu.
RKF	Numerical integration adaptive method Runge-Kutta-Fehlberg. Numerická integrační adaptivní metoda Runge-Kutta-Fehlberg.
SIMO	Single Input – Multiple Output. Systémy s jedním vstupem a s několika výstupy.
SISO	Single Input – Single Output. Systémy s jedním vstupem a s jedním výstupem.
TI	Texas Instruments.

Úvod

V dnešní době jsou elektrické stroje klíčovými součástmi moderních elektrotechnických systémů a mají široké uplatnění v průmyslu, energetice, dopravě a dalších oblastech. Provedení návrhu elektrického stroje je složitou úlohou, v níž je potřeba zohlednit velké množství faktorů. Pro zjednodušení návrhu je v současnosti možné použít výpočetní techniku a na základě matematických modelů simulovat např. rozložení magnetického pole ve stroji. Matematických modelů je možné využít také při návrhu řízení elektrických strojů. Pomocí dynamických modelů je možné simulovat chování elektrického stroje pro různé hodnoty napájení a sledovat jeho odezvy na nejrůznější provozní vlivy, jako jsou např. skokové změny v momentu zátěže. Tato práce se věnuje právě dynamickému modelování elektrických strojů s využitím vysokoúrovňového programovacího jazyka Python. Jejím výstupem je nástroj v jazyce Python, kterým je možné simulovat asynchronní stroj.

První část práce se zabývá odvozením matematického modelu asynchronního stroje ve stojícím souřadném systému α, β. Je zde podrobně vysvětleno, jakým způsobem provést odvození matematického modelu a z něj vycházejícího náhradního schématu asynchronního stroje. Z tohoto základního modelu je poté odvozen stavový model tohoto stroje, který je často používán pro počítačové simulace. Dále je uvedena stručná teorie k řízení strojů pomocí PID regulátorů, kde je také část věnována problému unášení integrační složky regulátoru a ochraně před ním (anti-windup). V této kapitole je také zmíněna jedna z možností provedení linearizace nelineárního systému v okolí pracovního bodu. Znalosti PID regulace a linearizace byly použiti při návrhu regulátoru, jež byl použit při realizaci jednoho z testovacích příkladů vyvinutého balíku. Tím je provedení skalárního řízení asynchronního motoru s regulací otáček. V další kapitole je tedy řešen princip skalárního řízení a jeho odvození.

V praktické části práce je řešen návrh nástroje pro modelování elektrických strojů v jazyce Python. Před samotným návrhem byla provedena rešerše na existující nástroje pro modelování strojů. Té se věnuje kapitola 4. V další kapitole je popsána realizace balíčku v jazyce Python. Jeho základem byl balík DynSyPy vyvinutý v rámci předešlé bakalářské práce [9]. Jsou zde popsány provedené změny původního balíku, nové třídy a jejich metody a atributy.

Poslední kapitola se zabývá testováním vytvořeného nástroje. Na dvou testovacích příkladech byla ověřována jednak správnost implementace modelu asynchronního stroje v balíčku DynSyPy, ale také schopnost balíčku simulovat složitější úlohy. Pro ověření

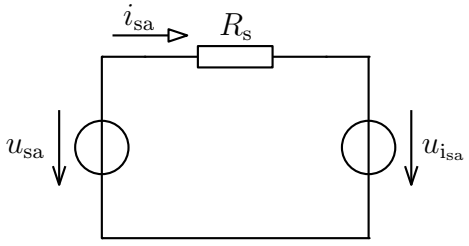
byly oba testovací příklady paralelně realizovány také pomocí Matlab Simulink a výstupy z obou nástrojů byly následně porovnány a vyhodnoceny.

Kapitola 1

Matematické modelování elektrických strojů

Návrh pohonů bývá velmi často realizován s využitím matematických modelů, pomocí nichž je možné popsat chování elektrických strojů, navrhovat pro ně řídicí algoritmy a také provádět počítačové simulace. Při návrhu elektrického stroje jsou velmi často využívány modely získané pomocí tzv. metody konečných prvků. Tím lze popsat rozložení magnetických a tepelných polí ve stroji a na základě takto získaných informací optimalizovat stroj pro konkrétní aplikaci v praxi. Pro návrh řízení jsou pak často využívány dynamické modely strojů, pomocí kterých je možné provést např. návrh regulátoru. Tato kapitola se bude věnovat odvozování matematického modelu asynchronního stroje. Bude zde podrobně vysvětleno, jakým způsobem při odvození modelu postupovat.

1.1 Matematický model asynchronního stroje



Obr. 1.1: Náhradní schéma jedné fáze statoru

Ze schématu na obr. 1.1 lze pomocí druhého Kirchhoffova zákona odvodit rovnici pro napětí jedné fáze statoru:

$$u_{\mathrm{sa}} = R_{\mathrm{s}} \cdot i_{\mathrm{sa}} + u_{\mathrm{i_{sa}}} \ , \tag{1.1}$$

kde:

u_{sa} (V) – napájecí napětí fáze „a“ statoru,
R_{s} (Ω) – odpor jedné fáze statorového vinutí,
i_{sa} (A) – proud fází „a“ statoru,
$u_{\mathrm{i_{sa}}}$ (V) – indukované napětí fáze „a“ statoru.

Indukované napětí $u_{\mathrm{i_{sa}}}$ lze vyjádřit jako

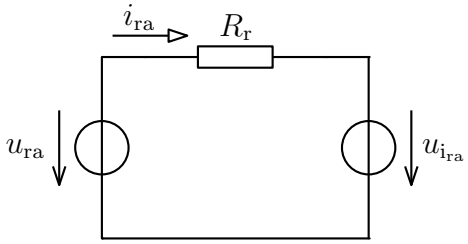
$$u_{\mathrm{i_{sa}}} = \frac{\mathrm{d}\Psi_{\mathrm{sa}}}{\mathrm{d}t} \ . \tag{1.2}$$

Obdobným způsobem je možné odvodit rovnice pro všechny fáze statoru: [24]

$$u_{\mathrm{sa}} = R_{\mathrm{s}} \cdot i_{\mathrm{sa}} + \frac{\mathrm{d}\Psi_{\mathrm{sa}}}{\mathrm{d}t} \ , \tag{1.3}$$

$$u_{\mathrm{sb}} = R_{\mathrm{s}} \cdot i_{\mathrm{sb}} + \frac{\mathrm{d}\Psi_{\mathrm{sb}}}{\mathrm{d}t} \ , \tag{1.4}$$

$$u_{\mathrm{sc}} = R_{\mathrm{s}} \cdot i_{\mathrm{sc}} + \frac{\mathrm{d}\Psi_{\mathrm{sc}}}{\mathrm{d}t} \ . \tag{1.5}$$



Obr. 1.2: Náhradní schéma jedné fáze rotoru

Ze schématu na obr. 1.2 lze pomocí druhého Kirchhoffova zákona odvodit rovnici pro napětí jedné fáze rotoru:

$$u_{\mathrm{ra}} = R_{\mathrm{r}} \cdot i_{\mathrm{ra}} + u_{\mathrm{i_{ra}}} \ , \tag{1.6}$$

kde:

u_{ra} (V) – napájecí napětí fáze „a“ rotoru,
R_{r} (Ω) – odpor jedné fáze rotorového vinutí,
i_{ra} (A) – proud fází „a“ rotoru,
$u_{\mathrm{i_{ra}}}$ (V) – indukované napětí fáze „a“ rotoru.

Indukované napětí $u_{\mathrm{i_{ra}}}$ lze vyjádřit jako

$$u_{\mathrm{i_{ra}}} = \frac{\mathrm{d}\Psi_{\mathrm{ra}}}{\mathrm{d}t} \tag{1.7}$$

Obdobným způsobem je možné odvodit rovnice pro všechny fáze rotoru: [24]

$$u_{\mathrm{ra}} = R_{\mathrm{r}} \cdot i_{\mathrm{ra}} + \frac{\mathrm{d}\Psi_{\mathrm{ra}}}{\mathrm{d}t} \ , \tag{1.8}$$

$$u_{\mathrm{rb}} = R_{\mathrm{r}} \cdot i_{\mathrm{rb}} + \frac{\mathrm{d}\Psi_{\mathrm{rb}}}{\mathrm{d}t} \ , \tag{1.9}$$

$$u_{\mathrm{rc}} = R_{\mathrm{r}} \cdot i_{\mathrm{rc}} + \frac{\mathrm{d}\Psi_{\mathrm{rc}}}{\mathrm{d}t} \ . \tag{1.10}$$

Pomocí transformace Clarkeové (popsána např. v [15]) lze fázová napětí statoru – vztahy (1.3), (1.4) a (1.5), i rotoru – vztahy (1.8), (1.9) a (1.10), transformovat na prostorové vektory v souřadném systému α, β:

$$\overline{u}_{\mathrm{s_I}} = k \cdot \left(u_{\mathrm{sa}} + \overline{a} \cdot u_{\mathrm{sb}} + \overline{a}^2 \cdot u_{\mathrm{sc}}\right) \ , \tag{1.11}$$

$$\overline{u}_{\mathrm{r_{II}}} = k \cdot \left(u_{\mathrm{ra}} + \overline{a} \cdot u_{\mathrm{rb}} + \overline{a}^2 \cdot u_{\mathrm{rc}}\right) \ , \tag{1.12}$$

kde $\overline{a}$ je tzv. operátor natočení, pro který platí [14]

$$\overline{a} = \mathrm{e}^{\mathrm{j} \cdot \frac{2 \cdot \pi}{3}} = -\frac{1}{2} + \mathrm{j} \cdot \frac{\sqrt{3}}{2} \ , \tag{1.13}$$

$$\overline{a}^2 = \mathrm{e}^{-\mathrm{j} \cdot \frac{2 \cdot \pi}{3}} = -\frac{1}{2} - \mathrm{j} \cdot \frac{\sqrt{3}}{2} \ . \tag{1.14}$$

Musí tedy také platit [14]

$$1 + \overline{a} + \overline{a}^2 = 0 \ . \tag{1.15}$$

Index I označuje prostorový vektor ve stojícím souřadném systému statoru. Index II označuje prostorový vektor v souřadném systému stojícím vzhledem k rotoru. Vzhledem ke statoru se tedy pohybuje elektrickou rychlostí rotoru $\omega_{\mathrm{II}} = p_{\mathrm{p}} \cdot \omega_{\mathrm{m}}$. Konstanta k vychází různě podle druhu zvolené transformace (např. pro amplitudově invariantní transformaci $k = \frac{2}{3}$). Odvození konstant pro transformaci Clarkeové je přehledně provedeno v [24].

$$\overline{u}_{\mathrm{s_I}} = R_{\mathrm{s}} \cdot \overline{i}_{\mathrm{s_I}} + \frac{\mathrm{d}\overline{\Psi}_{\mathrm{s_I}}}{\mathrm{d}t} \ , \tag{1.16}$$

$$\overline{u}_{\mathrm{r_{II}}} = R_{\mathrm{r}} \cdot \overline{i}_{\mathrm{r_{II}}} + \frac{\mathrm{d}\overline{\Psi}_{\mathrm{r_{II}}}}{\mathrm{d}t} \ . \tag{1.17}$$

Pro magnetické indukční toky $\overline{\Psi}_{\mathrm{s}}$ a $\overline{\Psi}_{\mathrm{r}}$ platí následující rovnosti:

$$\overline{\Psi}_{\mathrm{s}} = \overline{\Psi}_{\mathrm{h}} + \overline{\Psi}_{\sigma\mathrm{s}} = N_{\mathrm{s}} \cdot \overline{\Phi}_{\mathrm{h}} + N_{\mathrm{s}} \cdot \overline{\Phi}_{\sigma\mathrm{s}} = N_{\mathrm{s}} \cdot \left(\overline{\Phi}_{\mathrm{h}} + \overline{\Phi}_{\sigma\mathrm{s}}\right) = L_{\mathrm{s}} \cdot \overline{i}_{\mathrm{s}} \ , \tag{1.18}$$

$$\overline{\Psi}_{\mathrm{r}} = \overline{\Psi}_{\mathrm{h}} + \overline{\Psi}_{\sigma\mathrm{r}} = N_{\mathrm{r}} \cdot \overline{\Phi}_{\mathrm{h}} + N_{\mathrm{r}} \cdot \overline{\Phi}_{\sigma\mathrm{r}} = N_{\mathrm{r}} \cdot \left(\overline{\Phi}_{\mathrm{h}} + \overline{\Phi}_{\sigma\mathrm{r}}\right) = L_{\mathrm{r}} \cdot \overline{i}_{\mathrm{r}} \ . \tag{1.19}$$

kde:

- $\Psi_{\mathrm{s}}, \Psi_{\mathrm{r}}$ (Wb) – Magnetický indukční tok statoru/rotoru spřažený s vinutím statoru/rotoru.
- Ψ_{h} (Wb) – Hlavní magnetický indukční tok spřažený s hlavní indukčností (magnetizační).
- $\Psi_{\sigma\mathrm{s}}, \Psi_{\sigma\mathrm{r}}$ (Wb) – Rozptylový magnetický indukční tok statoru/rotoru spřažený s vinutím statoru/rotoru.
- Φ_{h} (Wb) – Hlavní magnetický indukční tok.
- $\Phi_{\sigma\mathrm{s}}, \Phi_{\sigma\mathrm{r}}$ (Wb) – Rozptylový magnetický indukční tok statoru/rotoru.
- $N_{\mathrm{s}}, N_{\mathrm{r}}$ $(-)$ – Počet závitů vinutí statoru/rotoru.

$L_\mathrm{s}, L_\mathrm{r}$ (H) – Indukčnost statorového/rotorového vinutí.

Na základě těchto vztahů je možné upravit rovnice (1.16) a (1.17) do tvaru:

$$\overline{u}_{\mathrm{s_I}} = R_\mathrm{s} \cdot \overline{i}_{\mathrm{s_I}} + N_\mathrm{s} \cdot \frac{\mathrm{d}\overline{\Phi}_{\mathrm{h_I}}}{\mathrm{d}t} + N_\mathrm{s} \cdot \frac{\mathrm{d}\overline{\Phi}_{\sigma\mathrm{s_I}}}{\mathrm{d}t} \ , \tag{1.20}$$

$$\overline{u}_{\mathrm{r_{II}}} = R_\mathrm{r} \cdot \overline{i}_{\mathrm{r_{II}}} + N_\mathrm{r} \cdot \frac{\mathrm{d}\overline{\Phi}_{\mathrm{h_{II}}}}{\mathrm{d}t} + N_\mathrm{r} \cdot \frac{\mathrm{d}\overline{\Phi}_{\sigma\mathrm{r_{II}}}}{\mathrm{d}t} \ . \tag{1.21}$$

Vyjádřená napětí je nyní potřeba přepočítat do společného souřadného systému – statorový stojící souřadný systém α, β. Přepočet rotorového napětí se provádí vynásobením komplexní exponenciální funkcí:

$$\overline{u}_{\mathrm{r_I}} = \overline{u}_{\mathrm{r_{II}}} \cdot \mathrm{e}^{-\mathrm{j}\cdot p_\mathrm{p}\cdot\omega_\mathrm{m}\cdot t} \ . \tag{1.22}$$

Po dosazení

$$\overline{u}_{\mathrm{r_I}} = R_\mathrm{r} \cdot \overline{i}_{\mathrm{r_{II}}} \cdot \mathrm{e}^{-\mathrm{j}\cdot p_\mathrm{p}\cdot\omega_\mathrm{m}\cdot t} + N_\mathrm{r} \cdot \frac{\mathrm{d}\left(\overline{\Phi}_{\mathrm{h_{II}}} \cdot \mathrm{e}^{-\mathrm{j}\cdot p_\mathrm{p}\cdot\omega_\mathrm{m}\cdot t}\right)}{\mathrm{d}t} + N_\mathrm{r} \cdot \frac{\mathrm{d}\left(\overline{\Phi}_{\sigma\mathrm{r_{II}}} \cdot \mathrm{e}^{-\mathrm{j}\cdot p_\mathrm{p}\cdot\omega_\mathrm{m}\cdot t}\right)}{\mathrm{d}t} \ . \tag{1.23}$$

Následuje derivace funkcí (derivace součinu funkcí):

$$\begin{aligned}
\overline{u}_{\mathrm{r_I}} = R_\mathrm{r} \cdot \underbrace{\overline{i}_{\mathrm{r_{II}}} \cdot \mathrm{e}^{-\mathrm{j}\cdot p_\mathrm{p}\cdot\omega_\mathrm{m}\cdot t}}_{\overline{i}_{\mathrm{r_I}}} + \\
+ N_\mathrm{r} \cdot \left(\underbrace{\frac{\mathrm{d}\overline{\Phi}_{\mathrm{h_{II}}}}{\mathrm{d}t} \cdot \mathrm{e}^{-\mathrm{j}\cdot p_\mathrm{p}\cdot\omega_\mathrm{m}\cdot t}}_{\frac{\mathrm{d}\overline{\Phi}_{\mathrm{h_I}}}{\mathrm{d}t}} + \underbrace{\overline{\Phi}_{\mathrm{h_{II}}} \cdot \mathrm{e}^{-\mathrm{j}\cdot p_\mathrm{p}\cdot\omega_\mathrm{m}\cdot t}}_{\overline{\Phi}_{\mathrm{h_I}}} \cdot (-\mathrm{j}\cdot p_\mathrm{p}\cdot\omega_\mathrm{m}) \right) + \\
+ N_\mathrm{r} \cdot \left(\underbrace{\frac{\mathrm{d}\overline{\Phi}_{\sigma\mathrm{r_{II}}}}{\mathrm{d}t} \cdot \mathrm{e}^{-\mathrm{j}\cdot p_\mathrm{p}\cdot\omega_\mathrm{m}\cdot t}}_{\frac{\mathrm{d}\overline{\Phi}_{\sigma\mathrm{r_I}}}{\mathrm{d}t}} + \underbrace{\overline{\Phi}_{\sigma\mathrm{r_{II}}} \cdot \mathrm{e}^{-\mathrm{j}\cdot p_\mathrm{p}\cdot\omega_\mathrm{m}\cdot t}}_{\overline{\Phi}_{\sigma\mathrm{r_I}}} \cdot (-\mathrm{j}\cdot p_\mathrm{p}\cdot\omega_\mathrm{m}) \right) \ .
\end{aligned} \tag{1.24}$$

Platí

$$\overline{i}_{\mathrm{r_I}} = \overline{i}_{\mathrm{r_{II}}} \cdot \mathrm{e}^{-\mathrm{j}\cdot p_\mathrm{p}\cdot\omega_\mathrm{m}\cdot t} \ , \tag{1.25}$$

$$\frac{\mathrm{d}\overline{\Phi}_{\mathrm{h_I}}}{\mathrm{d}t} = \frac{\mathrm{d}\overline{\Phi}_{\mathrm{h_{II}}}}{\mathrm{d}t} \cdot \mathrm{e}^{-\mathrm{j}\cdot p_\mathrm{p}\cdot\omega_\mathrm{m}\cdot t} \ , \tag{1.26}$$

$$\overline{\Phi}_{\mathrm{h_I}} = \overline{\Phi}_{\mathrm{h_{II}}} \cdot \mathrm{e}^{-\mathrm{j}\cdot p_\mathrm{p}\cdot\omega_\mathrm{m}\cdot t} \ , \tag{1.27}$$

$$\frac{\mathrm{d}\overline{\Phi}_{\sigma\mathrm{r_I}}}{\mathrm{d}t} = \frac{\mathrm{d}\overline{\Phi}_{\sigma\mathrm{r_{II}}}}{\mathrm{d}t} \cdot \mathrm{e}^{-\mathrm{j}\cdot p_\mathrm{p}\cdot\omega_\mathrm{m}\cdot t} \ , \tag{1.28}$$

$$\overline{\Phi}_{\sigma\mathrm{r_I}} = \overline{\Phi}_{\sigma\mathrm{r_{II}}} \cdot \mathrm{e}^{-\mathrm{j}\cdot p_\mathrm{p}\cdot\omega_\mathrm{m}\cdot t} \ . \tag{1.29}$$

Po dosazení

$$\overline{u}_{\mathrm{r_I}} = R_\mathrm{r} \cdot \overline{i}_{\mathrm{r_I}} + N_\mathrm{r} \cdot \left(\frac{\mathrm{d}\overline{\Phi}_{\mathrm{h_I}}}{\mathrm{d}t} - \mathrm{j}\cdot p_\mathrm{p}\cdot\omega_\mathrm{m}\cdot\overline{\Phi}_{\mathrm{h_I}} \right) + N_\mathrm{r} \cdot \left(\frac{\mathrm{d}\overline{\Phi}_{\sigma\mathrm{r_I}}}{\mathrm{d}t} - \mathrm{j}\cdot p_\mathrm{p}\cdot\omega_\mathrm{m}\cdot\overline{\Phi}_{\sigma\mathrm{r_I}} \right) \ . \tag{1.30}$$

Po roznásobení závorek

$$\overline{u}_{\mathrm{r_I}} = R_\mathrm{r} \cdot \overline{i}_{\mathrm{r_I}} + N_\mathrm{r} \cdot \frac{\mathrm{d}\overline{\Phi}_{\mathrm{h_I}}}{\mathrm{d}t} - \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \cdot N_\mathrm{r} \cdot \overline{\Phi}_{\mathrm{h_I}} + N_\mathrm{r} \cdot \frac{\mathrm{d}\overline{\Phi}_{\sigma\mathrm{r_I}}}{\mathrm{d}t} - \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \cdot N_\mathrm{r} \cdot \overline{\Phi}_{\sigma\mathrm{r_I}} \ . \quad (1.31)$$

Rotorová rovnice je nyní ve stejném souřadném systému jako statorová. Nyní je potřeba přepočítat rotorové veličiny na stator vynásobením obou stran rovnice výrazem $\frac{N_\mathrm{s}}{N_\mathrm{r}}$:

$$\begin{aligned}\overline{u}_{\mathrm{r_I}} \cdot \frac{N_\mathrm{s}}{N_\mathrm{r}} = R_\mathrm{r} \cdot \overline{i}_{\mathrm{r_I}} \cdot \frac{N_\mathrm{s}}{N_\mathrm{r}} + N_\mathrm{r} \cdot \frac{\mathrm{d}\overline{\Phi}_{\mathrm{h_I}}}{\mathrm{d}t} \cdot \frac{N_\mathrm{s}}{N_\mathrm{r}} - \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \cdot N_\mathrm{r} \cdot \overline{\Phi}_{\mathrm{h_I}} \cdot \frac{N_\mathrm{s}}{N_\mathrm{r}} + \\ + N_\mathrm{r} \cdot \frac{\mathrm{d}\overline{\Phi}_{\sigma\mathrm{r_I}}}{\mathrm{d}t} \cdot \frac{N_\mathrm{s}}{N_\mathrm{r}} - \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \cdot N_\mathrm{r} \cdot \overline{\Phi}_{\sigma\mathrm{r_I}} \cdot \frac{N_\mathrm{s}}{N_\mathrm{r}} \ .\end{aligned} \quad (1.32)$$

Po zkrácení

$$\underbrace{\overline{u}_{\mathrm{r_I}} \cdot \frac{N_\mathrm{s}}{N_\mathrm{r}}}_{\overline{u}'_{\mathrm{r_I}}} = \underbrace{R_\mathrm{r} \cdot \frac{N_\mathrm{s}}{N_\mathrm{r}}}_{R'_\mathrm{r}} \cdot \overline{i}_{\mathrm{r_I}} + N_\mathrm{s} \cdot \frac{\mathrm{d}\overline{\Phi}_{\mathrm{h_I}}}{\mathrm{d}t} - \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \cdot N_\mathrm{s} \cdot \overline{\Phi}_{\mathrm{h_I}} + N_\mathrm{s} \cdot \frac{\mathrm{d}\overline{\Phi}_{\sigma\mathrm{r_I}}}{\mathrm{d}t} - \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \cdot N_\mathrm{s} \cdot \overline{\Phi}_{\sigma\mathrm{r_I}} \ . \quad (1.33)$$

Platí

$$\overline{u}'_{\mathrm{r_I}} = \overline{u}_{\mathrm{r_I}} \cdot \frac{N_\mathrm{s}}{N_\mathrm{r}} \ , \quad (1.34)$$

$$R'_\mathrm{r} = R_\mathrm{r} \cdot \frac{N_\mathrm{s}}{N_\mathrm{r}} \ . \quad (1.35)$$

Po dosazení

$$\overline{u}'_{\mathrm{r_I}} = R'_\mathrm{r} \cdot \overline{i}_{\mathrm{r_I}} + N_\mathrm{s} \cdot \frac{\mathrm{d}\overline{\Phi}_{\mathrm{h_I}}}{\mathrm{d}t} - \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \cdot N_\mathrm{s} \cdot \overline{\Phi}_{\mathrm{h_I}} + N_\mathrm{s} \cdot \frac{\mathrm{d}\overline{\Phi}_{\sigma\mathrm{r_I}}}{\mathrm{d}t} - \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \cdot N_\mathrm{s} \cdot \overline{\Phi}_{\sigma\mathrm{r_I}} \ . \quad (1.36)$$

Výrazy N_s a $-\mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \cdot N_\mathrm{s}$ je možné vytknout před závorky

$$\overline{u}'_{\mathrm{r_I}} = R'_\mathrm{r} \cdot \overline{i}_{\mathrm{r_I}} - \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \cdot N_\mathrm{s} \cdot \left(\overline{\Phi}_{\mathrm{h_I}} + \overline{\Phi}_{\sigma\mathrm{r_I}}\right) + N_\mathrm{s} \cdot \left(\frac{\mathrm{d}\overline{\Phi}_{\mathrm{h_I}}}{\mathrm{d}t} + \frac{\mathrm{d}\overline{\Phi}_{\sigma\mathrm{r_I}}}{\mathrm{d}t}\right) \ . \quad (1.37)$$

Ze vztahů (1.18) a (1.19) vyplývá

$$\overline{\Phi}_\mathrm{s} = \overline{\Phi}_\mathrm{h} + \overline{\Phi}_{\sigma\mathrm{s}} \ , \quad (1.38)$$

$$\overline{\Phi}_\mathrm{r} = \overline{\Phi}_\mathrm{h} + \overline{\Phi}_{\sigma\mathrm{r}} \ . \quad (1.39)$$

Podle vztahu (1.39) je možné upravit rovnici do tvaru

$$\overline{u}'_{\mathrm{r_I}} = R'_\mathrm{r} \cdot \overline{i}_{\mathrm{r_I}} - \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \cdot \underbrace{N_\mathrm{s} \cdot \overline{\Phi}_{\mathrm{r_I}}}_{\overline{\Psi}'_{\mathrm{r_I}}} + \underbrace{N_\mathrm{s} \cdot \frac{\mathrm{d}\overline{\Phi}_{\mathrm{r_I}}}{\mathrm{d}t}}_{\frac{\mathrm{d}\overline{\Psi}'_{\mathrm{r_I}}}{\mathrm{d}t}} \ . \quad (1.40)$$

Platí

$$\overline{\Psi}'_{\mathrm{r_I}} = N_\mathrm{s} \cdot \overline{\Phi}_{\mathrm{r_I}} \ , \quad (1.41)$$

$$\frac{\mathrm{d}\overline{\Psi}'_{\mathrm{r_I}}}{\mathrm{d}t} = N_\mathrm{s} \cdot \frac{\mathrm{d}\overline{\Phi}_{\mathrm{r_I}}}{\mathrm{d}t} \ . \quad (1.42)$$

Po dosazení

$$\overline{u}'_{\mathrm{r_I}} = R'_{\mathrm{r}} \cdot \bar{i}_{\mathrm{r_I}} - \mathrm{j} \cdot p_{\mathrm{p}} \cdot \omega_{\mathrm{m}} \cdot \overline{\Psi}'_{\mathrm{r_I}} + \frac{\mathrm{d}\overline{\Psi}'_{\mathrm{r_I}}}{\mathrm{d}t} \ . \tag{1.43}$$

Ze vztahů (1.18) a (1.19) vyplývá, že magnetické toky statoru a rotoru lze vyjádřit jako

$$\overline{\Psi}_{\mathrm{s_I}} = \overline{\Psi}_{\mathrm{h_I}} + \overline{\Psi}_{\sigma\mathrm{s_I}} \ , \tag{1.44}$$

$$\overline{\Psi}'_{\mathrm{r_I}} = \overline{\Psi}_{\mathrm{h_I}} + \overline{\Psi}'_{\sigma\mathrm{r_I}} \ . \tag{1.45}$$

Hlavní magnetický tok $\overline{\Psi}_{\mathrm{h_I}}$ lze vyjádřit jako součet příspěvků od statoru a rotoru:

$$\overline{\Psi}_{\mathrm{h_I}} = \overline{\Psi}_{\mathrm{hs_I}} + \overline{\Psi}_{\mathrm{hr_I}} \ . \tag{1.46}$$

Příspěvek statoru do hlavního toku lze chápat jako magnetický tok, který vzniká pomocí statorového proudu zabírajícího s hlavní indukčností. Obdobně lze chápat rotorový příspěvek do hlavního toku. Lze tedy psát

$$\overline{\Psi}_{\mathrm{hs_I}} = L_{\mathrm{h}} \cdot \bar{i}_{\mathrm{s_I}} \ , \tag{1.47}$$

$$\overline{\Psi}_{\mathrm{hr_I}} = L_{\mathrm{h}} \cdot \bar{i}_{\mathrm{r_I}} \ . \tag{1.48}$$

Dosazením do (1.46) vychází vztah

$$\overline{\Psi}_{\mathrm{h_I}} = L_{\mathrm{h}} \cdot \left(\bar{i}_{\mathrm{s_I}} + \bar{i}_{\mathrm{r_I}}\right) \ . \tag{1.49}$$

Rozptylové toky statoru a rotoru lze vyjádřit jako

$$\overline{\Psi}_{\sigma\mathrm{s_I}} = L_{\sigma\mathrm{s}} \cdot \bar{i}_{\mathrm{s_I}} \ , \tag{1.50}$$

$$\overline{\Psi}'_{\sigma\mathrm{r_I}} = L'_{\sigma\mathrm{r}} \cdot \bar{i}_{\mathrm{r_I}} \ . \tag{1.51}$$

Dosazením (1.49) a (1.51) do (1.45)

$$\overline{\Psi}'_{\mathrm{r_I}} = L_{\mathrm{h}} \cdot \left(\bar{i}_{\mathrm{s_I}} + \bar{i}_{\mathrm{r_I}}\right) + L'_{\sigma\mathrm{r}} \cdot \bar{i}_{\mathrm{r_I}} \ . \tag{1.52}$$

Zavedením proměnné magnetizační proud $\bar{i}_{\mu_\mathrm{I}}$, která je dána součtem statorového a rotorového proudu (ve společném souřadném systému a po přepočtu rotorových veličin na stator):

$$\bar{i}_{\mu_\mathrm{I}} = \bar{i}_{\mathrm{s_I}} + \bar{i}_{\mathrm{r_I}} \ , \tag{1.53}$$

lze vztah (1.52) upravit do tvaru

$$\overline{\Psi}'_{\mathrm{r_I}} = L_{\mathrm{h}} \cdot \bar{i}_{\mu_\mathrm{I}} + L'_{\sigma\mathrm{r}} \cdot \bar{i}_{\mathrm{r_I}} \ . \tag{1.54}$$

Rovnici pro rotorové napětí (1.43) je nyní možné pomocí vztahů (1.52) a (1.54) upravit do tvaru

$$\overline{u}'_{\mathrm{r_I}} = R'_{\mathrm{r}} \cdot \bar{i}_{\mathrm{r_I}} - \mathrm{j} \cdot p_{\mathrm{p}} \cdot \omega_{\mathrm{m}} \cdot \left(L_{\mathrm{h}} \cdot \left(\bar{i}_{\mathrm{s_I}} + \bar{i}_{\mathrm{r_I}}\right) + L'_{\sigma\mathrm{r}} \cdot \bar{i}_{\mathrm{r_I}}\right) + L_{\mathrm{h}} \cdot \frac{\mathrm{d}\bar{i}_{\mu_\mathrm{I}}}{\mathrm{d}t} + L'_{\sigma\mathrm{r}} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{r_I}}}{\mathrm{d}t} \ . \tag{1.55}$$

Po roznásobení vnitřní závorky:

$$\overline{u}'_{\mathrm{r_I}} = R'_{\mathrm{r}} \cdot \bar{i}_{\mathrm{r_I}} - \mathrm{j} \cdot p_{\mathrm{p}} \cdot \omega_{\mathrm{m}} \cdot \big(L_{\mathrm{h}} \cdot \bar{i}_{\mathrm{s_I}} + \underbrace{L_{\mathrm{h}} \cdot \bar{i}_{\mathrm{r_I}} + L'_{\sigma\mathrm{r}} \cdot \bar{i}_{\mathrm{r_I}}}_{L'_{\mathrm{r}} \cdot \bar{i}_{\mathrm{r_I}}}\big) + L_{\mathrm{h}} \cdot \frac{\mathrm{d}\bar{i}_{\mu_\mathrm{I}}}{\mathrm{d}t} + L'_{\sigma\mathrm{r}} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{r_I}}}{\mathrm{d}t} \ . \quad (1.56)$$

Platí

$$L_{\mathrm{h}} \cdot \bar{i}_{\mathrm{r_I}} + L'_{\sigma\mathrm{r}} \cdot \bar{i}_{\mathrm{r_I}} = (L_{\mathrm{h}} + L'_{\sigma\mathrm{r}}) \cdot \bar{i}_{\mathrm{r_I}} = L'_{\mathrm{r}} \cdot \bar{i}_{\mathrm{r_I}} \ . \quad (1.57)$$

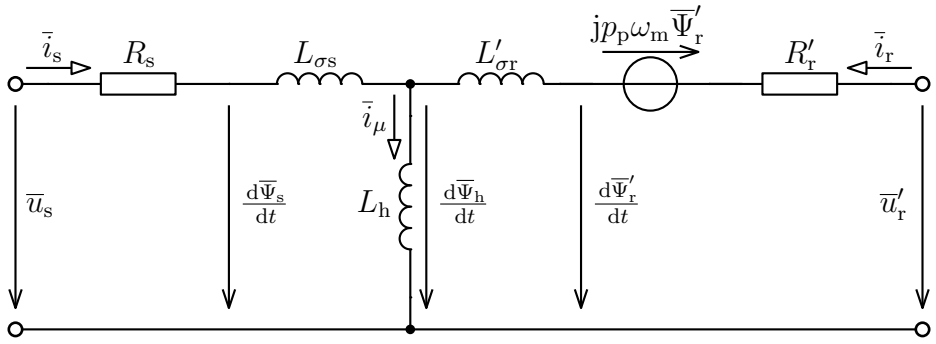
Na konec zbývá upravit rovnici pro napětí na statoru (1.16). Po rozdělení statorového toku $\overline{\Psi}_{\mathrm{s_I}}$ na hlavní tok a statorový rozptyl je možné obdobně jako u rotorového napětí tyto dílčí toky vyjádřit pomocí statorového a magnetizačního proudu.

Po těchto úpravách je získán matematický model asynchronního stroje ve stojícím souřadném systému α, β: [17][24]

$$\overline{u}_{\mathrm{s_I}} = R_{\mathrm{s}} \cdot \bar{i}_{\mathrm{s_I}} + L_{\mathrm{h}} \cdot \frac{\mathrm{d}\bar{i}_{\mu_\mathrm{I}}}{\mathrm{d}t} + L_{\sigma\mathrm{s}} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{s_I}}}{\mathrm{d}t} \ , \quad (1.58)$$

$$\overline{u}'_{\mathrm{r_I}} = R'_{\mathrm{r}} \cdot \bar{i}_{\mathrm{r_I}} - \mathrm{j} \cdot p_{\mathrm{p}} \cdot \omega_{\mathrm{m}} \cdot \big(L_{\mathrm{h}} \cdot \bar{i}_{\mathrm{s_I}} + L'_{\mathrm{r}} \cdot \bar{i}_{\mathrm{r_I}}\big) + L_{\mathrm{h}} \cdot \frac{\mathrm{d}\bar{i}_{\mu_\mathrm{I}}}{\mathrm{d}t} + L'_{\sigma\mathrm{r}} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{r_I}}}{\mathrm{d}t} \ . \quad (1.59)$$

Na základě rovnic (1.58) a (1.59) lze pomocí 2. Kirchhoffova zákona sestavit náhradní schéma asynchronního stroje. [17][24]



Obr. 1.3: Náhradní schéma asynchronního stroje

1.2 Odvození modelu asynchronního stroje pro počítačové simulace

Pro provádění počítačových simulací dynamických systémů, mezi které patří také elektrické stroje, se velmi často používá stavového popisu. Při něm je systém popsán sadou diferenciálních rovnic prvního řádu, kde počet stavů systému odpovídá počtu diferenciálních rovnic. Na základě znalosti vstupů a předchozího stavu systému je tak možné určit chování systému v následujícím okamžiku. Problematikou dynamických systémů se zabývá např. [21] nebo [22].

Před sestavováním stavového popisu je potřeba nejprve stanovit, co je stavem systému. V případě střídavých elektrických strojů jsou potřeba pro sestavení stavového popisu minimálně dvě komplexní stavové diferenciální rovnice a jedna pohybová stavová rovnice. Jako komplexní stavové proměnné se velmi často volí dvojice

- statorový proud $\overline{i}_{\mathrm{s_I}}$ a rotorový proud $\overline{i}_{\mathrm{r_I}}$,
- statorový proud $\overline{i}_{\mathrm{s_I}}$ a rotorový magnetický tok $\overline{\Psi}'_{\mathrm{r_I}}$,
- statorový proud $\overline{i}_{\mathrm{s_I}}$ a statorový magnetický tok $\overline{\Psi}_{\mathrm{s_I}}$.

V tomto případě byly jako stavové veličiny zvoleny statorový proud $\overline{i}_{\mathrm{s_I}}$, rotorový magnetický tok $\overline{\Psi}'_{\mathrm{r_I}}$ a mechanická úhlová rychlost rotoru ω_{m}. Po rozepsání komplexních stavových rovnic do složek α a β bude tedy celkem pět stavových rovnic.

Z důvodu lepší přehlednosti je vhodné před samotným odvozením definovat několik konstant: [24]

$$\lambda = L_{\mathrm{s}} - \frac{{L_{\mathrm{h}}}^2}{L'_{\mathrm{r}}} \ , \tag{1.60}$$

$$\alpha = \frac{R_{\mathrm{s}} + R'_{\mathrm{r}} \cdot \frac{{L_{\mathrm{h}}}^2}{{L'_{\mathrm{r}}}^2}}{\lambda} \ , \tag{1.61}$$

$$\beta = \frac{R'_{\mathrm{r}} \cdot \frac{L_{\mathrm{h}}}{{L'_{\mathrm{r}}}^2}}{\lambda} \ , \tag{1.62}$$

$$\gamma = \frac{p_{\mathrm{p}} \cdot \frac{L_{\mathrm{h}}}{L'_{\mathrm{r}}}}{\lambda} \ , \tag{1.63}$$

$$\delta = \frac{1}{\lambda} \ , \tag{1.64}$$

$$\varepsilon = \frac{k_{\mathrm{p}} \cdot p_{\mathrm{p}}}{J} \cdot \frac{L_{\mathrm{h}}}{L'_{\mathrm{r}}} \ . \tag{1.65}$$

1.2.1 Odvození komplexní stavové diferenciální rovnice pro rotorový tok

Nejprve je vhodné odvodit rovnici pro magnetický tok rotoru. Je zřejmé, že platí rovnost vztahů (1.43) a (1.59). Lze tedy říci, že platí

$$\overline{\Psi}'_{\mathrm{r_I}} = L_{\mathrm{h}} \cdot \overline{i}_{\mathrm{s_I}} + L'_{\mathrm{r}} \cdot \overline{i}_{\mathrm{r_I}} \ . \tag{1.66}$$

Z této rovnice lze vyjádřit rotorový proud

$$\overline{i}_{\mathrm{r_I}} = \frac{\overline{\Psi}'_{\mathrm{r_I}}}{L'_{\mathrm{r}}} - \frac{L_{\mathrm{h}}}{L'_{\mathrm{r}}} \cdot \overline{i}_{\mathrm{s_I}} \tag{1.67}$$

a dosadit jej do vztahu (1.43):

$$\overline{u}'_{\mathrm{r_I}} = R'_{\mathrm{r}} \cdot \left(\frac{\overline{\Psi}'_{\mathrm{r_I}}}{L'_{\mathrm{r}}} - \frac{L_{\mathrm{h}}}{L'_{\mathrm{r}}} \cdot \overline{i}_{\mathrm{s_I}} \right) - \mathrm{j} \cdot p_{\mathrm{p}} \cdot \omega_{\mathrm{m}} \cdot \overline{\Psi}'_{\mathrm{r_I}} + \frac{\mathrm{d}\overline{\Psi}'_{\mathrm{r_I}}}{\mathrm{d}t} \ . \tag{1.68}$$

Nyní je vhodné provést několik úprav. Nejprve je potřeba roznásobit závorku

$$\overline{u}'_{\mathrm{r_I}} = \frac{R'_\mathrm{r}}{L'_\mathrm{r}} \cdot \overline{\Psi}'_{\mathrm{r_I}} - R'_\mathrm{r} \cdot \frac{L_\mathrm{h}}{L'_\mathrm{r}} \cdot \bar{i}_{\mathrm{s_I}} - \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \cdot \overline{\Psi}'_{\mathrm{r_I}} + \frac{\mathrm{d}\overline{\Psi}'_{\mathrm{r_I}}}{\mathrm{d}t} \tag{1.69}$$

a následně vytknout rotorový tok $\overline{\Psi}'_{\mathrm{r_I}}$

$$\overline{u}'_{\mathrm{r_I}} = -R'_\mathrm{r} \cdot \frac{L_\mathrm{h}}{L'_\mathrm{r}} \cdot \bar{i}_{\mathrm{s_I}} + \left(\frac{R'_\mathrm{r}}{L'_\mathrm{r}} - \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \right) \cdot \overline{\Psi}'_{\mathrm{r_I}} + \frac{\mathrm{d}\overline{\Psi}'_{\mathrm{r_I}}}{\mathrm{d}t} \ . \tag{1.70}$$

Po těchto úpravách zbývá vyjádřit výraz $\frac{\mathrm{d}\overline{\Psi}'_{\mathrm{r_I}}}{\mathrm{d}t}$, čímž je získána první komplexní stavová diferenciální rovnice [17][24]

$$\boxed{\frac{\mathrm{d}\overline{\Psi}'_{\mathrm{r_I}}}{\mathrm{d}t} = R'_\mathrm{r} \cdot \frac{L_\mathrm{h}}{L'_\mathrm{r}} \cdot \bar{i}_{\mathrm{s_I}} - \left(\frac{R'_\mathrm{r}}{L'_\mathrm{r}} - \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \right) \cdot \overline{\Psi}'_{\mathrm{r_I}} + \overline{u}'_{\mathrm{r_I}} \ .} \tag{1.71}$$

Pro následné rozepsání rovnice do složek α a β je vhodné rozdělit člen s $\overline{\Psi}'_{\mathrm{r_I}}$ na reálnou a imaginární část:

$$\frac{\mathrm{d}\overline{\Psi}'_{\mathrm{r_I}}}{\mathrm{d}t} = R'_\mathrm{r} \cdot \frac{L_\mathrm{h}}{L'_\mathrm{r}} \cdot \bar{i}_{\mathrm{s_I}} - \frac{R'_\mathrm{r}}{L'_\mathrm{r}} \cdot \overline{\Psi}'_{\mathrm{r_I}} + \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \cdot \overline{\Psi}'_{\mathrm{r_I}} + \overline{u}'_{\mathrm{r_I}} \ . \tag{1.72}$$

Složky α a β rotorového toku po rozepsání předchozí rovnice budou [17][24]

$$\boxed{\frac{\mathrm{d}\Psi'_{\mathrm{r}_\alpha}}{\mathrm{d}t} = R'_\mathrm{r} \cdot \frac{L_\mathrm{h}}{L'_\mathrm{r}} \cdot i_{\mathrm{s}_\alpha} - \frac{R'_\mathrm{r}}{L'_\mathrm{r}} \cdot \Psi'_{\mathrm{r}_\alpha} - p_\mathrm{p} \cdot \omega_\mathrm{m} \cdot \Psi'_{\mathrm{r}_\beta} + u'_{\mathrm{r}_\alpha} \ ,} \tag{1.73}$$

$$\boxed{\frac{\mathrm{d}\Psi'_{\mathrm{r}_\beta}}{\mathrm{d}t} = R'_\mathrm{r} \cdot \frac{L_\mathrm{h}}{L'_\mathrm{r}} \cdot i_{\mathrm{s}_\beta} + p_\mathrm{p} \cdot \omega_\mathrm{m} \cdot \Psi'_{\mathrm{r}_\alpha} - \frac{R'_\mathrm{r}}{L'_\mathrm{r}} \cdot \Psi'_{\mathrm{r}_\beta} + u'_{\mathrm{r}_\beta} \ .} \tag{1.74}$$

1.2.2 Odvození komplexní stavové diferenciální rovnice pro statorový proud

Před odvozením druhé komplexní stavové rovnice je nejprve potřeba pomocí vztahu (1.53) upravit rovnici (1.58):

$$\overline{u}_{\mathrm{s_I}} = R_\mathrm{s} \cdot \bar{i}_{\mathrm{s_I}} + L_\mathrm{h} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{s_I}}}{\mathrm{d}t} + L_\mathrm{h} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{r_I}}}{\mathrm{d}t} + L_{\sigma\mathrm{s}} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{s_I}}}{\mathrm{d}t} \ . \tag{1.75}$$

Statorová indukčnost je dána vztahem $L_\mathrm{s} = L_\mathrm{h} + L_{\sigma\mathrm{s}}$, na jehož základě je možné upravit předchozí rovnici do tvaru

$$\overline{u}_{\mathrm{s_I}} = R_\mathrm{s} \cdot \bar{i}_{\mathrm{s_I}} + L_\mathrm{h} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{s_I}}}{\mathrm{d}t} + L_\mathrm{h} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{r_I}}}{\mathrm{d}t} + (L_\mathrm{s} - L_\mathrm{h}) \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{s_I}}}{\mathrm{d}t} \ . \tag{1.76}$$

Nyní je nutné vyjádřit pomocí stavových veličin výraz $L_\mathrm{h} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{r_I}}}{\mathrm{d}t}$. Derivací vztahu (1.67) vznikne rovnice

$$\frac{\mathrm{d}\bar{i}_{\mathrm{r_I}}}{\mathrm{d}t} = \frac{1}{L'_\mathrm{r}} \cdot \frac{\mathrm{d}\overline{\Psi}'_{\mathrm{r_I}}}{\mathrm{d}t} - \frac{L_\mathrm{h}}{L'_\mathrm{r}} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{s_I}}}{\mathrm{d}t} \ . \tag{1.77}$$

Tento vztah nyní stačí pouze vynásobit výrazem L_h:

$$L_\mathrm{h} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{r_I}}}{\mathrm{d}t} = \frac{L_\mathrm{h}}{L'_\mathrm{r}} \cdot \frac{\mathrm{d}\overline{\Psi}'_{\mathrm{r_I}}}{\mathrm{d}t} - \frac{{L_\mathrm{h}}^2}{L'_\mathrm{r}} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{s_I}}}{\mathrm{d}t} \ . \tag{1.78}$$

Výraz $\frac{\mathrm{d}\overline{\Psi}'_{\mathrm{r_I}}}{\mathrm{d}t}$ je již vyjádřen první komplexní stavovou rovnicí (1.71). Do předchozího vztahu lze tedy rovnou dosadit

$$L_\mathrm{h} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{r_I}}}{\mathrm{d}t} = \frac{L_\mathrm{h}}{L'_\mathrm{r}} \cdot \left(R'_\mathrm{r} \cdot \frac{L_\mathrm{h}}{L'_\mathrm{r}} \cdot \bar{i}_{\mathrm{s_I}} - \left(\frac{R'_\mathrm{r}}{L'_\mathrm{r}} - \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \right) \cdot \overline{\Psi}'_{\mathrm{r_I}} + \overline{u}'_{\mathrm{r_I}} \right) - \frac{{L_\mathrm{h}}^2}{L'_\mathrm{r}} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{s_I}}}{\mathrm{d}t} \ . \tag{1.79}$$

Tuto rovnici je vhodné ještě upravit roznásobením vnější závorky

$$L_\mathrm{h} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{r_I}}}{\mathrm{d}t} = R'_\mathrm{r} \cdot \frac{{L_\mathrm{h}}^2}{{L'_\mathrm{r}}^2} \cdot \bar{i}_{\mathrm{s_I}} - \frac{L_\mathrm{h}}{L'_\mathrm{r}} \cdot \left(\frac{R'_\mathrm{r}}{L'_\mathrm{r}} - \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \right) \cdot \overline{\Psi}'_{\mathrm{r_I}} + \frac{L_\mathrm{h}}{L'_\mathrm{r}} \cdot \overline{u}'_{\mathrm{r_I}} - \frac{{L_\mathrm{h}}^2}{L'_\mathrm{r}} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{s_I}}}{\mathrm{d}t} \tag{1.80}$$

a vynásobením vnitřku zbývající závorky výrazem $\frac{L_\mathrm{h}}{L'_\mathrm{r}}$

$$L_\mathrm{h} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{r_I}}}{\mathrm{d}t} = R'_\mathrm{r} \cdot \frac{{L_\mathrm{h}}^2}{{L'_\mathrm{r}}^2} \cdot \bar{i}_{\mathrm{s_I}} - \left(R'_\mathrm{r} \cdot \frac{L_\mathrm{h}}{{L'_\mathrm{r}}^2} - \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \cdot \frac{L_\mathrm{h}}{L'_\mathrm{r}} \right) \cdot \overline{\Psi}'_{\mathrm{r_I}} + \frac{L_\mathrm{h}}{L'_\mathrm{r}} \cdot \overline{u}'_{\mathrm{r_I}} - \frac{{L_\mathrm{h}}^2}{L'_\mathrm{r}} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{s_I}}}{\mathrm{d}t} \ . \tag{1.81}$$

Nyní lze dosadit do rovnice (1.76)

$$\begin{aligned} \overline{u}_{\mathrm{s_I}} = R_\mathrm{s} \cdot \bar{i}_{\mathrm{s_I}} + L_\mathrm{h} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{s_I}}}{\mathrm{d}t} + R'_\mathrm{r} \cdot \frac{{L_\mathrm{h}}^2}{{L'_\mathrm{r}}^2} \cdot \bar{i}_{\mathrm{s_I}} - \left(R'_\mathrm{r} \cdot \frac{L_\mathrm{h}}{{L'_\mathrm{r}}^2} - \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \cdot \frac{L_\mathrm{h}}{L'_\mathrm{r}} \right) \cdot \overline{\Psi}'_{\mathrm{r_I}} + \\ + \frac{L_\mathrm{h}}{L'_\mathrm{r}} \cdot \overline{u}'_{\mathrm{r_I}} - \frac{{L_\mathrm{h}}^2}{L'_\mathrm{r}} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{s_I}}}{\mathrm{d}t} + (L_\mathrm{s} - L_\mathrm{h}) \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{s_I}}}{\mathrm{d}t} \end{aligned} \tag{1.82}$$

a před závorky vytknout výrazy $\bar{i}_{\mathrm{s_I}}$ a $\frac{\mathrm{d}\bar{i}_{\mathrm{s_I}}}{\mathrm{d}t}$

$$\begin{aligned} \overline{u}_{\mathrm{s_I}} = \left(R_\mathrm{s} + R'_\mathrm{r} \cdot \frac{{L_\mathrm{h}}^2}{{L'_\mathrm{r}}^2} \right) \cdot \bar{i}_{\mathrm{s_I}} - \left(R'_\mathrm{r} \cdot \frac{L_\mathrm{h}}{{L'_\mathrm{r}}^2} - \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \cdot \frac{L_\mathrm{h}}{L'_\mathrm{r}} \right) \cdot \overline{\Psi}'_{\mathrm{r_I}} + \frac{L_\mathrm{h}}{L'_\mathrm{r}} \cdot \overline{u}'_{\mathrm{r_I}} + \\ + \underbrace{\left(L_\mathrm{h} - \frac{{L_\mathrm{h}}^2}{L'_\mathrm{r}} + L_\mathrm{s} - L_\mathrm{h} \right)}_{L_\mathrm{s} - \frac{{L_\mathrm{h}}^2}{L'_\mathrm{r}}} \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{s_I}}}{\mathrm{d}t} \ . \end{aligned} \tag{1.83}$$

Z předchozího vztahu lze vyjádřit člen obsahující výraz $\frac{\mathrm{d}\bar{i}_{\mathrm{s_I}}}{\mathrm{d}t}$:

$$\begin{aligned} \left(L_\mathrm{s} - \frac{{L_\mathrm{h}}^2}{L'_\mathrm{r}} \right) \cdot \frac{\mathrm{d}\bar{i}_{\mathrm{s_I}}}{\mathrm{d}t} = - \left(R_\mathrm{s} + R'_\mathrm{r} \cdot \frac{{L_\mathrm{h}}^2}{{L'_\mathrm{r}}^2} \right) \cdot \bar{i}_{\mathrm{s_I}} + \\ + \left(R'_\mathrm{r} \cdot \frac{L_\mathrm{h}}{{L'_\mathrm{r}}^2} - \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \cdot \frac{L_\mathrm{h}}{L'_\mathrm{r}} \right) \cdot \overline{\Psi}'_{\mathrm{r_I}} - \frac{L_\mathrm{h}}{L'_\mathrm{r}} \cdot \overline{u}'_{\mathrm{r_I}} + \overline{u}_{\mathrm{s_I}} \ . \end{aligned} \tag{1.84}$$

Druhá komplexní stavová diferenciální rovnice je tedy [17][24]

$$\boxed{\begin{aligned}\frac{\mathrm{d}\bar{i}_{\mathrm{s_I}}}{\mathrm{d}t} = &- \frac{R_\mathrm{s} + R'_\mathrm{r} \cdot \dfrac{{L_\mathrm{h}}^2}{{L'_\mathrm{r}}^2}}{L_\mathrm{s} - \dfrac{{L_\mathrm{h}}^2}{L'_\mathrm{r}}} \cdot \bar{i}_{\mathrm{s_I}} + \frac{R'_\mathrm{r} \cdot \dfrac{L_\mathrm{h}}{{L'_\mathrm{r}}^2} - \mathrm{j} \cdot p_\mathrm{p} \cdot \omega_\mathrm{m} \cdot \dfrac{L_\mathrm{h}}{L'_\mathrm{r}}}{L_\mathrm{s} - \dfrac{{L_\mathrm{h}}^2}{L'_\mathrm{r}}} \cdot \overline{\Psi}'_{\mathrm{r_I}} - \\ &- \frac{L_\mathrm{h}}{L'_\mathrm{r} \cdot \left(L_\mathrm{s} - \dfrac{{L_\mathrm{h}}^2}{L'_\mathrm{r}}\right)} \cdot \overline{u}'_{\mathrm{r_I}} + \frac{1}{L_\mathrm{s} - \dfrac{{L_\mathrm{h}}^2}{L'_\mathrm{r}}} \cdot \overline{u}_{\mathrm{s_I}} \ .\end{aligned}} \tag{1.85}$$

Pro následné rozepsání rovnice do složek α a β je opět vhodné rozdělit člen s $\overline{\Psi}'_{\mathrm{r_I}}$ na reálnou a imaginární část:

$$\begin{aligned}\frac{\mathrm{d}\bar{i}_{\mathrm{s_I}}}{\mathrm{d}t} = &- \underbrace{\frac{R_\mathrm{s} + R'_\mathrm{r} \cdot \dfrac{{L_\mathrm{h}}^2}{{L'_\mathrm{r}}^2}}{L_\mathrm{s} - \dfrac{{L_\mathrm{h}}^2}{L'_\mathrm{r}}}}_{\alpha} \cdot \bar{i}_{\mathrm{s_I}} + \underbrace{\frac{R'_\mathrm{r} \cdot \dfrac{L_\mathrm{h}}{{L'_\mathrm{r}}^2}}{L_\mathrm{s} - \dfrac{{L_\mathrm{h}}^2}{L'_\mathrm{r}}}}_{\beta} \cdot \overline{\Psi}'_{\mathrm{r_I}} - \mathrm{j} \cdot \underbrace{\frac{p_\mathrm{p} \cdot \dfrac{L_\mathrm{h}}{L'_\mathrm{r}}}{L_\mathrm{s} - \dfrac{{L_\mathrm{h}}^2}{L'_\mathrm{r}}}}_{\gamma} \cdot \omega_\mathrm{m} \cdot \overline{\Psi}'_{\mathrm{r_I}} - \\ &- \frac{L_\mathrm{h}}{L'_\mathrm{r}} \cdot \underbrace{\frac{1}{L_\mathrm{s} - \dfrac{{L_\mathrm{h}}^2}{L'_\mathrm{r}}}}_{\delta} \cdot \overline{u}'_{\mathrm{r_I}} + \underbrace{\frac{1}{L_\mathrm{s} - \dfrac{{L_\mathrm{h}}^2}{L'_\mathrm{r}}}}_{\delta} \cdot \overline{u}_{\mathrm{s_I}} \ .\end{aligned} \tag{1.86}$$

Dosazením dříve definovaných konstant (1.60) až (1.64) lze upravit předchozí vztah do úhlednějšího tvaru

$$\frac{\mathrm{d}\bar{i}_{\mathrm{s_I}}}{\mathrm{d}t} = -\alpha \cdot \bar{i}_{\mathrm{s_I}} + \beta \cdot \overline{\Psi}'_{\mathrm{r_I}} - \mathrm{j} \cdot \gamma \cdot \omega_\mathrm{m} \cdot \overline{\Psi}'_{\mathrm{r_I}} - \frac{L_\mathrm{h}}{L'_\mathrm{r}} \cdot \delta \cdot \overline{u}'_{\mathrm{r_I}} + \delta \cdot \overline{u}_{\mathrm{s_I}} \ . \tag{1.87}$$

Složky α a β statorového proudu po rozepsání předchozí rovnice budou [17][24]

$$\boxed{\begin{aligned}\frac{\mathrm{d}i_{\mathrm{s}_\alpha}}{\mathrm{d}t} &= -\alpha \cdot i_{\mathrm{s}_\alpha} + \beta \cdot \Psi'_{\mathrm{r}_\alpha} + \gamma \cdot \omega_\mathrm{m} \cdot \Psi'_{\mathrm{r}_\beta} - \frac{L_\mathrm{h}}{L'_\mathrm{r}} \cdot \delta \cdot u'_{\mathrm{r}_\alpha} + \delta \cdot u_{\mathrm{s}_\alpha} \ , \qquad (1.88) \\ \frac{\mathrm{d}i_{\mathrm{s}_\beta}}{\mathrm{d}t} &= -\alpha \cdot i_{\mathrm{s}_\beta} - \gamma \cdot \omega_\mathrm{m} \cdot \Psi'_{\mathrm{r}_\alpha} + \beta \cdot \Psi'_{\mathrm{r}_\beta} - \frac{L_\mathrm{h}}{L'_\mathrm{r}} \cdot \delta \cdot u'_{\mathrm{r}_\beta} + \delta \cdot u_{\mathrm{s}_\beta} \ . \qquad (1.89)\end{aligned}}$$

1.2.3 Odvození pohybové stavové diferenciální rovnice

Při odvozování stavové rovnice pro mechanickou úhlovou rychlost rotoru lze vyjít z následujícího vztahu [17]

$$J \cdot \frac{\mathrm{d}\omega_\mathrm{m}}{\mathrm{d}t} = M - M_\mathrm{z} - B_{\omega_\mathrm{m}} \ , \tag{1.90}$$

kde:

J $(\mathrm{kg}\cdot\mathrm{m}^2)$ – moment setrvačnosti,
ω_m $(\mathrm{rad}\cdot\mathrm{s}^{-1})$ – mechanická úhlová rychlost rotoru,
M $(\mathrm{N}\cdot\mathrm{m})$ – moment motoru,
M_z $(\mathrm{N}\cdot\mathrm{m})$ – moment zátěže,
B_{ω_m} $(\mathrm{N}\cdot\mathrm{m})$ – koeficient tření.

Nejprve je potřeba z rovnice (1.90) vyjádřit výraz $\frac{\mathrm{d}\omega_\mathrm{m}}{\mathrm{d}t}$:

$$\frac{\mathrm{d}\omega_\mathrm{m}}{\mathrm{d}t} = \frac{M - M_\mathrm{z} - B_{\omega_\mathrm{m}}}{J} \ . \tag{1.91}$$

Moment je nutné vyjádřit pomocí stavových veličin. K tomu lze využít vztah

$$M = k_\mathrm{p}\cdot p_\mathrm{p}\cdot\frac{L_\mathrm{h}}{L'_\mathrm{r}}\cdot\begin{bmatrix} -\Psi'_{\mathrm{r}\beta} & \Psi'_{\mathrm{r}\alpha}\end{bmatrix}\cdot\begin{bmatrix} i_{\mathrm{s}\alpha}\\ i_{\mathrm{s}\beta}\end{bmatrix} \ . \tag{1.92}$$

Detailní odvození tohoto vztahu je provedeno v [24]. Provedením maticového součinu vychází

$$M = k_\mathrm{p}\cdot p_\mathrm{p}\cdot\frac{L_\mathrm{h}}{L'_\mathrm{r}}\cdot\left(-\Psi'_{\mathrm{r}\beta}\cdot i_{\mathrm{s}\alpha} + \Psi'_{\mathrm{r}\alpha}\cdot i_{\mathrm{s}\beta}\right) \tag{1.93}$$

a po roznásobení závorky je výsledek

$$M = -k_\mathrm{p}\cdot p_\mathrm{p}\cdot\frac{L_\mathrm{h}}{L'_\mathrm{r}}\cdot\Psi'_{\mathrm{r}\beta}\cdot i_{\mathrm{s}\alpha} + k_\mathrm{p}\cdot p_\mathrm{p}\cdot\frac{L_\mathrm{h}}{L'_\mathrm{r}}\cdot\Psi'_{\mathrm{r}\alpha}\cdot i_{\mathrm{s}\beta} \ . \tag{1.94}$$

Nyní je možné dosadit do (1.91):

$$\frac{\mathrm{d}\omega_\mathrm{m}}{\mathrm{d}t} = \frac{-k_\mathrm{p}\cdot p_\mathrm{p}\cdot\frac{L_\mathrm{h}}{L'_\mathrm{r}}\cdot\Psi'_{\mathrm{r}\beta}\cdot i_{\mathrm{s}\alpha} + k_\mathrm{p}\cdot p_\mathrm{p}\cdot\frac{L_\mathrm{h}}{L'_\mathrm{r}}\cdot\Psi'_{\mathrm{r}\alpha}\cdot i_{\mathrm{s}\beta} - M_\mathrm{z} - B_{\omega_\mathrm{m}}}{J} \ . \tag{1.95}$$

Tím je získána poslední stavová rovnice, kterou je ještě potřeba upravit do finálního tvaru. Po rozdělení zlomku vychází tvar

$$\frac{\mathrm{d}\omega_\mathrm{m}}{\mathrm{d}t} = -\underbrace{\frac{k_\mathrm{p}\cdot p_\mathrm{p}}{J}\cdot\frac{L_\mathrm{h}}{L'_\mathrm{r}}}_{\varepsilon}\cdot\Psi'_{\mathrm{r}\beta}\cdot i_{\mathrm{s}\alpha} + \underbrace{\frac{k_\mathrm{p}\cdot p_\mathrm{p}}{J}\cdot\frac{L_\mathrm{h}}{L'_\mathrm{r}}}_{\varepsilon}\cdot\Psi'_{\mathrm{r}\alpha}\cdot i_{\mathrm{s}\beta} - \frac{1}{J}\cdot M_\mathrm{z} - \frac{1}{J}\cdot B_{\omega_\mathrm{m}} \ . \tag{1.96}$$

Dosazením konstanty (1.65) je získán finální tvar [17][24]

$$\boxed{\frac{\mathrm{d}\omega_\mathrm{m}}{\mathrm{d}t} = -\varepsilon\cdot\Psi'_{\mathrm{r}\beta}\cdot i_{\mathrm{s}\alpha} + \varepsilon\cdot\Psi'_{\mathrm{r}\alpha}\cdot i_{\mathrm{s}\beta} - \frac{1}{J}\cdot M_\mathrm{z} - \frac{1}{J}\cdot B_{\omega_\mathrm{m}} \ .} \tag{1.97}$$

Kompletní matematický model asynchronního motoru ve stojícím souřadném systému

$\alpha\beta$ zahrnuje rovnice (1.73), (1.74), (1.88), (1.89) a (1.97): [11][24]

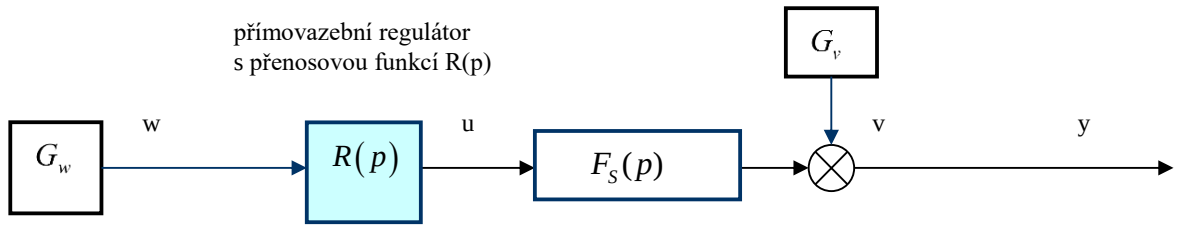
$$\begin{aligned}
\frac{\mathrm{d}i_{\mathrm{s}_\alpha}}{\mathrm{d}t} &= -\alpha \cdot i_{\mathrm{s}_\alpha} + \beta \cdot \Psi'_{\mathrm{r}_\alpha} + \gamma \cdot \omega_{\mathrm{m}} \cdot \Psi'_{\mathrm{r}_\beta} - \frac{L_{\mathrm{h}}}{L'_{\mathrm{r}}} \cdot \delta \cdot u'_{\mathrm{r}_\alpha} + \delta \cdot u_{\mathrm{s}_\alpha} \ , \\
\frac{\mathrm{d}i_{\mathrm{s}_\beta}}{\mathrm{d}t} &= -\alpha \cdot i_{\mathrm{s}_\beta} - \gamma \cdot \omega_{\mathrm{m}} \cdot \Psi'_{\mathrm{r}_\alpha} + \beta \cdot \Psi'_{\mathrm{r}_\beta} - \frac{L_{\mathrm{h}}}{L'_{\mathrm{r}}} \cdot \delta \cdot u'_{\mathrm{r}_\beta} + \delta \cdot u_{\mathrm{s}_\beta} \ , \\
\frac{\mathrm{d}\Psi'_{\mathrm{r}_\alpha}}{\mathrm{d}t} &= R'_{\mathrm{r}} \cdot \frac{L_{\mathrm{h}}}{L'_{\mathrm{r}}} \cdot i_{\mathrm{s}_\alpha} - \frac{R'_{\mathrm{r}}}{L'_{\mathrm{r}}} \cdot \Psi'_{\mathrm{r}_\alpha} - p_{\mathrm{p}} \cdot \omega_{\mathrm{m}} \cdot \Psi'_{\mathrm{r}_\beta} + u'_{\mathrm{r}_\alpha} \ , \\
\frac{\mathrm{d}\Psi'_{\mathrm{r}_\beta}}{\mathrm{d}t} &= R'_{\mathrm{r}} \cdot \frac{L_{\mathrm{h}}}{L'_{\mathrm{r}}} \cdot i_{\mathrm{s}_\beta} + p_{\mathrm{p}} \cdot \omega_{\mathrm{m}} \cdot \Psi'_{\mathrm{r}_\alpha} - \frac{R'_{\mathrm{r}}}{L'_{\mathrm{r}}} \cdot \Psi'_{\mathrm{r}_\beta} + u'_{\mathrm{r}_\beta} \ , \\
\frac{\mathrm{d}\omega_{\mathrm{m}}}{\mathrm{d}t} &= -\varepsilon \cdot \Psi'_{\mathrm{r}\beta} \cdot i_{\mathrm{s}\alpha} + \varepsilon \cdot \Psi'_{\mathrm{r}\alpha} \cdot i_{\mathrm{s}\beta} - \frac{1}{J} \cdot M_{\mathrm{z}} - \frac{1}{J} \cdot B_{\omega_{\mathrm{m}}} \ .
\end{aligned} \tag{1.98}$$

Kapitola 2

Regulace elektrických strojů

Regulátory či řídicí systémy jsou v dnešní době nepostradatelnou součástí většiny zařízení od mobilního telefonu, kde zajišťují např. správné nabití baterie, po autopilota v letadle. V případě elektrických strojů se používají např. k regulaci proudu, ale také rychlosti otáčení nebo polohy. Jejich primární funkcí je prostřednictvím vstupů regulovat chování jedné nebo více proměnných na řízeném systému (procesu). Regulovaná veličina je regulátorem udržována na zadané konstantní hodnotě nebo sleduje zadanou trajektorii. Požadované chování je regulátoru předáváno formou referenčního vstupního signálu.

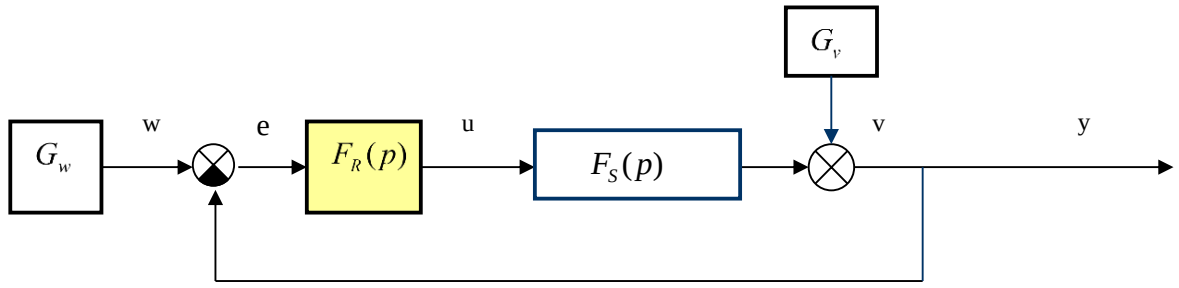
Regulátor může generovat řízení systému pouze na základě referenčního signálu bez informace o výstupu z řízeného systému, jak je naznačeno na obr. 2.1. V takovém případě se jedná o přímovazební (programové) řízení. Pokud na řízený systém nebudou působit vnější poruchy, nebo jejich působení nebude působit významnější odchylky regulované veličiny od hodnot zadávaných referenčním signálem, pak je tento způsob řízení použitelný. [12]



Obr. 2.1: Přímovazební regulační obvod [12]

V případě, že působící poruchy mají významný vliv na regulovanou veličinu, je potřeba zajistit, aby regulátor byl schopen odchylku regulované veličiny detekovat a odstranit. K tomuto účelu je využívána zpětná vazba od regulované veličiny k referenčnímu signálu. Na obr. 2.2 je naznačena realizace zpětnovazebního regulačního obvodu s regulátorem 1DoF (s jedním stupněm volnosti – 1 Degree of Freedom). [12]

V rámci této diplomové práce byla řešena regulace asynchronního stroje s klecovou kotvou. Ta bývá velmi často řešena pomocí algoritmu PI(D) regulace [20]. Další část této kapitoly bude tedy věnována stručnému popisu tohoto druhu regulátoru.



Obr. 2.2: Zpětnovazební regulační obvod s regulátorem 1DoF [12]

kde:

$F_\mathrm{S}(p)$	– Přenosová funkce řízeného systému,
$F_\mathrm{R}(p)$	– Přenosová funkce regulátoru,
G_w	– generátor referenčního signálu $w(t)$,
G_v	– generátor výstupní poruchy $v(t)$,
$e(t) = w(t) - y(t)$	– regulační odchylka,
$u(t)$	– řízení,
$y(t)$	– měřený regulovaný výstup. [12]

2.1 PID (PI, PD) regulátory

Název regulátorů je odvozen od způsobu, jakým generuje řízení $u(t)$ v reakci na regulační odchylku $e(t) = w(t) - y(t)$ v regulačním obvodu. Algoritmus řízení se skládá ze tří složek: proporcionální složka (**P**), integrační složka (**I**) a derivační složka (**D**). Matematicky jej lze popsat jako

$$u(t) = K \cdot \left(e(t) + \frac{1}{T_i} \cdot \int_0^t e(\tau)\,\mathrm{d}\tau + T_d \cdot \frac{\mathrm{d}e(t)}{\mathrm{d}t} \right) , \tag{2.1}$$

kde K je proporcionální zesílení, T_i integrační časová konstanta a T_d derivační časová konstanta. Aplikací Laplaceovy transformace na tento vztah lze získat spojitý přenos PID regulátoru: [13]

$$F_\mathrm{R}(p) = \frac{U(p)}{E(p)} = K \cdot \left(1 + \frac{1}{T_i \cdot p} + T_d \cdot p \right) = K + \frac{K_i}{p} + K_d \cdot p = \frac{K_d \cdot p^2 + K \cdot p + K_i}{p} \tag{2.2}$$

Proporcionální složka

Proporcionální složka generuje regulační zásah, který je úměrný velikosti regulační odchylky. Je-li zvyšována konstanta K, dochází ke zpřesnění regulace, k nárůstu rychlosti odezvy a k většímu potlačení nízkofrekvenčních poruch. Zároveň však také dochází k většímu přeregulování a ke snížení robustnosti ve stabilitě. [13] Zvyšování hodnoty K

také způsobuje, že systém je více citlivý na šum měření. Volba velikosti hodnoty K je tedy kompromisem mezi stavem, kdy reakce regulátoru je dostatečně rychlá, ale zároveň charakter přechodného děje odpovídá požadavkům (rozumně velký překmit, nekmitavý děj, ...) a systém není nepříznivě ovlivňován šumem měření. [3]

Integrační složka

Integrační složka je úměrná integrálu regulační odchylky a její hlavní funkcí je zajistit, aby procesní výstup souhlasil s požadovanou hodnotou v ustáleném stavu. U proporcionálního řízení obvykle v ustáleném stavu trvale zůstává regulační odchylka. Při zavedení integrační složky povede i malá kladná hodnota regulační odchylky vždy ke zvýšení řídicího signálu a malou zápornou hodnotou regulační odchylky bude velikost řídicího signálu snižována.[3] Její zavedení je tedy žádoucí pro dosažení přesnosti regulace. Má však také za následek zpomalení rychlosti odezvy systému a snížení robustnosti ve stabilitě. Při zmenšování hodnoty T_i dochází ke zrychlení integrace, což způsobuje větší přeregulování. Integrační složkou je do regulační smyčky zavedeno fázové zpoždění. [13]

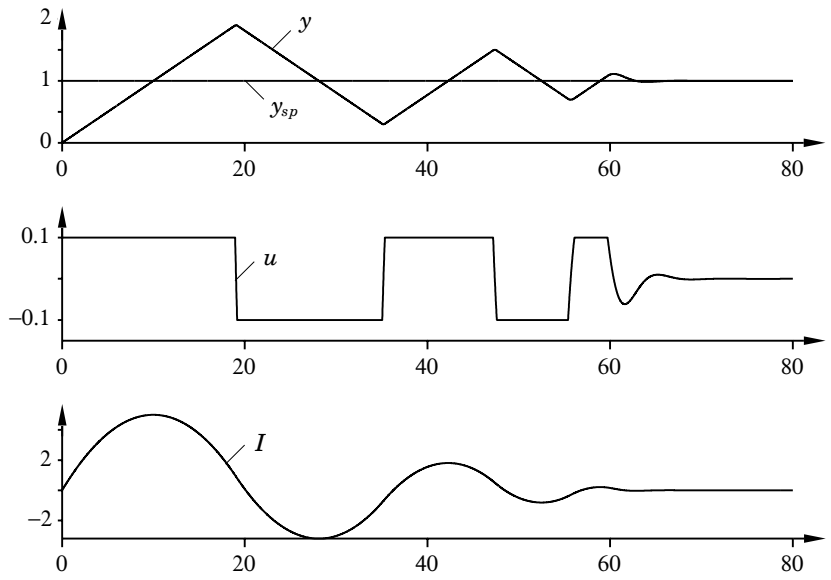
Derivační složka

Derivační složka je úměrná derivaci regulační odchylky a jejím účelem je zlepšit stabilitu uzavřené smyčky. Činnost regulátoru s proporcionálním a derivačním působením lze interpretovat tak, jako by bylo řízení provedeno proporcionálně k predikovanému výstupu procesu, kde se předpověď provádí extrapolací chyby tečnou ke křivce chyby. [3] Derivační složka tedy do regulační smyčky zavádí fázový předstih a zrychluje rychlost odezvy. Při zvyšování hodnoty T_d se zmenšuje velikost přeregulování a může dojít ke zvýšení robustnosti ve stabilitě. [13]

2.1.1 Unášení integrační složky, anti-windup

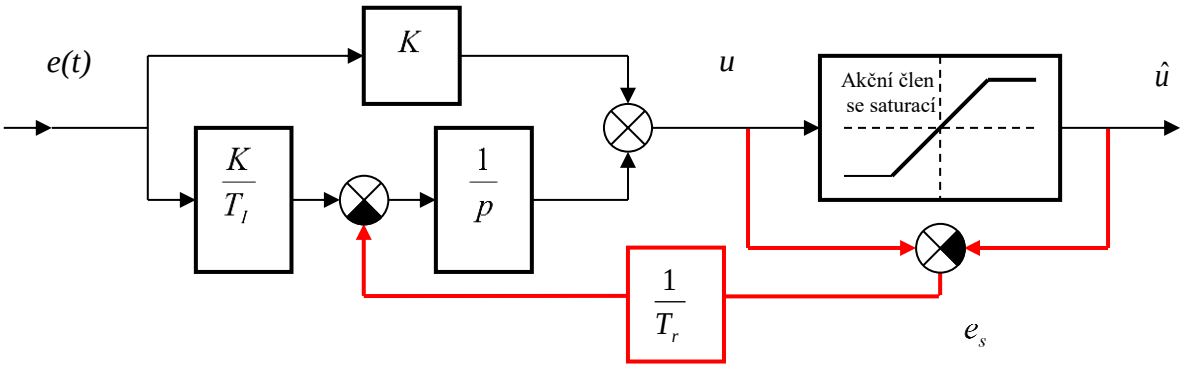
V reálném světě mají všechna zařízení své limity: motor se může točit pouze omezenou rychlostí, ventil nelze otevřít „více než zcela“, ... Pokud jsou tato zařízení používána jako akční členy a dosáhnou této saturace, dojde k přerušení zpětnovazební smyčky, protože akční člen zůstane na svém limitu nezávisle na výstupu regulátoru. Pokud regulátor obsahuje integrační složku, regulační odchylka bude neustále integrována. [3] Po odeznění saturace pak může trvat značně dlouho, než se vlivem „naintegrované“ hodnoty obnoví správná funkce regulátoru. Tento efekt je často nazýván jako „unášení integrační složky“ (windup effect). [13] Důsledky tohoto jevu jsou názorně ukázány na obr. 2.3.

Počáteční změna požadavku je tak velká, že dochází k saturaci akčního členu na horní hranici. Integrační složka se zvyšuje až do okamžiku, kdy hodnota regulované veličiny překročí požadovanou hodnotu. Naintegrovaná hodnota je však příliš velká na to, aby akční člen přestal být nasycen, a dochází k výraznému překmitu. V okamžiku, kdy hodnota



Obr. 2.3: Unášení integrační složky při saturaci akčního členu [3]

integrační složky dostatečně poklesne a umožní odsycení akčního členu, dochází k saturaci naopak na dolní hranici a celý proces se opakuje, dokud se velikost regulační odchylky nedostane pod úroveň, při které již nedochází k saturaci akčního členu. Windup efekt má tedy neblahý vliv na kvalitu regulace. [3]



Obr. 2.4: PI regulátor s ochranou anti-windup založenou na principu back-calculation [13]

Jednou z možností realizace anti-windup ochrany je tzv. back-calculation, jehož princip je ukázán na obr. 2.4. PI regulátor je doplněn o další zpětnovazební smyčku odvozenou od odchylky $e_s = u - \hat{u}$, dané rozdílem měřeného výstupu regulátoru u a měřeného výstupu akčního členu $\hat{u}$. Odchylka je přivedena zpět do integrátoru přes zesílení $1/T_r$. Pokud akční člen není saturován, platí $u = \hat{u}$, odchylka e_s je nulová a řízení probíhá v lineární oblasti. Pokud dojde k saturaci akčního členu, odchylkou e_s násobenou konstantou $1/T_r$ začne být velikost integrace zmenšována tak, aby se výstup regulátoru dostal na mez saturace. [13] V praxi může nastat situace, kdy není možné měřit hodnotu na výstupu akčního členu. V takovém případě je možné použít uspořádání, kdy by blok akční člen se saturací na obr. 2.4 představoval model akčního členu, na kterém je už možné měření provést. [3]

2.2 Linearizace nelineárních dynamických systémů

Většina fyzikálních systémů má nelineární chování, nicméně při malých změnách systémových proměnných se mnoho z nich chová téměř lineárně. Je tedy možné nahradit takový nelineární dynamický systém (NDS) jeho linearizovaným modelem, získaným lineární aproximací stavové a výstupní rovnice NDS v okolí rovnovážných nebo ustálených stavů – „pracovních bodů“. Jedná se tedy o lokální aproximativní linearizaci NDS. Vlastnosti NDS odvozené z jeho linearizovaného modelu proto platí pouze v okolí pracovního bodu a při větších odchylkách systémových proměnných od tohoto bodu se mohou od skutečných vlastností výrazně lišit.

Je dán nelineární dynamický systém [12]

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\left[\boldsymbol{x}(t), \boldsymbol{u}(t)\right] , \tag{2.3}$$

$$\boldsymbol{y}(t) = \boldsymbol{h}\left[\boldsymbol{x}(t), \boldsymbol{u}(t)\right] , \tag{2.4}$$

kde:

$\boldsymbol{x}(t) \in \mathbb{R}^n$ – vektor stavu systému,
$\boldsymbol{u}(t) \in \mathbb{R}^m$ – vektor vstupu systému,
$\boldsymbol{y}(t) \in \mathbb{R}^p$ – vektor výstupu systému,
$\boldsymbol{f}[.]$ – n-rozměrná nelineární vektorová funkce,
$\boldsymbol{h}[.]$ – p-rozměrná nelineární vektorová funkce.

Je potřeba vytvořit linearizovaný model, který bude aproximovat chování NDS v blízkém okolí ustáleného stavu (pracovního bodu) $\boldsymbol{x}_r$, určeného definicí rovnovážných a ustálených stavů $\boldsymbol{x}_r$ či výstupů $\boldsymbol{y}_r$, která zní: [12]

Rovnovážné stavy $\boldsymbol{x}_r$ jsou dány řešením $0 = \boldsymbol{f}\left[\boldsymbol{x}_r, 0\right]$,
rovnovážný výstup $\boldsymbol{y}_r = \boldsymbol{h}\left[\boldsymbol{x}_r, 0\right]$. (2.5)

Ustálené stavy $\boldsymbol{x}_r$ jsou dány řešením $0 = \boldsymbol{f}\left[\boldsymbol{x}_r, \boldsymbol{u}_{\text{konst.}}\right]$,
ustálený výstup $\boldsymbol{y}_r = \boldsymbol{h}\left[\boldsymbol{x}_r, \boldsymbol{u}_{\text{konst.}}\right]$. (2.6)

Blízké okolí pracovního bodu bude respektováno zavedením tzv. odchylkových proměnných

$$\begin{aligned} \Delta\boldsymbol{x}(t) &= \boldsymbol{x}(t) - \boldsymbol{x}_r \quad &\Rightarrow \quad \boldsymbol{x}(t) &= \boldsymbol{x}_r + \Delta\boldsymbol{x}(t) , \\ \Delta\boldsymbol{u}(t) &= \boldsymbol{u}(t) - \boldsymbol{u}_{\text{konst.}} \quad &\Rightarrow \quad \boldsymbol{u}(t) &= \boldsymbol{u}_{\text{konst.}} + \Delta\boldsymbol{u}(t) , \\ \Delta\boldsymbol{y}(t) &= \boldsymbol{y}(t) - \boldsymbol{y}_r \quad &\Rightarrow \quad \boldsymbol{y}(t) &= \boldsymbol{y}_r + \Delta\boldsymbol{y}(t) . \end{aligned} \tag{2.7}$$

Linearizovaný model lze potom získat rozvojem nelineárních vektorových funkcí $\boldsymbol{f}[.]$ resp. $\boldsymbol{h}[.]$ ve stavové resp. výstupní rovnici NDS v Taylorovu řadu v okolí pracovního bodu při zanedbání vyšších členů rozvoje. Pro stavovou rovnici potom vychází

$$\begin{aligned} \dot{\boldsymbol{x}}_r + \Delta\dot{\boldsymbol{x}}(t) &= \boldsymbol{f}[\boldsymbol{x}_r + \Delta\boldsymbol{x}(t), \boldsymbol{u}_{\text{konst.}} + \Delta\boldsymbol{u}(t)] = \\ &= \boldsymbol{f}(\boldsymbol{x}_r, \boldsymbol{u}_{\text{konst.}}) + \left.\frac{\partial \boldsymbol{f}[.]}{\partial \boldsymbol{x}}\right|_{x_r, u_{\text{konst.}}} \cdot \left[\boldsymbol{x}(t) - \boldsymbol{x}_r\right] + \left.\frac{\partial \boldsymbol{f}[.]}{\partial \boldsymbol{u}}\right|_{x_r, u_{\text{konst.}}} \cdot \left[\boldsymbol{u}(t) - \boldsymbol{u}_{\text{konst.}}\right] \end{aligned}$$

a stejně tak pro výstupní rovnici vychází

$$\boldsymbol{y}_r + \Delta\boldsymbol{y}(t) = \boldsymbol{h}[\boldsymbol{x}_r + \Delta\boldsymbol{x}(t), \boldsymbol{u}_{\text{konst.}} + \Delta\boldsymbol{u}(t)] = \\ = \boldsymbol{h}(\boldsymbol{x}_r, \boldsymbol{u}_{\text{konst.}}) + \left.\frac{\partial \boldsymbol{h}[.]}{\partial \boldsymbol{x}}\right|_{x_r, u_{\text{konst.}}} \cdot \left[\boldsymbol{x}(t) - \boldsymbol{x}_r\right] + \left.\frac{\partial \boldsymbol{h}[.]}{\partial \boldsymbol{u}}\right|_{x_r, u_{\text{konst.}}} \cdot \left[\boldsymbol{u}(t) - \boldsymbol{u}_{\text{konst.}}\right].$$

Jelikož pro ustálený stav resp. výstup systému platí $\dot{\boldsymbol{x}}_r = 0 = \boldsymbol{f}\left(\boldsymbol{x}_r, \boldsymbol{u}_{\text{konst.}}\right)$, resp. $\boldsymbol{y}_r = \boldsymbol{h}\left(\boldsymbol{x}_r, \boldsymbol{u}_{\text{konst.}}\right)$, stavová a výstupní rovnice linearizovaného modelu v odchylkových proměnných potom bude

$$\Delta\dot{\boldsymbol{x}}(t) = \left.\frac{\partial \boldsymbol{f}[.]}{\partial \boldsymbol{x}}\right|_{x_r, u_{\text{konst.}}} \cdot \Delta\boldsymbol{x}(t) + \left.\frac{\partial \boldsymbol{f}[.]}{\partial \boldsymbol{u}}\right|_{x_r, u_{\text{konst.}}} \cdot \Delta\boldsymbol{u}(t)\ , \tag{2.8}$$

$$\Delta\boldsymbol{y}(t) = \left.\frac{\partial \boldsymbol{h}[.]}{\partial \boldsymbol{x}}\right|_{x_r, u_{\text{konst.}}} \cdot \Delta\boldsymbol{x}(t) + \left.\frac{\partial \boldsymbol{h}[.]}{\partial \boldsymbol{u}}\right|_{x_r, u_{\text{konst.}}} \cdot \Delta\boldsymbol{u}(t)\ . \tag{2.9}$$

V blízkém okolí pracovního bodu jsou tyto rovnice formálně shodné se stavovou reprezentací lineárního dynamického systému (LDS)

$$\dot{\boldsymbol{x}}(t) = \mathbf{A} \cdot \boldsymbol{x}(t) + \mathbf{B} \cdot \boldsymbol{u}(t)\ , \tag{2.10}$$

$$\boldsymbol{y}(t) = \mathbf{C} \cdot \boldsymbol{x}(t) + \mathbf{D} \cdot \boldsymbol{u}(t)\ . \tag{2.11}$$

kde:

A – matice systému o rozměrech ($n \times n$),
B – matice vstupu o rozměrech ($n \times m$),
C – matice výstupu o rozměrech ($p \times n$),
D – matice přímého působení vstupu na výstup o rozměrech ($p \times m$).

Matice **A**, **B**, **C** a **D** lze určit pomocí tzv. Jacobiových matic (vztahy (2.12) až (2.15)) po dosazení veličin definujících pracovní bod ($\boldsymbol{x}(t) = \boldsymbol{x}_r$ a $\boldsymbol{u}(t) = \boldsymbol{u}_{\text{konst.}}$) [12]

$$\mathbf{A} = \left.\frac{\partial \boldsymbol{f}[.]}{\partial \boldsymbol{x}}\right|_{x_r, u_{\text{konst.}}} = \begin{bmatrix} \dfrac{\partial f_1(.)}{\partial x_1} & \cdots & \dfrac{\partial f_1(.)}{\partial x_n} \\ \vdots & & \vdots \\ \dfrac{\partial f_n(.)}{\partial x_1} & \cdots & \dfrac{\partial f_n(.)}{\partial x_n} \end{bmatrix}_{x_r, u_{\text{konst.}}}, \tag{2.12}$$

$$\mathbf{B} = \left.\frac{\partial \boldsymbol{f}[.]}{\partial \boldsymbol{u}}\right|_{x_r, u_{\text{konst.}}} = \begin{bmatrix} \dfrac{\partial f_1(.)}{\partial u_1} & \cdots & \dfrac{\partial f_1(.)}{\partial u_m} \\ \vdots & & \vdots \\ \dfrac{\partial f_n(.)}{\partial u_1} & \cdots & \dfrac{\partial f_n(.)}{\partial u_m} \end{bmatrix}_{x_r, u_{\text{konst.}}}, \tag{2.13}$$

$$\mathbf{C} = \left.\frac{\partial \boldsymbol{h}[.]}{\partial \boldsymbol{x}}\right|_{x_r, u_{\text{konst.}}} = \begin{bmatrix} \dfrac{\partial h_1(.)}{\partial x_1} & \cdots & \dfrac{\partial h_1(.)}{\partial x_n} \\ \vdots & & \vdots \\ \dfrac{\partial h_p(.)}{\partial x_1} & \cdots & \dfrac{\partial h_p(.)}{\partial x_n} \end{bmatrix}_{x_r, u_{\text{konst.}}}, \tag{2.14}$$

$$\mathbf{D} = \left.\frac{\partial \boldsymbol{h}[.]}{\partial \boldsymbol{u}}\right|_{x_r, u_{\text{konst.}}} = \left[\begin{array}{ccc} \dfrac{\partial h_1(.)}{\partial u_1} & \cdots & \dfrac{\partial h_1(.)}{\partial u_m} \\ \vdots & & \vdots \\ \dfrac{\partial h_p(.)}{\partial u_1} & \cdots & \dfrac{\partial h_p(.)}{\partial u_m} \end{array}\right]_{x_r, u_{\text{konst.}}} . \tag{2.15}$$

Kapitola 3

Skalární řízení asynchronních strojů

Nejjednodušší varianta frekvenčního řízení asynchronního motoru je tzv. skalární řízení (také nazývané jako napěťově kmitočtové řízení [24]). Jeho princip je založen na současné změně amplitudy statorového napětí a jeho frekvence. To lze realizovat např. pomocí napěťového střídače nebo frekvenčního měniče. [17]

3.1 Odvození pravidel řízení ze statorové rovnice

Pro odvození pravidel skalárního řízení lze vyjít z rovnice pro statorové napětí v souřadném systému α, β (1.16). V ustáleném stavu platí, že vektory napětí, proudů a toků se pohybují konstantní rychlostí $\omega_\mathrm{s} = 2 \cdot \pi \cdot f_\mathrm{s}$. Prostorové vektory je tedy možné nahradit komplexory:

$$\overline{u}_\mathrm{s} \rightarrow \overline{U}_\mathrm{s} \cdot \mathrm{e}^{\mathrm{j} \cdot \omega_\mathrm{s} \cdot t} \ , \tag{3.1}$$

$$\overline{\Psi}_\mathrm{s} \rightarrow \overline{\Psi}_\mathrm{s} \cdot \mathrm{e}^{\mathrm{j} \cdot \omega_\mathrm{s} \cdot t} \ . \tag{3.2}$$

Pro zjednodušení úvah je dále možné uvažovat, že $R_\mathrm{s} \rightarrow 0$, čímž je možné zanedbat úbytek na statorovém odporu R_s. Potom platí

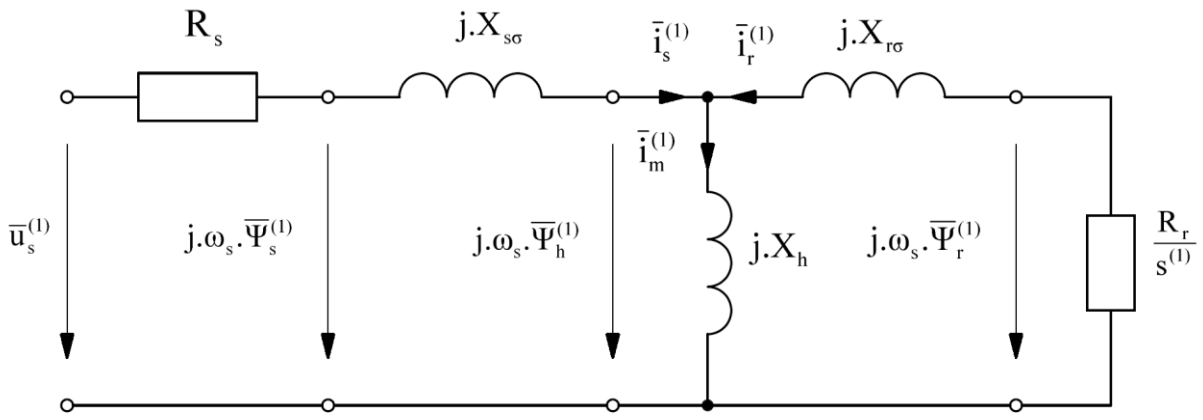
$$\overline{U}_\mathrm{s} \cdot \mathrm{e}^{\mathrm{j} \cdot \omega_\mathrm{s} \cdot t} = \frac{\mathrm{d}}{\mathrm{d}t} \left(\overline{\Psi}_\mathrm{s} \cdot \mathrm{e}^{\mathrm{j} \cdot \omega_\mathrm{s} \cdot t} \right) = \mathrm{j} \cdot \omega_\mathrm{s} \cdot \overline{\Psi}_\mathrm{s} \cdot \mathrm{e}^{\mathrm{j} \cdot \omega_\mathrm{s} \cdot t} \ . \tag{3.3}$$

Vyjádřením statorového toku z předchozího vztahu vychází

$$\overline{\Psi}_\mathrm{s} = -\mathrm{j} \cdot \frac{\overline{U}_\mathrm{s}}{\omega_\mathrm{s}} \Rightarrow \left| \overline{\Psi}_\mathrm{s} \right| = \frac{\left| \overline{U}_\mathrm{s} \right|}{\omega_\mathrm{s}} \ . \tag{3.4}$$

Z tohoto vztahu je patrné, že pokud má být udržován konstantní tok ve stroji, je nutné udržovat konstantní poměr velikostí statorového napětí a statorové frekvence: [18]

$$\left| \overline{\Psi}_\mathrm{s} \right| = \frac{\left| \overline{U}_\mathrm{s} \right|}{\omega_\mathrm{s}} = \mathrm{konst.} \tag{3.5}$$



Obr. 3.1: Náhradní schéma pro první harmonické veličin ASM v ustáleném stavu [17]

U tohoto typu řízení dochází k regulaci velikosti vektorů veličin nikoliv jejich vzájemné polohy [18]. Algoritmy napěťově kmitočtového řízení vychází z následujících požadavků:

- $f_\mathrm{s} < f_\mathrm{sN}$ ... v ustáleném stavu $\left|\overline{\Psi}_\mathrm{s}\right| = \text{konst.}$
- $f_\mathrm{s} > f_\mathrm{sN}$ ... $\left|\overline{U}_\mathrm{s}^{(1)}\right| = U_\mathrm{sN} \cdot \sqrt{2}$

Dle náhradního schématu asynchronního stroje pro první harmonické na obr. 3.1 platí

$$\overline{U}_\mathrm{s}^{(1)} = R_\mathrm{s} \cdot \overline{I}_\mathrm{s}^{(1)} + \mathrm{j} \cdot \omega_\mathrm{s} \cdot \overline{\Psi}_\mathrm{s}^{(1)} = \Delta\overline{U}_\mathrm{R}^{(1)} + \overline{U}_\mathrm{ind}^{(1)} \ . \tag{3.6}$$

Za předpokladu $\overline{\Psi}_\mathrm{s}^{(1)} = \text{konst.}$ platí

$$\left|\overline{U}_\mathrm{ind}^{(1)}\right| = \left|\overline{\Psi}_\mathrm{s}^{(1)}\right| \cdot \omega_\mathrm{s} = K_U \cdot f_\mathrm{s} \ , \tag{3.7}$$

z čehož je možné přibližně odvodit konstantu K_U (viz obr. 3.3): [24]

$$\boxed{K_U \cong \frac{U_\mathrm{sN} \cdot \sqrt{2}}{f_\mathrm{sN}} \ .} \tag{3.8}$$

Výpočet požadavku na amplitudu statorového napětí se většinou pro jednoduchost provádí pomocí prostého algebraického součtu (horní součtový člen na obr. 3.3)

$$\left|\overline{U}_\mathrm{s}^{(1)}\right| \cong \left|\overline{U}_\mathrm{ind}^{(1)}\right| + \Delta U \tag{3.9}$$

kde ΔU je skalár a je průmětem vektoru $\overline{U}_\mathrm{R}^{(1)}$ do směru $\overline{U}_\mathrm{ind}^{(1)}$. Veličinu ΔU lze vyhodnocovat následujícími způsoby:

$\Delta U = f(f_\mathrm{s})$ **...** Většinou se vychází z předpokladu, že při $f_\mathrm{s} \to 0$ se vždy motor rozbíhá, takže je nutno zadávat ΔU. Při $f_\mathrm{s} \to f_\mathrm{sN}$ často pracuje motor naprázdno, vektory $\overline{U}_\mathrm{ind}^{(1)}$ a $\Delta\overline{U}_\mathrm{R}^{(1)}$ jsou navzájem kolmé, takže $\Delta U \cong 0$.

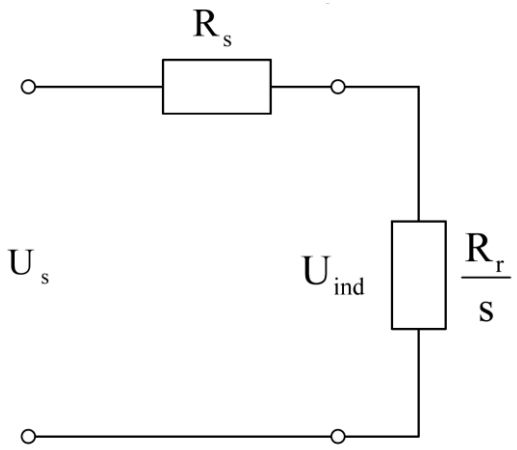
$\Delta U = f(f_\mathrm{r})$ **...** Závislost velikosti ΔU na rotorové frekvenci lze odvodit pomocí zjednodušeného náhradního schématu, ve kterém neuvažujeme magnetizační proud (viz obr. 3.2), neboť tato složka proudu $\overline{I}_\mathrm{s}$ nemá na určovaní velikosti ΔU vliv.

$$\Delta U = R_\mathrm{s} \cdot \frac{U_\mathrm{ind}}{\frac{R_\mathrm{r}}{s}} = R_\mathrm{s} \cdot \frac{U_\mathrm{ind}}{R_\mathrm{r} \cdot f_\mathrm{s}} \cdot f_\mathrm{r} \cong \underbrace{\frac{U_\mathrm{sN} \cdot \sqrt{2} \cdot R_\mathrm{s}}{R_\mathrm{r} \cdot f_\mathrm{sN}}}_{K_{f_\mathrm{r}}} \cdot f_\mathrm{r} \ , \tag{3.10}$$

$$\Delta U = K_{f_\mathrm{r}} \cdot f_\mathrm{r} \ . \tag{3.11}$$

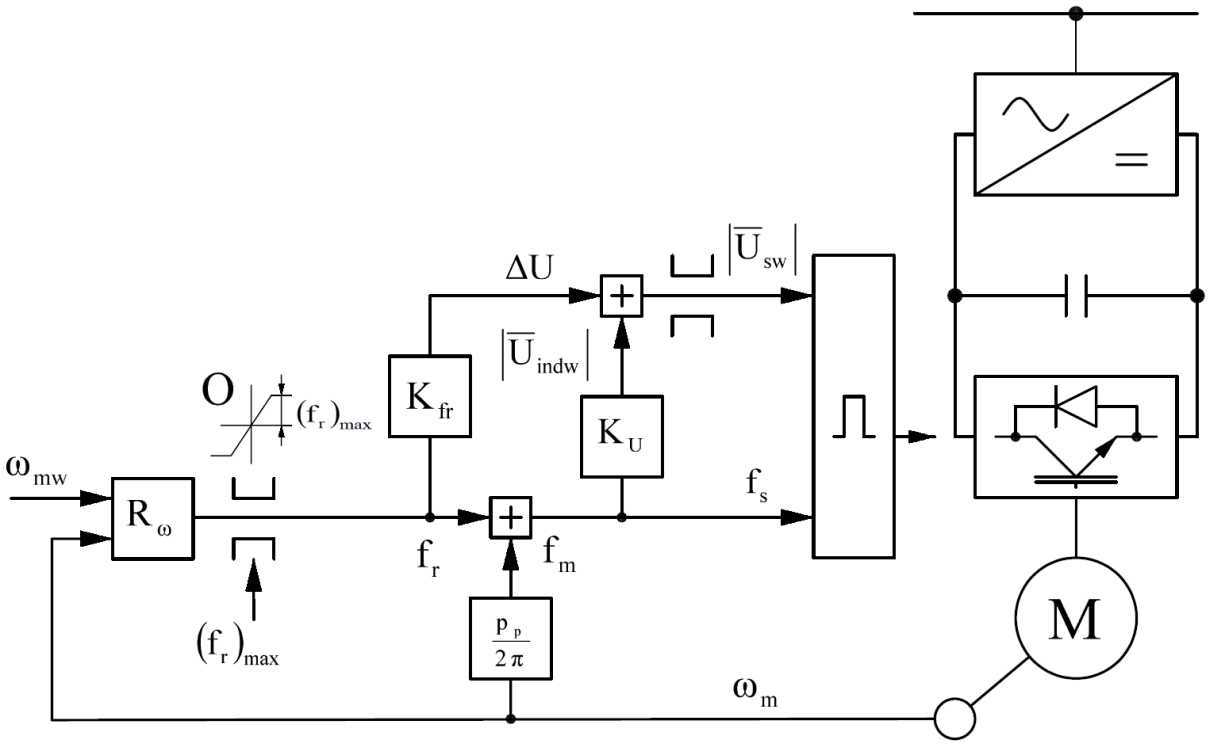
Jak je výše naznačeno, zbylá konstanta K_{f_r} viz obr. 3.3 bude [24]

$$\boxed{K_{f_\mathrm{r}} = \frac{U_\mathrm{sN} \cdot \sqrt{2} \cdot R_\mathrm{s}}{R_\mathrm{r} \cdot f_\mathrm{sN}} \ .} \tag{3.12}$$



Obr. 3.2: Zjednodušené náhradní schéma ASM [18]

Na obr. 3.3 je naznačeno schéma pohonu s regulací otáček. Výstupem z regulátoru je požadavek na velikost rotorové frekvence, který bývá omezen na hodnotu $f_{\mathrm{r_{MAX}}}$. Při řízení asynchronního stroje je nutné, aby stroj pracoval pouze v lineární části momentové charakteristiky daleko od tzv. momentu zvratu, kterému odpovídá hodnota tzv. kritické rotorové frekvence f_rk. Omezením rotorové frekvence na hodnotu $f_{\mathrm{r_{MAX}}} < f_\mathrm{rk}$ je toto zajištěno. Přičtením změřené elektrické frekvence otáčení rotoru, přepočítané z mechanické úhlové rychlosti, k požadované rotorové frekvenci je získán požadavek na frekvenci statorového napětí. Pomocí konstanty K_U je realizován princip skalárního řízení, tedy udržování konstantního poměru velikostí statorového napětí a statorové frekvence dle vztahu (3.5). První složka požadavku na amplitudu statorového napětí $|\overline{U}_\mathrm{indw}|$ tedy vznikne vynásobením požadované statorové frekvence konstantou K_U. Při nízkých otáčkách motoru se významněji projevují úbytky na statorovém odporu. Tyto úbytky je potřeba kompenzovat, a proto je potřeba hodnotu $|\overline{U}_\mathrm{indw}|$ zvětšit o hodnotu ΔU. Ta je počítána vynásobením požadavku na velikost rotorové frekvence konstantou K_{f_r}. [24]



Obr. 3.3: Strukturní schéma pohonu s regulací otáček [17]

Kapitola 4

Dostupné balíky v jazyce Python pro simulaci a modelování elektrických strojů

V rámci této práce byla provedena rešerše na existující dostupné prostředky pro simulaci dynamických systémů pomocí jazyka Python. V dnešní době je programování v jazyce Python stále populárnější nejen v oblasti datového zpracování a strojového učení, ale také v oblasti matematického modelování. V Pythonu existuje mnoho balíků a knihoven, které usnadňují vytváření matematických modelů a simulaci elektrických strojů.

Smyslem této rešerše bylo nalézt nástroj v jazyce Python, který slouží k modelování a simulaci elektrických strojů, a tento nástroj následně implementovat do stávajícího balíku DynSyPy vyvíjenému za účelem matematického modelování dynamických systémů, případně nalézt balík, jehož některé funkcionality by bylo možné v DynSyPy použít, nebo se jimi při vývoji inspirovat.

V této kapitole budou podrobněji popsány balíčky věnující se přímo elektrickým strojům. Je však vhodné zde zmínit i další nástroje dostupné v jazyce Python. Jedním ze základních balíčků používaných pro práci s dynamickými systémy je SciPy. Ten je vybaven např. optimalizovanými metodami pro numerickou integraci nebo metodami pro analýzu lineárních dynamických systémů. [23] Balík SciPy je v DynSyPy použit pro analýzu lineárních časově invariantních systémů ve frekvenční oblasti. Balíček Pynamical je nástroj umožňující modelování, simulaci, vizualizaci a animaci diskrétních nelineárních dynamických systémů a chaosu [4]. Dalším balíčkem je např. PyDSTool, který je převážně napsán v jazyce Python, pro výpočetně náročnější úlohy je možné vyvolat kód napsaný v jazycích C a Fortran (děje se automaticky bez zásahu uživatele). Tento balík je zaměřen na modely systémů popsaných obyčejnými diferenciálními rovnicemi a diferenciálně-algebraickými rovnicemi. [5]

Konkrétně k simulaci elektrických strojů jsou určeny následující balíky.

4.1 _TINY_ACMSIMC

Volně dostupný projekt na GitHub s názvem _TINY_ACMSIMC je balík v jazyce Python určený k simulaci asynchronního stroje. [1]

Balík obsahuje třídu ACIM, jež shromažďuje atributy a metody, které slouží pro simulaci asynchronního motoru. Kromě metod sloužících přímo k inicializaci parametrů konkrétního stroje a k jeho simulaci jsou zde zahrnuty také metody, díky nimž je možné uvažovat napájení motoru z měniče. V rámci metody `inverter_model()` je možné simulovat vliv měniče. Metodu je možné provozovat buď bez uvažování nelinearit, případně lze nelinearity měniče uvažovat také. [1]

Pro měření a pozorování veličin motoru v průběhu simulace je zde implementována třída Observer. Ta je vybavena jednak metodou `measurement()`, pomocí které je zajištěno měření „měřitelných“ veličin motoru, a jednak metodou `observation()`, kde jsou „neměřitelné“ veličiny dopočítávány. [1]

Dále jsou v balíku obsaženy třídy pro regulaci asynchronního stroje. Kromě PI regulátoru je zde implementována třída s_curve, která skokové změny požadavků přemění na pozvolnější náběh po křivce ve tvaru písmene S. Tím jsou v praxi odstraněny skoky v momentu motoru, což má např. příznivý vliv na jeho životnost. [1]

Grafický výstup ze simulace realizován pomocí knihovny Plotly. Pomocí této knihovny je možné vytvářet grafické výstupy v publikační kvalitě a vytváření, prohlížení a distribuci grafiky lze provádět zcela offline. Knihovna Plotly je bezplatný open-source software, kde lze nahlížet do zdrojového kódu, ale také do kódu přispívat pomocí úložiště na GitHub.[8]

4.2 ACMSimPy

Obr. 4.1: Okno real-time simulace pomocí balíku ACMSimPy

Balíček v jazyce Python s názvem ACMSimPy je dostupný na platformě GitHub. Jedná se o nástroj sloužící pro simulace řízení elektrických strojů a jejich vizualizaci. [2]

V rámci balíku je implementován jednotný model střídavého motoru založený na konceptu tzv. aktivního toku (angl. active flux concept). Tento princip je popsán např. v článku [7]. Jako solver diferenciálních rovnic je zde použita numerická integrační metoda Runge–Kutta 4. řádu (RK4, ode4). Další významnou funkcionalitou tohoto balíčku je řízení měniče za použití space-vector PWM, implementovanou podle TI ControlSUITE, tedy podle způsobu, kterým firma Texas Instruments implementuje princip space-vector PWM. [2]

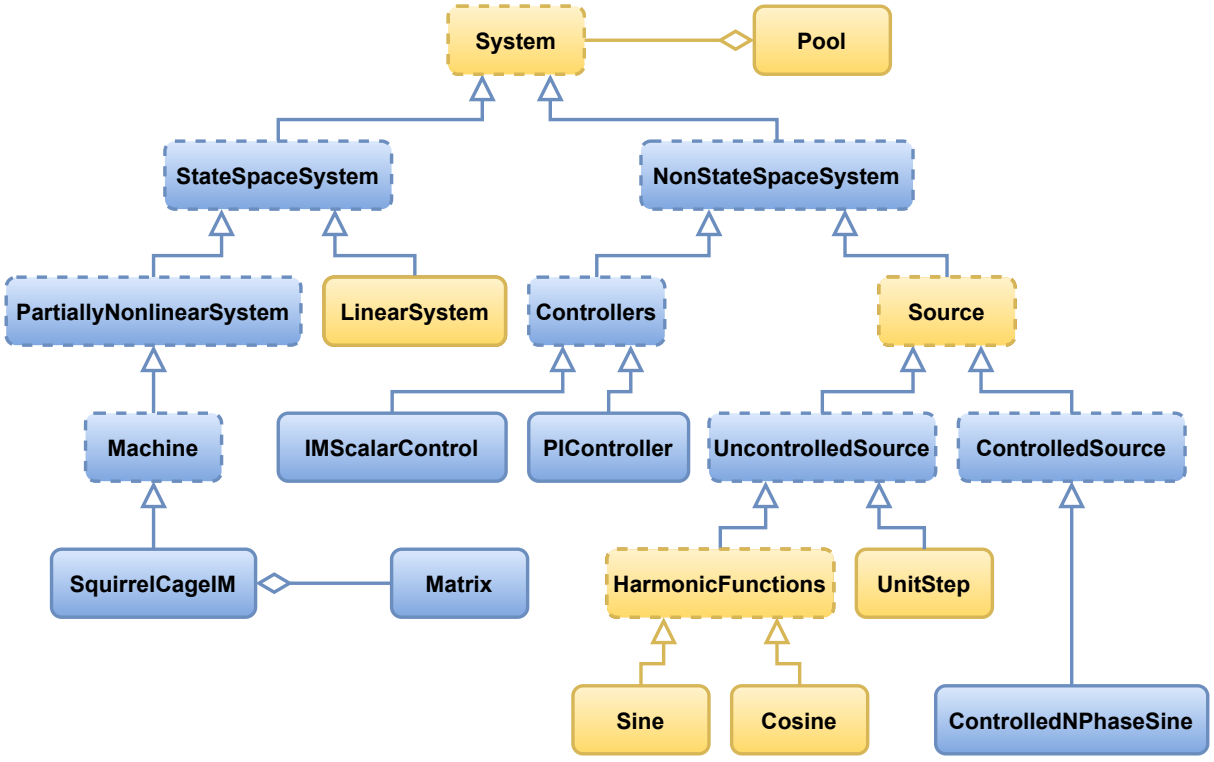
Simulace řízení stroje je provedena v reálném čase, za běhu programu je tedy možné měnit jakékoliv parametry a ladit tak pohon pohodlně v počítači. Logika uživatelského vstupu je oddělena od kódu pomocí moderního uživatelského rozhraní. [2]

Pro vytvoření uživatelského rozhraní používá balík ACMSimPy nástroj Dear PyGui, což je snadno použitelná, dynamická, GPU akcelerovaná, multiplatformní sada nástrojů grafického uživatelského rozhraní (GUI). Zahrnuje tradiční prvky GUI, jako jsou tlačítka, přepínače, nabídky a různé metody k vytvoření funkčního rozvržení. Zároveň je vybavena i pro tvorbu složitých a náročných grafických rozhraní.[6]

Kapitola 5

Realizace balíčku v jazyce Python

V této kapitole bude podrobně rozebráno, jakým způsobem jsou v jazyce Python implementovány funkce pro tvorbu dynamických modelů elektrických strojů a jejich simulaci. Na obr. 5.1 je ukázána struktura tříd balíčku DynSyPy (bez atributů a metod), jejíž současná podoba je výsledkem této diplomové práce. Třídy ohraničené přerušovanou čarou jsou abstraktní (nelze tedy vytvářet jejich instance) a slouží k rozčlenění tříd do celků, které spolu nějakým způsobem souvisí. Žlutou barvou jsou zvýrazněny třídy, které byly vytvořeny v rámci bakalářské práce [9], jež se zabývala modelováním lineárních dynamických systémů (konkrétně LTI systémů). V průběhu realizace této práce byly odhaleny některé nedostatky těchto tříd a bylo potřeba je upravit. Tomu je věnována první část této kapitoly. Modře vybarvené třídy byly vytvořeny až při zpracování diplomové práce a



Obr. 5.1: Zjednodušený diagram tříd balíčku DynSyPy

ve zbývající části této kapitoly budou podrobně popsány.

5.1 Úpravy třídy System

Abstraktní třída System je základem celého balíku. Všechny třídy kromě tříd Pool a Matrix jsou potomky třídy System. Jsou zde definovány základní atributy a metody společné pro všechny systémy.

Prvním rozdílem od původní realizace je, že metoda `step()` je nyní ve třídě System pouze definována jako prázdná metoda a její chování si dodefinují až dceřinné třídy. Důvodem k tomuto kroku bylo oddělení tříd představujících systémy popisované pomocí vnitřního popisu (soustavou diferenciálních rovnic), které je nutné řešit pomocí metod numerické integrace, od ostatních systémů, které jsou řešeny jiným způsobem. Podrobnosti budou vysvětleny v části 5.2.

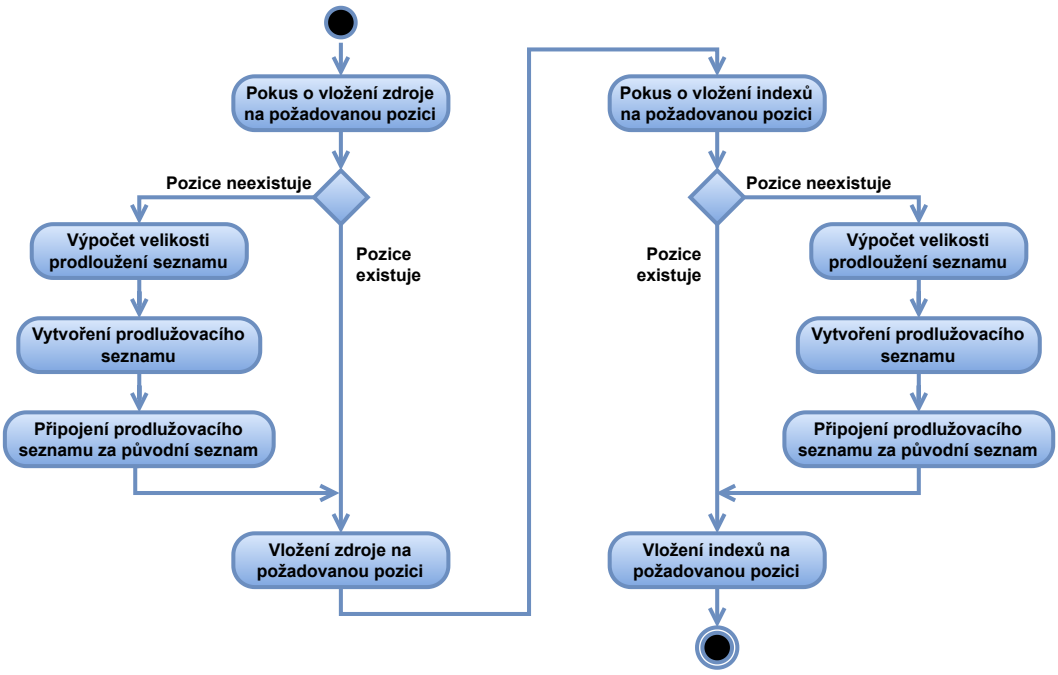
Další rozdíly třídy System jsou výraznější a jejich vysvětlení je komplikovanější, budou jim proto věnovány vlastní sekce 5.1.1 až 5.1.3. Rozdíly v ostatních třídách původního balíčku nejsou tak významné, nebudou jim proto vyhrazeny vlastní kapitoly. Jedná se převážně o přesuny definic metod do rodičovských tříd. Tyto změny budou vysvětleny při popisu nových tříd balíčku DynSyPy.

5.1.1 Metoda `connect()`

Metoda `connect()` slouží k „připojování“ zdrojů (vstupů) k systému. Původní metoda spolehlivě fungovala pouze v případě, kdy byly simulovány SISO (Single Input – Single Output) popř. MISO (Multiple Input – Single Output) systémy, nebo systémy s více výstupy, ale pouze s jedním využívaným. Poslední zmíněný případ byl řešen pomocí metody `select_output()`, pomocí níž byl před simulací vybrán jeden z výstupů. Pokud byla tato metoda volána víckrát po sobě, při simulaci byl použit výstup vybraný jako poslední.

Nebylo zde např. vyřešeno, jakým způsobem připojit ke stávajícímu systému s jedním vstupem jeden z n výstupů jiného systému. Stávající systém totiž neměl jak získat informaci o tom, který z výstupů zdroje má být použit. Nebylo také možné připojit na vstup systému vektorový výstup zdroje (např. připojení třífázového zdroje napětí).

V aktuální verzi metody `connect()` jsou tyto problémy vyřešeny. Metoda má nyní dva povinné parametry, `source` (funkce) a `position` (celé číslo), a nepovinný parametr `source_output_indexes` (typu `list[int]`, `tuple[int]` nebo `None`). Parametrem `source` je metodě předána výstupní funkce zdroje a je uložena na pozici `position` (druhý povinný parametr) v seznamu `self.sources`. Tento proces popisuje levá polovina diagramu na obr. 5.2. Výstupní funkce zdroje ovšem může vracet vektor. Je-li z tohoto vektoru potřeba např. pouze jeden výstup, je metodě předán poslední nepovinný parametr obsahující indexy požadovaných výstupů tohoto konkrétního zdroje. Pokud tento parametr není vyplněn, jeho výchozí hodnota je `None` a znamená to, že u tohoto zdroje je využíván



Obr. 5.2: Diagram aktivit metody `connect()`

celý výstup. Indexy jsou potom uloženy na stejnou pozici, jakou má zdroj, do seznamu `self.source_output_indexes`. Tento proces znázorňuje pravá polovina diagramu na obr. 5.2.

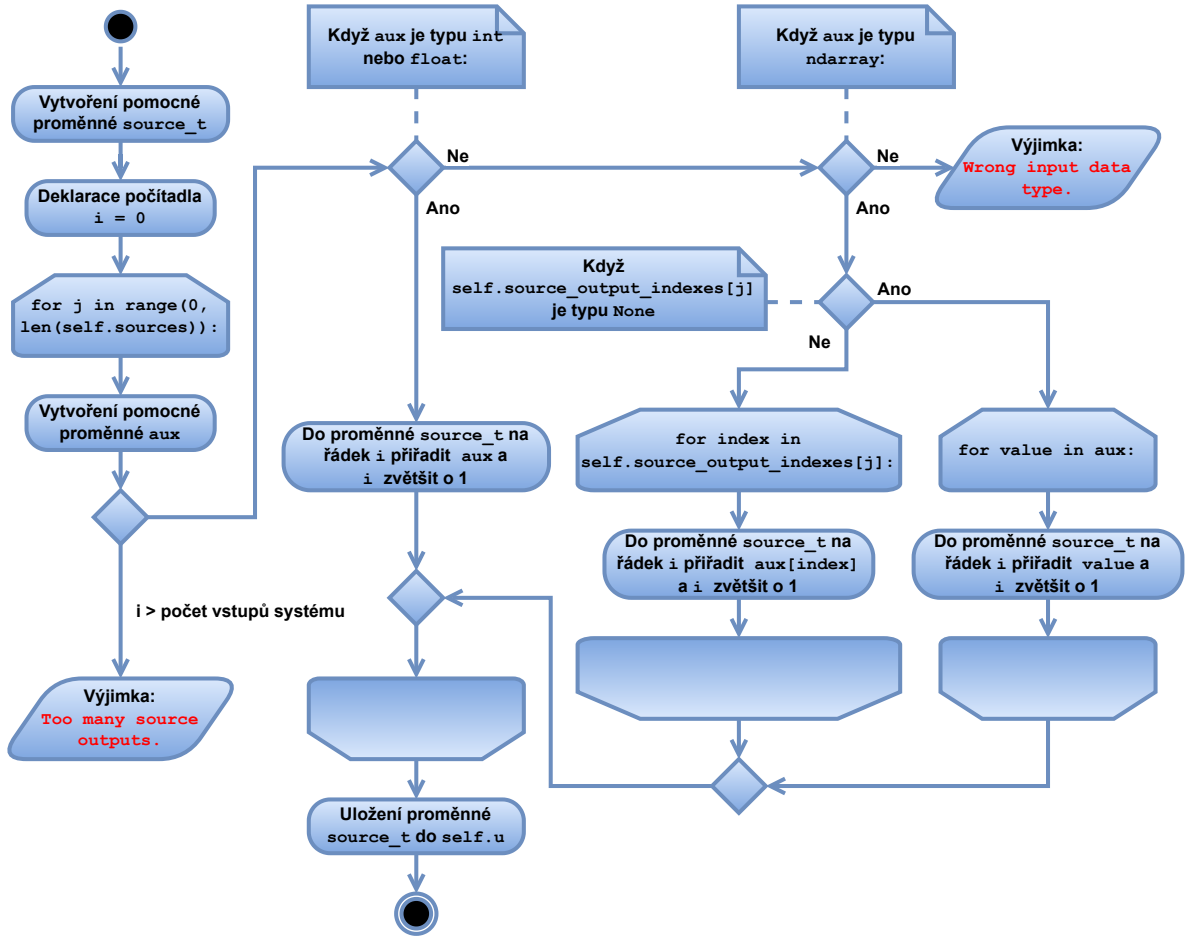
5.1.2 Metoda `update_input()`

Pomocí této metody je zjišťována hodnota vstupu (vstupů) v požadovaném čase. Požadovaným časem je rozuměna hodnota `self.t` představující aktuální čas daného systému. Kromě aktualizace vstupu je zde potřeba také jednotlivé vstupní hodnoty seřadit do vektoru vstupu `self.u`. Celý tento proces je znázorněn na obr. 5.3.

Po zavolání metody je nejprve deklarována lokální pomocná proměnná `source_t`, což je vektor o stejných rozměrech jako vektor vstupu `self.u`, a počítadlo `i`. Poté je pomocí cyklu `for` procházen seznam zdrojů `self.sources` s použitím počítadla `j`. Počítadlo `j` slouží pro procházení seznamů `self.sources` a `self.source_output_indexes`, zatímco počítadlo `i` slouží k procházení pomocné proměnné `source_t` a zajišťuje seřazení získaných hodnot vstupu ve vektoru vstupu. V každém kroku cyklu `for` je do pomocné proměnné `aux` přiřazena aktuální hodnota zdroje na indexu `j` a je zkontrolováno, zda již není zaplněn vektor vstupu (v takovém případě dochází k vyvolání výjimky).

Dále je třeba zjistit, zda výstupní funkce zdroje vrátila skalární či vektorovou hodnotu, což je realizováno pomocí přepínače `match case`. V případě skalární hodnoty je tato rovnou zapsána do pomocného vektoru `source_t` na příslušný index `i`. Po každém zápisu do `source_t` je počítadlo `i` inkrementováno.

V případě vektorové hodnoty je nejprve potřeba zjistit, které z hodnot tohoto vek-



Obr. 5.3: Diagram aktivit metody `update_input()`

toru mají být použity. Program tedy nejprve zkontroluje datový typ hodnoty v seznamu `self.source_output_indexes` na pozici `j`. Pokud je typu `None`, znamená to, že se mají použít všechny hodnoty vektoru. Pomocí cyklu `for` je tedy celý vektor prvek po prvku překopírován do vektoru `source_t`. Pokud jsou udány konkrétní indexy, pomocí cyklu `for` jsou vybrány pouze požadované hodnoty a opět jsou uloženy do pomocného vektoru `source_t`. Proces je opakován, dokud nejsou zpracovány všechny zdroje.

Posledním krokem této metody po proběhnutí cyklu `for` je uložení pomocné proměnné `source_t` do vektoru vstupu `self.u`, po němž je metoda `update_input()` ukončena.

5.1.3 Metody `linear_regression()`, `input_linear_regression()` a `output_linear_regression()`

Skupina těchto tří metod je využívána zejména metodami `output()`. U většiny systémů v tomto balíčku je metoda `output()` řešena tak, že při jejím zavolání prohledá archiv, kde se pokusí nalézt již vypočítanou hodnotu výstupu v požadovaném čase. Pokud taková hodnota v archivu není, je pomocí metody `linear_regression()` na základě hodnoty v čase předešlé a následující vypočítána rovnice přímky, která těmito body prochází. Pro

požadovaný čas je pak hodnota výstupu dopočítána z rovnice této přímky.

Některé systémy mají svůj stav popř. výstup popsaný funkcí času. U takových systémů lze zjistit hodnotu výstupu v požadovaném čase jednoduše zavoláním této funkce pro požadovaný čas.

U některých systémů je hodnota výstupu dána pouze hodnotou vstupu. V případě těchto systémů je nejvýhodnější najít hodnotu vstupu v požadovaném čase (přímo nebo proložením přímkou) a z této hodnoty teprve vypočítat požadovaný výstup.

Aby se zamezilo opakování podobného kódu, byla metoda `linear_regression()` vytvořená v rámci bakalářské práce upravena tak, že archiv, ve kterém je hledána požadovaná hodnota (archiv vstupu nebo výstupu), je jí předán jako parametr jednou z metod `input_linear_regression()` a `output_linear_regression()`. Tato metoda je tedy univerzální. Metody `output()` ostatních systémů potom volají jednu z metod `input_linear_regression()` a `output_linear_regression()`, které následně zavolají metodu `linear_regression()`, předají jí v parametru buď archiv vstupu nebo výstupu a poté její výsledky pouze vrátí.

5.2 Nové abstraktní třídy balíčku DynSyPy

Pro zachování modularity celého balíčku a aby bylo umožněno balíček nadále rozšiřovat, bylo vhodné rozdělit systémy (třídy reprezentující tyto systémy) do celků, které např. používají společnou metodu. K tomuto účelu bylo vytvořeno několik abstraktních tříd, čímž došlo k rozdělení systémů podle jejich charakteru do několika větví. Kromě rozsáhlejší třídy Machine, které je věnována kapitola 5.3, budou všechny nově vytvořené abstraktní třídy popsány v této kapitole.

Abstraktní třídy StateSpaceSystem a NonStateSpaceSystem

(a) Třída StateSpaceSystem

(b) Třída NonStateSpaceSystem

Obr. 5.4: Rozdělení systémů na state-space a non-state-space systémy

První rozdělení třídy System bylo provedeno vytvořením tříd StateSpaceSystem a NonStateSpaceSystem. Tím byly systémy rozděleny na ty, které jsou popsány pomocí vnitřního popisu, a na ostatní systémy.

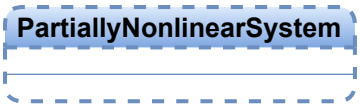
Třída StateSpaceSystem zahrnuje systémy, které jsou popsány soustavou stavových diferenciálních rovnic. Takové systémy je potřeba řešit některou z metod numerické inte-

grace. V balíku DynSyPy jsou implementovány dvě takové metody. První z nich je tzv. klasická metoda Runge-Kutta, neboli Runge-Kutta 4. řádu (RK4). Druhou je tzv. metoda Runge-Kutta-Fehlberg (zkráceně RKF), což je integrační metoda s adaptivním krokem. Obě tyto metody byly implementovány ve třídě System již v bakalářské práci [9] jako metody `runge_kutta_step()` (RK4) a `adaptive_step()` (RKF). Třída StateSpaceSystem upravuje chování metody `step()` deklarované ve třídě System tak, aby při simulaci používala metodu pro numerickou integraci `runge_kutta_step()`. Činnost metody `step()` zůstala nezměněna a je popsána v [9].

Ve třídě NonStateSpaceSystem je kromě metody `step()` upraveno také chování metody `adaptive_step()`. Je zde také deklarována nová metoda `update_state()`. Činnost metody `step()` se zde liší pouze v tom, že místo metody `runge_kutta_step()` je použita nová metoda `update_state()`. Ta je abstraktní a každá dceřinná třída tedy musí její chování dodefinovat.

Jelikož o tom, zda bude použita integrační metoda s konstantním nebo s adaptivním krokem, se rozhoduje globálně (ne pro každý systém zvlášť), je nutné zajistit, aby metodu `adaptive_step()` bylo možné zavolat pro libovolný systém. V rámci jedné simulace mohou být totiž společně modelovány systémy state-space i non-state-space. Z tohoto důvodu je metoda `adaptive_step()` deklarována už ve třídě System. Ve třídě NonStateSpaceSystem je však tato metoda přepsána tak, aby pouze zavolala metodu `step()`.

Abstraktní třída PartiallyNonLinearSystem



Obr. 5.5: Třída PartiallyNonlinearSystem

Abstraktní třída s pracovním názvem PartiallyNonLinearSystem je mateřskou třídou třídy Machine a zatím slouží pouze k vytvoření další odnože dynamických systémů. Nedefinuje zatím žádné atributy ani metody. Má představovat skupinu systémů, které jsou zařazeny do kategorie nelineárních systémů, ale lze je simulovat podobnými nástroji jako systémy lineární (např. je možné je popsat pomocí vnitřního popisu – stavový popis). Důvodem pro její vytvoření bylo odlišení těchto systémů od LTI systémů, které je v DynSyPy možné jednoduše simulovat pomocí třídy LinearSystem, a od ostatních nelineárních systémů.

Abstraktní třídy Source, UncontrolledSource a ControlledSource

Tyto třídy upravují chování zdrojů. Třída Source byla vytvořena v rámci bakalářské práce a je základem pro všechny ostatní zdroje. Oproti původní implementaci zde však došlo k drobným změnám. Původně zde bylo upraveno chování metod `step()` a

(a) Třída Source

(b) Třída ControlledSource

(c) Třída UncontrolledSource

Obr. 5.6: Rozdělení systémů představujících zdroje

`adaptive_step()`. To je nyní provedeno již ve třídě NonStateSpaceSystem. Dříve zde bylo také upraveno chování metody `output()`. Tato redefinice byla přesunuta do dceřinné třídy UncontrolledSource. Ostatní metody třídy Source zůstaly nezměněny.

Třída ControlledSource podle názvu zahrnuje systémy (zdroje), které je možné ovládat. Jelikož nebyly shledány žádné důvody, proč měnit chování definované rodičovskými třídami, nebylo zde potřeba žádné metody deklarovat ani upravovat. Jediným potomkem této třídy je zatím třída ControlledNPhaseSine, jež je popsána v kapitole 5.6.

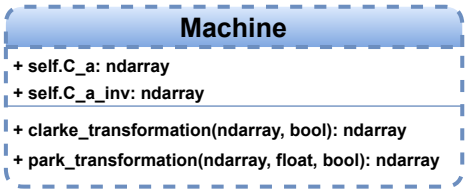
Veškeré zdroje vytvořené při zpracování bakalářské práce byly neřízené, proto jsou nyní potomky třídy UncontrolledSource. Neřízenými systémy jsou zde rozuměny systémy, které nemají vstup. V této třídě tedy bylo nutné upravit metodu `step()` a odstranit z ní kód pracující se vstupem. Dále zde byla upravena metoda `output()`. Protože všechny neřízené zdroje v balíku jsou popsány funkcí času, pro získání výstupu v požadovaném čase stačí zavolat metodu `source_function()` a předat jí hodnotu času v parametru.

Abstraktní třída Controllers

Obr. 5.7: Třída Controllers

Třída Controllers byla vytvořena z důvodu testování balíku. Jejími potomky jsou třídy IMScalarControl a PIController popsané v kapitolách 5.7 a 5.8. Jako jeden z testovacích příkladů byla zvolena realizace skalárního řízení ASM. Z tohoto důvodu bylo potřeba implementovat třídu pro realizaci principu skalárního řízení popsaného v kapitole 3 a také třídu pro vytvoření regulátoru. Konkrétně byl implementován PI regulátor s anti-windup úpravou. V této třídě nejsou deklarovány žádné metody ani atributy, slouží opět pouze pro vytvoření nové větve systémů a umožňuje tomuto druhu systémů v budoucnu dodefinovat nové společné vlastnosti.

5.3 Abstraktní třída Machine



Obr. 5.8: Třída Machine

Základním stavebním kamenem pro sestavování dynamických modelů elektrických strojů je abstraktní třída Machine. Jejím úkolem je definovat atributy a metody společné pro všechny elektrické stroje. Jejím potomkem je však zatím pouze třída SquirrelCageIM, nelze tedy s jistotou říci, co vše by mělo být ve třídě Machine obsaženo, aby byl tento úkol splněn. Zatím jsou zde definovány dva atributy a dvě metody.

```
self.C_a = 2 / 3 * np.array([[1, -1 / 2, -1 / 2],
                             [0, np.sqrt(3) / 2, -np.sqrt(3) / 2]])

self.C_a_inv = np.array([[1, 0],
                         [-1 / 2, np.sqrt(3) / 2],
                         [-1 / 2, -np.sqrt(3) / 2]])
```

Atributy `C_a` a `C_a_inv` využívá metoda `clarke_transformation()` podrobněji popsaná v části 5.3.1. Obě metody definované v této třídě představují matematické transformace hojně používané při modelování střídavých elektrických strojů, ale také při jejich řízení.

5.3.1 Metoda `clarke_transformation()`

Činností této metody je převod třífázového signálu do stojící souřadné soustavy α, β. K tomu je využito principu tzv. transformace Clarkeové někdy také nazývané jako transformace na prostorový vektor (její princip je popsán např. v [15], příp. [16]). V této práci byla využívána pouze amplitudově invariantní transformace, tato metoda je tedy implementována pouze jako amplitudově invariantní. Použitá hodnota konstanty k je tedy $2/3$. Oproti teorii jsou zde také použity pozměněné transformační matice. Ty mají původně rozměr 3×3 a kromě složek α a β je transformací získána také nulová složka. Té v simulaci není zapotřebí, výstupem tedy budou pouze složky α a β. Tím dojde ke snížení počtu řádků dopředné transformační matice na dva a bude vypadat následujícím způsobem [16]

$$\mathbf{C}_A = \frac{2}{3} \cdot \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \sqrt{\frac{3}{2}} & -\sqrt{\frac{3}{2}} \end{bmatrix} . \tag{5.1}$$

Ze stejného důvodu se také sníží počet sloupců transformační matice pro zpětnou transformaci: [16]

$$\mathbf{C}_A^{-1} = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & \sqrt{\frac{3}{2}} \\ -\frac{1}{2} & -\sqrt{\frac{3}{2}} \end{bmatrix} . \tag{5.2}$$

Transformační matice jsou v tomto tvaru přímo implementovány jako atributy třídy Machine `self.C_a` a `self.C_a_inv`, viz výše.

Metoda má dva parametry: `input_signal_vector` typu `ndarray` a `reverse` typu `bool`. První parametr představuje zpracovávaný signál (vektorový) a druhý nepovinný parametr určuje, zda bude provedena transformace dopředná nebo zpětná. Pokud je metodě předán pouze první parametr, je provedena dopředná transformace, protože parametr `reverse` je implicitně nastaven na hodnotu `False`. Implementace metody je provedena následujícím způsobem:

```
if not reverse:
    return self.C_a @ input_signal_vector
else:
    return self.C_a_inv @ input_signal_vector
```

Po zavolání metoda pouze zjistí na základě hodnoty parametru `reverse`, která transformace má být provedena (dopředná nebo zpětná), a poté vrátí výsledek maticového součinu příslušné transformační matice s proměnnou `input_signal_vector`.

5.3.2 Metoda `park_transformation()`

Metoda `park_transformation()` slouží pro převod mezi stojící souřadnou soustavou α, β a rotující souřadnou soustavou d, q. Činností této metody je provádění tzv. Parkovy transformace, jejíž princip je popsán např. v [15], příp. [16]. Vstupními parametry jsou vstupní vektor `input_signal_vector`, úhel `theta`, což je okamžitý úhel natočení osy d rotujícího souřadného systému vzhledem k ose α stojícího souřadného systému, a nepovinný parametr `reverse`, který opět určuje, zda bude provedena transformace dopředná nebo zpětná.

Metoda `park_transformation()` je v jazyce Python realizována podobně jako metoda `clarke_transformation()`. Na základě hodnoty parametru `reverse` je rozhodnuto o provedení dopředné nebo inverzní transformace a poté je parametr `input_signal_vector` jednoduše vynásoben příslušnou transformační maticí. Jelikož se matice v čase mění, je po každém zavolání metody `park_transformation()` počítána znovu. Realizace této metody je naznačena v kódu:

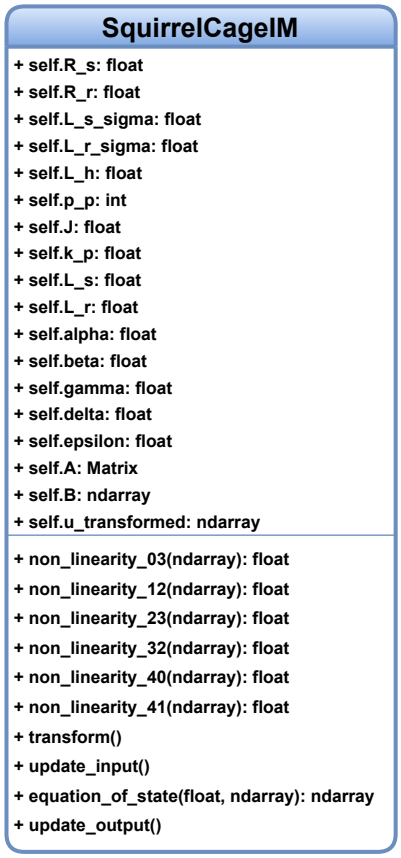
```
if not reverse:
    r_dq = np.array([[np.cos(theta), np.sin(theta)],
                     [-np.sin(theta), np.cos(theta)]])

    return r_dq @ input_signal_vector
else:
    r_dq_inv = np.array([[np.cos(theta), -np.sin(theta)],
                         [np.sin(theta), np.cos(theta)]])

    return r_dq_inv @ input_signal_vector
```

Transformační matice metody `park_transformation()` by také bylo možné realizovat s využitím třídy Matrix, která řeší problém proměnlivosti prvků matice v čase. Jelikož tato metoda byla vytvořena dříve, než byla implementována třída Matrix, byla zatím ponechána v tomto stavu. V rámci dalšího vývoje balíku lze zvážit její přepracování.

5.4 Třída SquirrelCageIM

Třída SquirrelCageIM slouží k simulaci asynchronního stroje s klecovou kotvou – kotvou nakrátko. Využívá tedy matematického modelu, který byl odvozen v kapitole 1.2. Tento matematický model tvořený soustavou stavových diferenciálních rovnic (1.98) je však obecnější a je možné jím simulovat více typů asynchronních strojů. Pro potřeby této třídy bylo vhodné jej zjednodušit.

V případě ASM s kotvou nakrátko platí, že rotorové napětí je rovno nule. Pro jednoduchost byl zatím také zanedbán koeficient tření B_{ω_m}, který umožňuje v simulacích uvažovat např. tření v ložiskách. Tření v motoru je ve skutečnosti závislé na rychlosti otáčení rotoru, popř. i jiných faktorech (teplota), a jeho modelování je poměrně složitá záležitost. V rámci této práce problematika tření řešena nebyla.



Obr. 5.9: Třída SquirrelCageIM

Po těchto úpravách se model zjednoduší do tvaru

$$\begin{aligned}
\frac{\mathrm{d}i_{\mathrm{s}_\alpha}}{\mathrm{d}t} &= -\alpha \cdot i_{\mathrm{s}_\alpha} + \beta \cdot \Psi'_{\mathrm{r}_\alpha} + \gamma \cdot \omega_{\mathrm{m}} \cdot \Psi'_{\mathrm{r}_\beta} + \delta \cdot u_{\mathrm{s}_\alpha} \ , \\
\frac{\mathrm{d}i_{\mathrm{s}_\beta}}{\mathrm{d}t} &= -\alpha \cdot i_{\mathrm{s}_\beta} - \gamma \cdot \omega_{\mathrm{m}} \cdot \Psi'_{\mathrm{r}_\alpha} + \beta \cdot \Psi'_{\mathrm{r}_\beta} + \delta \cdot u_{\mathrm{s}_\beta} \ , \\
\frac{\mathrm{d}\Psi'_{\mathrm{r}_\alpha}}{\mathrm{d}t} &= R'_{\mathrm{r}} \cdot \frac{L_{\mathrm{h}}}{L'_{\mathrm{r}}} \cdot i_{\mathrm{s}_\alpha} - \frac{R'_{\mathrm{r}}}{L'_{\mathrm{r}}} \cdot \Psi'_{\mathrm{r}_\alpha} - p_{\mathrm{p}} \cdot \omega_{\mathrm{m}} \cdot \Psi'_{\mathrm{r}_\beta} \ , \\
\frac{\mathrm{d}\Psi'_{\mathrm{r}_\beta}}{\mathrm{d}t} &= R'_{\mathrm{r}} \cdot \frac{L_{\mathrm{h}}}{L'_{\mathrm{r}}} \cdot i_{\mathrm{s}_\beta} + p_{\mathrm{p}} \cdot \omega_{\mathrm{m}} \cdot \Psi'_{\mathrm{r}_\alpha} - \frac{R'_{\mathrm{r}}}{L'_{\mathrm{r}}} \cdot \Psi'_{\mathrm{r}_\beta} \ , \\
\frac{\mathrm{d}\omega_{\mathrm{m}}}{\mathrm{d}t} &= -\varepsilon \cdot \Psi'_{\mathrm{r}\beta} \cdot i_{\mathrm{s}\alpha} + \varepsilon \cdot \Psi'_{\mathrm{r}\alpha} \cdot i_{\mathrm{s}\beta} - \frac{1}{J} \cdot M_{\mathrm{z}} \ .
\end{aligned} \tag{5.3}$$

Nyní lze zavést následující substituce: $x_1 = i_{\mathrm{s}_\alpha}$, $x_2 = i_{\mathrm{s}_\beta}$, $x_3 = \Psi_{\mathrm{r}_\alpha}$, $x_4 = \Psi_{\mathrm{r}_\beta}$, $x_5 = \omega_{\mathrm{m}}$, $u_1 = u_{\mathrm{s}_\alpha}$, $u_2 = u_{\mathrm{s}_\beta}$, $u_3 = M_{\mathrm{z}}$. Převedením matematického modelu (5.3) do maticového

tvaru za použití těchto substitucí vychází vztah

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \underbrace{\begin{bmatrix} -\alpha & 0 & \beta & \gamma \cdot x_5 & 0 \\ 0 & -\alpha & -\gamma \cdot x_5 & \beta & 0 \\ R_\mathrm{r} \cdot \dfrac{L_\mathrm{h}}{L_\mathrm{r}} & 0 & -\dfrac{R_\mathrm{r}}{L_\mathrm{r}} & -p_\mathrm{p} \cdot x_5 & 0 \\ 0 & R_\mathrm{r} \cdot \dfrac{L_\mathrm{h}}{L_\mathrm{r}} & p_\mathrm{p} \cdot x_5 & -\dfrac{R_\mathrm{r}}{L_\mathrm{r}} & 0 \\ -\varepsilon \cdot x_4 & \varepsilon \cdot x_3 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{A}} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} + \underbrace{\begin{bmatrix} \delta & 0 & 0 \\ 0 & \delta & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -\dfrac{1}{J} \end{bmatrix}}_{\mathbf{B}} \cdot \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}. \tag{5.4}$$

Tento tvar je formálně shodný se stavovou reprezentací LDS, viz rovnice (2.10), lze tedy zjistit matice **A** a **B**, jak je naznačeno.

Matice **B** se skládá pouze z konstant, je tedy možné ji v tomto tvaru přímo implementovat jako jeden z atributů třídy SquirrelCageIM jako proměnnou typu `ndarray`. Implementace matice **A** je však problematičtější, neboť červeně vyznačené prvky této matice nejsou konstantní a mění se v čase. Řešení tohoto problému nabízí třída Matrix, která byla vyvinuta právě za účelem vyřešení tohoto problému. Jejímu podrobnějšímu popisu se věnuje část 5.5.

Nejběžnějšími veličinami, které jsou na motoru měřeny, jsou proudy tekoucí jednotlivými fázemi statoru a rychlost otáčení rotoru. Tyto veličiny proto byly zvoleny jako výstup ASM. Výstupní rovnice pro model ASM v soustavě α, β v maticovém tvaru potom bude

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\mathbf{C}} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \tag{5.5}$$

Před tím, než bude přistoupeno k popisu metod této třídy, je vhodné zmínit, co je potřeba k vytvoření instance třídy SquirrelCageIM. Konstruktor požaduje jeden povinný parametr `parameters` typu `dict` (slovník). V této proměnné by měly být hodnoty charakterizující konkrétní ASM. Povinný parametr `parameters` musí pro bezchybnou funkci obsahovat správné hodnoty. Vzor pro správné vytvoření požadovaného slovníku vypadá následujícím způsobem:

```
motor_params = {
    "R_s": R_s_value,
    "R_r": R_r_value,
    "L_s_sigma": L_s_sigma_value,
    "L_r_sigma": L_r_sigma_value,
    "L_h": L_h_value,
```

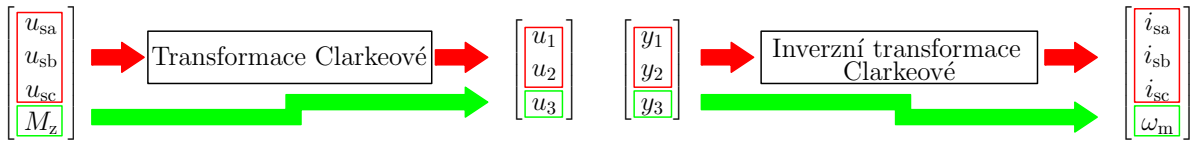
```
    "p_p": p_p_value,
    "U_s_N": U_s_N_value,
    "f_s_N": f_s_N_value,
    "J": J_value
}
```

Slovníky jsou neuspořádané kolekce dvojic klíč-hodnota. Uvozují se pomocí složených závorek a jednotlivé dvojice uvnitř se oddělují čárkou. Před dvojtečkou je klíč, za dvojtečkou je jemu odpovídající hodnota. V případě balíčku DynSyPy všechny třídy, jimž jsou parametry předávány pomocí slovníku, vyžadují, aby klíče byly typu `str` (string, textový řetězec). Konstruktor třídy SquirrelCageIM potřebuje ve slovníku najít všechny výše uvedené klíče a jim odpovídající hodnoty typu `float`.

Nyní je vhodné popsat metody této třídy. Metoda `update_input()` pouze zavolá metodu rodiče a potom zavolá metodu `transform()`, která přizpůsobí vektor vstupu potřebám této třídy (viz část 5.4.1). Není tedy nutné ji popisovat ve zvláštní části. Ostatní metody jsou popsány v následujících sekcích.

5.4.1 Metody `transform()` a `update_output()`

Tyto dvě metody pracují velmi podobně. Zatímco metoda `transform()` je používána pro úpravu vektoru vstupů, metoda `update_output()`, jak název napovídá, upravuje výstupní vektor. Zásadní problém, který bylo potřeba pro tyto dvě metody vyřešit, je skutečnost, že část vektoru vstupu/výstupu je potřeba transformovat a část nikoliv.



(a) Činnost metody `transform()` **(b)** Činnost metody `update_output()`

Obr. 5.10: Znázornění činnosti metod `transform()` a `update_output()`

Jak je vidět na obrázku 5.10, v případě metody `transform()` je vektor vstupu transformací zmenšen ze čtyř prvků na tři (obr. 5.10a). Je tedy potřeba první tři prvky vektoru transformovat dopřednou transformací Clarkeové a výsledek rozšířit o zbylý prvek původního vektoru vstupu. Realizace v jazyce Python je provedena s využitím funkce `vstack()` z balíku Numpy:

```
self.u_transformed = np.vstack(
    (self.clarke_transformation(self.u[0:3, :]), self.u[-1, :]))
```

Naproti tomu metoda `update_output()` musí mít jako výstup opět vektor o čtyřech prvcích. Výstup definovaný vztahem (5.5) má ale pouze tři prvky. Je tedy potřeba první

dva prvky transformovat inverzní transformací Clarkeové zpět do fázových souřadnic a tento výsledek opět sloučit s posledním prvkem výstupního vektoru. Vzhledem k tomu, že výstupem ASM jsou přímo stavové veličiny, je transformace stavu na výstup triviální a tedy matice C je jednotková matice.

Obr. 5.11: Skutečná realizace metody `update_output()`

V jazyce Python je to realizováno následujícím způsobem:

```
self.y = np.vstack(
    (self.clarke_transformation(self.x[0:2, :], reverse=True),
     self.x[-1, :]))
```

Na tomto místě je vhodné ještě dodat, že až bude balíček rozšířen o modely dalších strojů, které mohou být počítány v rotujícím souřadném systému d, q, bude nutné doplnit kód, který zjistí, které transformace budou při simulaci potřeba. Tyto metody bude nutné potom upravit tak, aby prováděly všechny potřebné transformace automaticky.

5.4.2 Metoda `equation_of_state()`

Metoda `equation_of_state()` je definována již ve třídě System, její chování si ale definují až její dceřinné třídy. Jak název napovídá, v této metodě by měla být implementována soustava stavových diferenciálních rovnic popisujících asynchronní stroj s klecovou kotvou. Nejjednodušším způsobem, jak toto realizovat, je prosté přepsání soustavy rovnic (5.3) do kódu v jazyce Python, což by mohlo vypadat například takto:

```
x1 = x[0, 0]
x2 = x[1, 0]
x3 = x[2, 0]
x4 = x[3, 0]
x5 = x[4, 0]

u1 = self.u_transformed[0, 0]
u2 = self.u_transformed[1, 0]
u3 = self.u_transformed[2, 0]
```

```
dx1 = -self.alpha * x1 + self.beta * x3 + self.gamma * x5 * x4\
      + self.delta * u1
dx2 = -self.alpha * x2 - self.gamma * x5 * x3 + self.beta * x4\
      + self.delta * u2
dx3 = self.R_r * (self.L_h / self.L_r) * x1 - (self.R_r / self.L_r) * x3\
      - self.p_p * x5 * x4
dx4 = self.R_r * (self.L_h / self.L_r) * x2 + self.p_p * x5 * x3\
      - (self.R_r / self.L_r) * x4
dx5 = -self.epsilon * x4 * x1 + self.epsilon * x3 * x2 - u3 / self.J

dx = np.array([[dx1],
               [dx2],
               [dx3],
               [dx4],
               [dx5]])

return dx
```

Realizace tímto způsobem je sice funkční, ale jsou zde některé nedostatky tohoto řešení. Kromě nepřehlednosti kódu je to hlavně cyklická deklarace čtrnácti nových lokálních proměnných při každém zavolání této metody a v tomto případě také provádění několika operací dělení. Problém dělení by bylo možné vyřešit deklarací nových proměnných v konstruktoru, což by ale mělo vliv na přehlednost kódu.

Efektivnějším způsobem je soustavu stavových diferenciálních rovnic přepsat do maticového tvaru, viz vztah (5.4), a implementovat ji tímto způsobem. Tím se kód výrazně zjednoduší a zpřehlední. Zároveň si lze všimnout, že všechny prvky matic, kde se provádí dělení, jsou konstantní. Jelikož deklarace těchto matic probíhá již v konstruktoru třídy SquirrelCageIM, dělení se provede pouze jednou a v průběhu simulace se už vůbec neprovádí. Maticový tvar stavové rovnice je tedy implementován následujícím způsobem:

```
self.A.eval(x)

dx = self.A.matrix @ x + self.B @ self.u_transformed

return dx
```

Nejprve je provedena aktualizace nelineární systémové matice `self.A` podle hodnot současného stavu systému zavoláním její metody `eval()`, viz sekce 5.5. Poté je vypočten vektor diferenciálu stavu a ten je následně metodou vrácen. Tento způsob realizace je velmi podobný způsobu použitému pro tuto metodu ve třídě LinearSystem, viz [9], popř. [10].

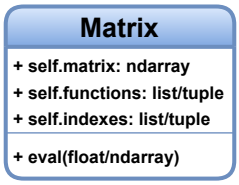
5.4.3 Metody `non_linearity_03()` až `non_linearity_41()`

Tyto metody popisují nelinearity v modelu asynchronního stroje ve stojícím souřadném systému α, β. Ve vztahu (5.4) jsou v matici **A** červeně vyznačeny všechny nelinearity, které je potřeba pomocí těchto metod popsat.

Názvy těchto metod byly voleny tak, že dvojčíslí uvedené na konci názvu metody představuje pozici, kde se nelinearita v matici **A** nachází (indexováno od nuly). Podle názvů je tedy možné vyplnit seznam indexů a tomu odpovídající seznam funkcí. Tyto seznamy je následně možné předat konstruktoru třídy Matrix (popsána v následující sekci). Seznamy nelinearit a indexů jsou deklarovány následujícím způsobem:

```
non_linearity_functions =\
    [self.non_linearity_03, self.non_linearity_12, self.non_linearity_23,
     self.non_linearity_32, self.non_linearity_40, self.non_linearity_41]

non_linearity_indexes = [[0, 3], [1, 2], [2, 3], [3, 2], [4, 0], [4, 1]]
```

5.5 Třída Matrix

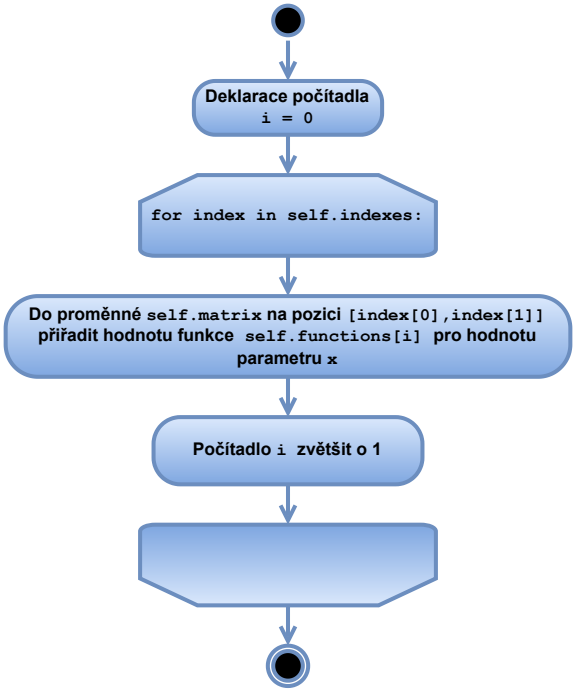


Obr. 5.12: Třída Matrix

Třída Matrix byla vytvořena pro potřeby třídy SquirrelCageIM, kde bylo potřeba vyřešit problematiku v čase se měnících prvků systémové matice **A**. Její struktura je znázorněna na obr. 5.12. Z tohoto obrázku je patrné, že třída Matrix obsahuje tři atributy. Prvním je `self.matrix` typu `ndarray`, ve kterém je uložena matice, jejíž prvky se mohou v čase měnit. Informace o tom, které prvky této matice se mají měnit, jsou uloženy v atributu `self.indexes`, který může být typu `list` nebo `tuple` (seznam nebo n-tice). Poslední atribut `self.functions` (také seznam nebo n-tice) říká, jak se tyto prvky mají měnit. Obsahuje přímo funkce, které tyto změny popisují. Hodnoty všech tří atributů jsou nastavovány již při vytváření objektu, kdy je potřeba předat je konstruktoru jako parametry.

Třída Matrix také obsahuje metodu `eval()`, jejímž úkolem je po zavolání aktualizovat všechny prvky matice `self.matrix` pro hodnotu `x`, která je jí předána jako parametr. Hodnota `x` může být typu `float` nebo `ndarray`. Popis metody `eval()` je uveden v části 5.5.1.

5.5.1 Metoda `eval()`

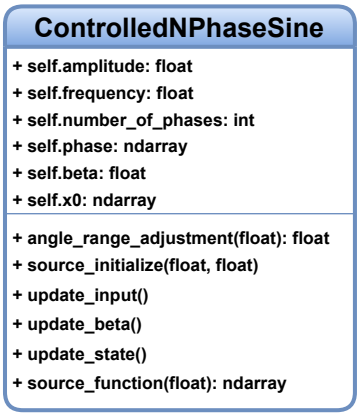


Obr. 5.13: Diagram aktivit metody `eval()`

Činnost metody `eval()` znázorňuje obr. 5.13. Po jejím zavolání je počítadlo `i` nastaveno na hodnotu 0. Potom je pomocí cyklu `for` procházen seznam `self.indexes`, kde jsou uloženy indexy, na kterých se prvky matice `self.matrix` mění v čase. V každém kroku cyklu `for` je zavolána funkce uložená v seznamu `self.functions` na pozici `i` a v parametru je jí předána hodnota `x`. Návratová hodnota této funkce je poté uložena do matice `self.matrix` na pozici danou hodnotou `index`. Na konci každého kroku je inkrementováno počítadlo `i` a proces se opakuje, dokud nejsou všechny nelinearity aktualizovány pro požadovanou hodnotu `x`.

5.6 Třída ControlledNPhaseSine

Instancemi této třídy jsou n-fázové zdroje sinusového signálu. Tyto zdroje slouží jako zjednodušená náhrada měničů (např. napěťový střídač), jejichž implementace by byla velmi komplikovaná. Vstupem do těchto zdrojů jsou požadavky na amplitudu a frekvenci výstupního signálu, které mohou být generovány např. algoritmem skalárního řízení.

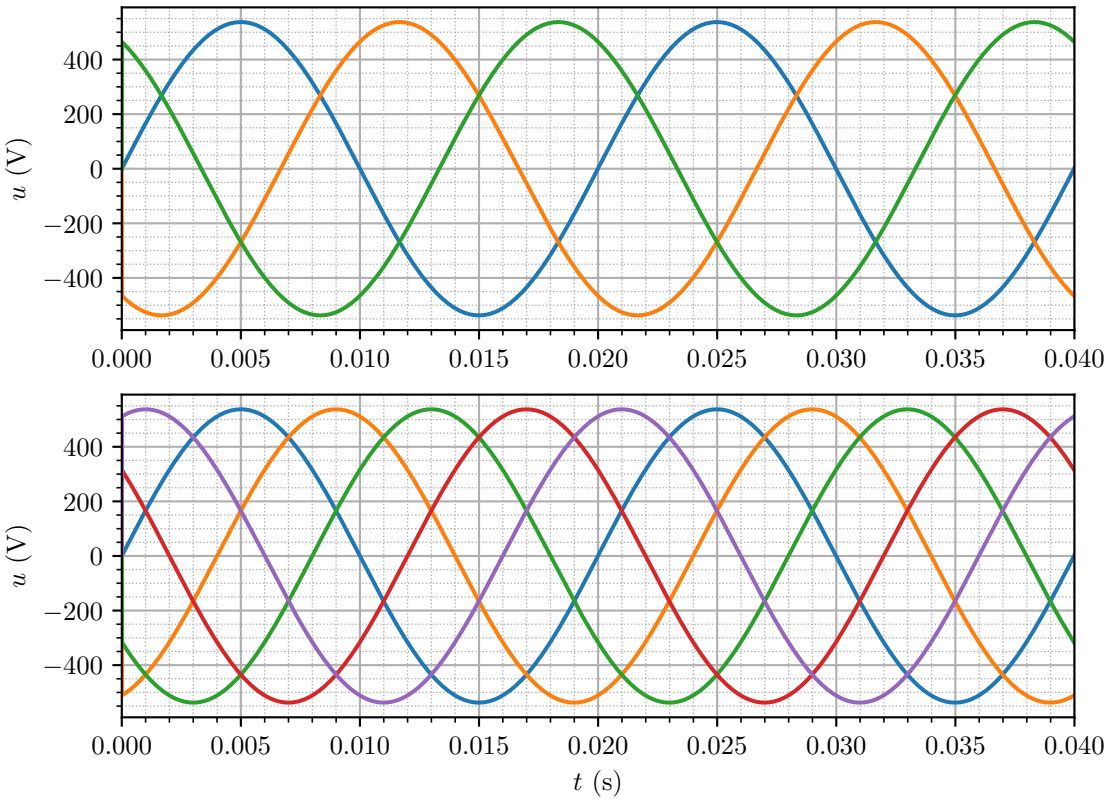


Obr. 5.14: Třída ControlledNPhaseSine

Konstruktor třídy ControlledNPhaseSine má jeden povinný parametr, který je typu `dict`. Tímto slovníkem jsou instanci předány informace o amplitudě, frekvenci, počtu fází a počátečním fázovém posunu. Vzor pro vytvoření slovníku vypadá následujícím způsobem:

```
source_params = {
    "amplitude": amplitude_value,
    "frequency": frequency_value,
    "number_of_phases": number_of_phases_value,
    "phase": phase_value
}
```

V této třídě je upraveno několik metod mateřských tříd. Metoda `update_input()` je zde upravena tak, že nejprve zavolá metodu rodiče, čímž dojde k aktualizaci vektoru vstupu. Poté jsou jednotlivé prvky vektoru vstupu jednoduše přiřazeny do atributů `self.amplitude` a `self.phase`.V metodě `update_state()` je naopak nejprve zavolána metoda `update_beta()`, která bude popsána v části 5.6.1, a teprve potom je zavolána metoda rodiče. V rámci metody `source_function()` je provedena úprava hodnoty `self.beta` pomocí metody `angle_range_adjustment()`, která je popsána v části 5.6.2. Takto upravená hodnota je přičtena k vektoru `self.phase` a z těchto hodnot fázových posunů je vypočítána pomocí funkce sinus nová hodnota stavu (vektorová) a ta je metodou vrácena. Popis ostatních metod je rozsáhlejší, jsou proto popsány níže ve zvláštních sekcích.



Obr. 5.15: Ukázka výstupu objektů třídy ControlledNPhaseSine, nahoře třífázový zdroj, dole pětifázový zdroj

5.6.1 Metoda `update_beta()`

Aby bylo možné se v těchto simulacích přiblížit reálné aplikaci v pohonech, je nutné ošetřit, aby i při skokových změnách požadavku na frekvenci výstupního signálu docházelo ke změnám výstupního signálu plynule. Kdyby zde byla funkce sinus implementována obvyklým způsobem

$$u(t) = U_\mathrm{m} \cdot \sin\left(2 \cdot \pi \cdot f \cdot t + \varphi\right) , \tag{5.6}$$

pak při skokové změně požadované frekvence dochází ke skokové změně ve fázi výstupního signálu, jako je uvedeno na horním průběhu na obr. 5.16.

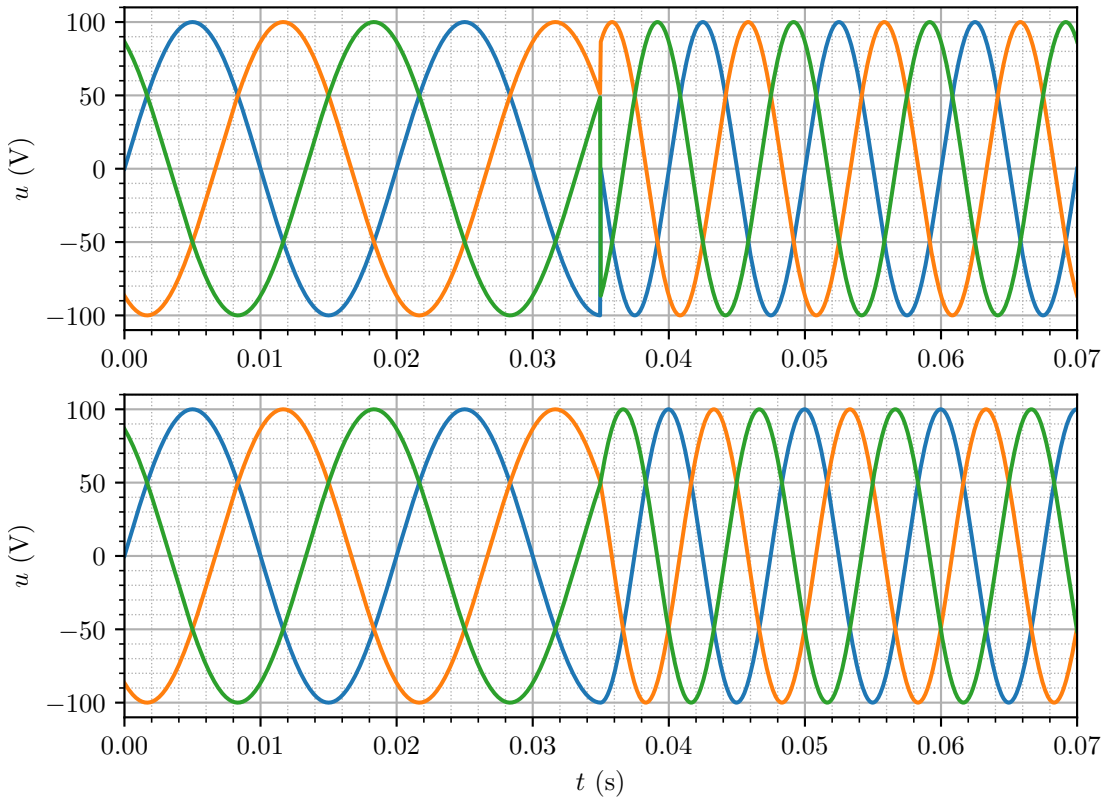
Aby při skokové změně požadované frekvence nedocházelo také ke skokové změně fáze výstupního signálu, byl zde implementován princip tzv. rotujícího úhlu β. Funkci lze po této úpravě zapsat jako

$$u(t) = U_\mathrm{m} \cdot \sin\left(\beta + \varphi\right) . \tag{5.7}$$

Úhel β je potom v metodě `update_beta()` počítán následujícím způsobem:

```
self.beta = self.beta + 2 * np.pi * self.dt * self.frequency
```

Je patrné, že při výpočtu funkce sinus nikdy nemůže dojít ke skokům ve fázi výstupního signálu, protože změna frekvence se projeví pouze na velikosti přírůstku úhlu β. Na obr.



Obr. 5.16: Realizace n-fázového zdroje, nahoře běžný způsob, dole využití rotujícího úhlu β

5.16 dole je vidět, že ke změně frekvence výstupního signálu dochází plynule.

5.6.2 Metoda `angle_range_adjustment()`

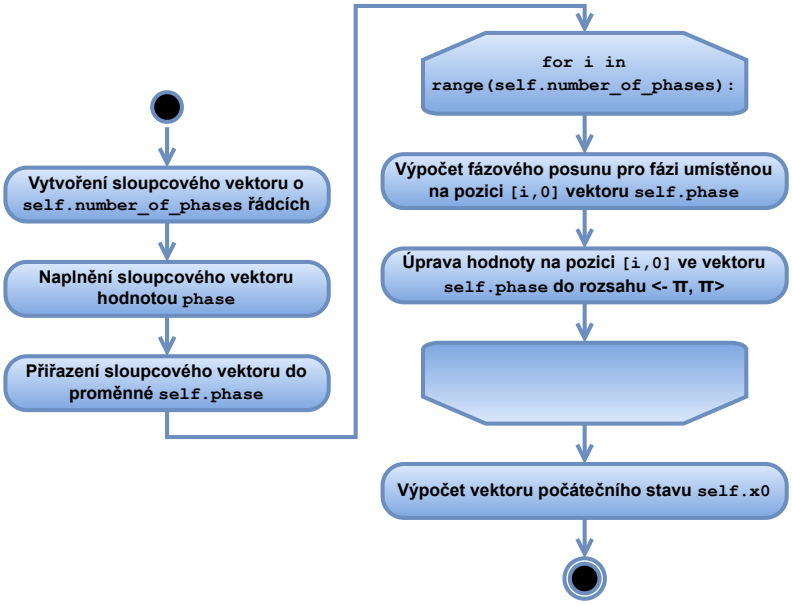
Pomocí této metody je zajištěno, že hodnota rotujícího úhlu β se bude pohybovat pouze v rozsahu $< -\pi, \pi >$. Ten je metodě předán v parametru. Pokud je jeho hodnota větší než π, je od něho odečtena hodnota $2 \cdot \pi$. Pokud je naopak hodnota úhlu β menší než $-\pi$, je k němu hodnota $2 \cdot \pi$ přičtena.

```
if angle > np.pi:
    angle = angle - 2 * np.pi

if angle < -np.pi:
    angle = angle + 2 * np.pi

return angle
```

Důvodem, proč je potřeba omezit rozsah hodnot úhlu β, je způsob interpretace hodnoty typu `float` v paměti počítače. Kdyby hodnota tedy nebyla omezena, po delší době běhu programu by došlo k situaci, kdy by hodnota úhlu β byla tak velká, že hodnota

přírůstku např. v řádu 10^{-5} by se po přičtení na výsledku neprojevila, což by způsobilo nesprávný chod programu.

5.6.3 Metoda `source_initialize()`



Obr. 5.17: Diagram aktivit metody `source_initialize()`

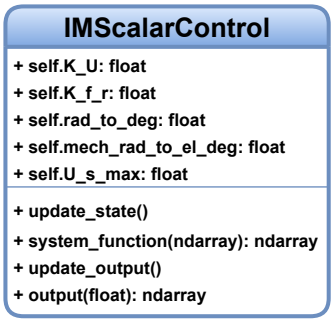
Již při vytváření objektu tohoto zdroje je potřeba nastavit správný počet fází a nastavit podle nich správné počáteční hodnoty vektoru stavu. To je realizováno pomocí metody `source_initialize()`. Tento proces znázorňuje diagram na obr. 5.17.

Metodě jsou předány dva parametry typu `float`: `phase` a `t0`. Parametr `phase` udává počáteční fázový posun první fáze sinusového průběhu. Parametr `t0` je počáteční čas simulace a v konstruktoru je implicitně nastaven na hodnotu 0. Atributu `self.phase` je přiřazen sloupcový vektor naplněný hodnotou parametru `phase`. Následně je pomocí cyklu `for` vektor `self.phase` upravován tak, aby jednotlivé fáze zdroje měly mezi sebou správný úhel (úměrný počtu fází). Za použití metody `angle_range_adjustment()` jsou zároveň jednotlivé hodnoty fázových posunů upraveny do rozsahu $< -\pi, \pi >$. Po proběhnutí cyklu `for` je do atributu `self.x0` přiřazen vektor stavu dopočítaný pro hodnotu počátečního času `t0`.

5.7 Třída IMScalarControl

Jak již bylo zmíněno v kapitole 5.2, tato třída a také třída PIController byly vytvořeny zejména pro účely testování balíčku DynSyPy, konkrétně na příkladu návrhu skalárního řízení ASM. Třída IMScalarControl je jedním ze dvou potomků třídy Controllers.

Konstruktor třídy IMScalarControl má opět pouze jeden povinný parametr typu `dict`,



Obr. 5.18: Třída IMScalarControl

který obsahuje potřebná data pro nastavení instance na konkrétní asynchronní stroj. Zároveň je navržen tak, aby pro jeho správné nastavení stačilo předat mu stejný slovník, který sloužil pro nastavení parametrů asynchronního stroje. Požadované náležitosti tohoto slovníku již byly uvedeny v sekci 5.4, není tedy potřeba je zde vypisovat znovu. V konstruktoru jsou tedy vypočítány všechny atributy této třídy. Jsou jimi konstanty K_U a K_{f_r}, jejichž odvození je uvedeno v kapitole 3, a které jsou nezbytné pro realizaci skalárního řízení. Také jsou zde deklarovány konstanty pro přepočet mezi úhlovou frekvencí v $\mathrm{rad} \cdot \mathrm{s}^{-1}$ a frekvencí v Hz a pro přepočet mechanické úhlové frekvence v $\mathrm{rad} \cdot \mathrm{s}^{-1}$ na elektrickou frekvenci v Hz. Posledním atributem je velikost saturační hodnoty požadavku na amplitudu výstupního napětí měniče počítané z hodnoty jmenovitého napětí motoru.

```
self.K_U = (parameters["U_s_N"] * np.sqrt(2)) / parameters["f_s_N"]
self.K_f_r =\
    (parameters["U_s_N"] * np.sqrt(2) * parameters["R_s"]) /\
    (parameters["f_s_N"] * parameters["R_r"])

self.rad_to_deg = 1 / (2 * np.pi)
self.mech_rad_to_el_deg = parameters["p_p"] / (2 * np.pi)

self.U_s_max = parameters["U_s_N"] * np.sqrt(2)
```

Třída IMScalarControl je zatím jedna ze dvou, kde metoda `output()` nepracuje s archivem výstupu. Při zavolání metody `output()` se nejprve ověřuje, zda pro požadovaný čas je v archivu vstupu uložena hodnota. Pokud ano, je z ní pomocí metody `system_function()` rovnou vypočítána hodnota výstupu, která je následně metodou vrácena. Pokud ne, je to řešeno pomocí metody `input_linear_regression()`, která je popsána v sekci 5.1.3.

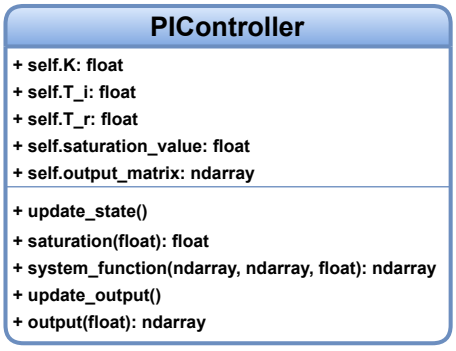
Činností metody `update_state()` je prosté zavolání metody `system_function()`, které je jako parametr předán vektor vstupu `self.u`, a uložení návratové hodnoty do vektoru stavu `self.x`. Metoda `update_output()` po zavolání pouze přiřazuje vektor stavu `self.x` do vektoru výstupu `self.y`.

5.7.1 Metoda `system_function()`

Celý princip skalárního řízení popsaný v kapitole 3 je implementován právě v této metodě. Na základě vektoru vstupu, který je metodě předán jako parametr, je vypočítán vektor stavu. Jak bylo řečeno výše, hodnoty stavu jsou shodné s hodnotami výstupu a jedná se o požadavek na amplitudu statorového napětí a požadavek na frekvenci statorového napětí. Realizace v DynSyPy je provedena následujícím způsobem:

```
x = np.zeros([2, 1])

f_r = self.rad_to_deg * u[0][0]
f_me = self.mech_rad_to_el_deg * u[1][0]

x[1][0] = f_me + f_r
x[0][0] = self.K_f_r * abs(f_r) + self.K_U * abs(x[1][0])

if x[0][0] > self.U_s_max:
    x[0][0] = self.U_s_max

return x
```

5.8 Třída PIController



Obr. 5.19: Třída PIController

Druhým potomkem třídy Controllers a zároveň poslední třídou vytvořenou v rámci balíčku DynSyPy je třída PIController. Ta byla implementována také za účelem testování balíku na příkladu skalárního řízení asynchronního stroje. Jejími instancemi jsou regulátory typu PI se zabudovanou anti-windup ochranou založenou na principu back-calculation popsanou v kapitole 2.1.1.

Třída PIController definuje pět atributů. Atribut `self.K` představuje proporční zesílení a `self.T_i` je časová konstanta integrační složky regulátoru. S využitím atributu

`self.saturation_value` je možné vytvořit model saturace aktuátoru, který je realizován pomocí metody `saturation()` (viz část 5.8.1). Hodnoty těchto tří atributů jsou opět předány konstruktoru jako parametr typu `dict`, jehož požadované náležitosti znázorňuje následující kód:

```
PI_controller_params = {
    "K": K_value,
    "T_i": T_i_value,
    "saturation_value": saturation_value
}
```

Atribut `self.T_r` je z důvodu jednoduchosti nastaven na polovinu integrační časové konstanty. Posledním atributem je `self.output_matrix` představující matici $\mathbf{C}$ výstupní rovnice systému dané vztahem (2.11), kterému je přiřazena matice [0 0 1].

Regulátor definovaný touto třídou má dva vstupy a jeden výstup. Vstupem do regulátoru je požadovaná hodnota regulované veličiny a její skutečná hodnota. Výstupem z regulátoru je pak řídicí veličina.

Činnost metod `update_state()`, `update_output()` a `output()` je podobná jako u třídy IMScalarControl. Metoda `update_state()` pracuje stejným způsobem s metodu `system_function()`, která ale vyžaduje více parametrů (viz část 5.8.2). Metoda pro aktualizaci výstupu `update_output()` nepřiřazuje výstupnímu vektoru přímo vektor stavu, ale vektor výstupu je počítán výstupní rovnicí systému danou vztahem (2.11). Metoda `output()` pracuje stejným způsobem jako ve třídě IMScalarControl, její úprava byla však nutná, protože pracuje s jinou metodou `system_function()` požadující tři parametry.

5.8.1 Metoda `saturation()`

Metoda `saturation()` představuje model aktuátoru, jehož výstup je omezen na nějakou konečnou výstupní hodnotu. Tato saturace aktuátoru může mít fyzikální základ (např. výkonový tranzistor na výstupu lineárního zdroje nelze více otevřít), nebo může být zavedena uměle. Toho je využito např. u skalárního řízení, kde je omezována maximální hodnota rotorové frekvence. Tím je zajištěno, že stroj bude vždy pracovat v lineární části momentové charakteristiky daleko od tzv. momentu zvratu. Tomu odpovídá hodnota tzv. kritické rotorové frekvence f_{rk}. Pro omezení rotorové frekvence tedy musí být vždy voleno $f_{\mathrm{r_{MAX}}} < f_{\mathrm{rk}}$.

Činnost této metody je velice přímočará. Pomocí dvou podmínek je zkontrolováno, jestli je hodnota `y` předaná metodě jako parametr v rozmezí $\pm$`self.saturation_value`. Pokud ano, je hodnota `y` rovnou vrácena, v opačném případě je její velikost omezena.

```
if y > self.saturation_value:
    return self.saturation_value
```

```
elif y < -self.saturation_value:
    return -self.saturation_value
else:
    return y
```

Z kódu je patrné, že tato saturace je symetrická vzhledem k hodnotě 0.

5.8.2 Metoda `system_function()`

Jak bylo výše zmíněno, metoda `system_function()` požaduje ke své činnosti tři parametry. Jsou jimi kromě vektoru vstupu systému také vektor stavu systému a aktuální velikost kroku systému. Pomocí vektoru vstupu je nejprve vypočítána hodnota regulační odchylky a zní je následně vypočítána hodnota proporční části regulačního zásahu.

```
e = u[0][0] - u[1][0]

proportional_part = self.K * e
x[0][0] = x[0][0] + (self.K / self.T_i * e - x[1][0]) * dt

y = proportional_part + x[0][0]

x[2][0] = self.saturation(y)

x[1][0] = (y - x[2][0]) / self.T_r  # anti-windup

return x
```

Jelikož se v regulátoru nachází integrační část a také anti-windup ochrana, které ke své správné činnosti potřebují znát svoji hodnotu z předchozího kroku, byly tyto hodnoty umístěny do vektoru stavu systému spolu s hodnotou na výstupu regulátoru. Z kódu vyplývá, že prvním prvkem vektoru stavu je vypočítaná hodnota integrační složky, druhým je hodnota zásahu anti-windup ochrany a posledním prvkem je hodnota výstupu z regulátoru po omezení hodnoty pomocí metody `saturation()`.

Kapitola 6

Ověření funkčnosti balíčku

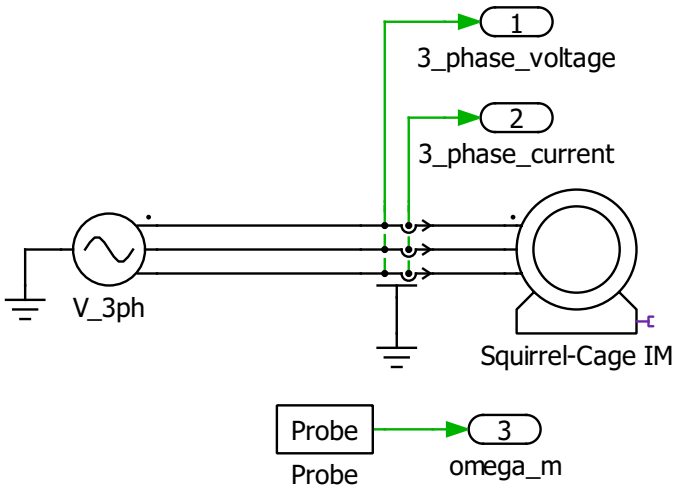
V této kapitole bude demonstrována funkčnost balíčku DynSyPy. Na následujících dvou testovacích příkladech bude předvedeno, jakým způsobem balík pracuje. Bude také provedeno porovnání výsledků získaných pomocí balíčku DynSyPy s výsledky získanými pomocí nástroje Matlab Simulink. V prvním testovacím příkladu bude simulováno přímé připojení nezatíženého ASM s kotvou nakrátko ke zdroji napětí. Druhým testovacím příkladem bude realizace skalárního řízení téhož motoru.

Pro oba testovací příklady byl použity následující parametry asynchronního stroje:

$$\begin{aligned} R_{\mathrm{s}} &= 1,617\ \Omega\ , \\ R'_{\mathrm{r}} &= 1,609\ \Omega\ , \\ L_{\mathrm{s}\sigma} &= 8,5\ \mathrm{mH}\ , \\ L'_{\mathrm{r}\sigma} &= 8,5\ \mathrm{mH}\ , \\ L_{\mathrm{h}} &= 134,4\ \mathrm{mH}\ , \\ p_{\mathrm{p}} &= 2\ , \\ U_{\mathrm{sN}} &= 3 \times 380\ \mathrm{V}\ , \\ f_{\mathrm{sN}} &= 50\ \mathrm{Hz}\ , \\ J &= 0,03\ \mathrm{kg} \cdot \mathrm{m}^2\ . \end{aligned}$$

Jedná se o hodnoty používané pro simulaci asynchronního motoru při cvičení z předmětu Pohony a výkonová elektronika 2 [19].

6.1 Testovací příklad – připojení ASM s kotvou nakrátko bez mechanické zátěže přímo k napěťovému zdroji



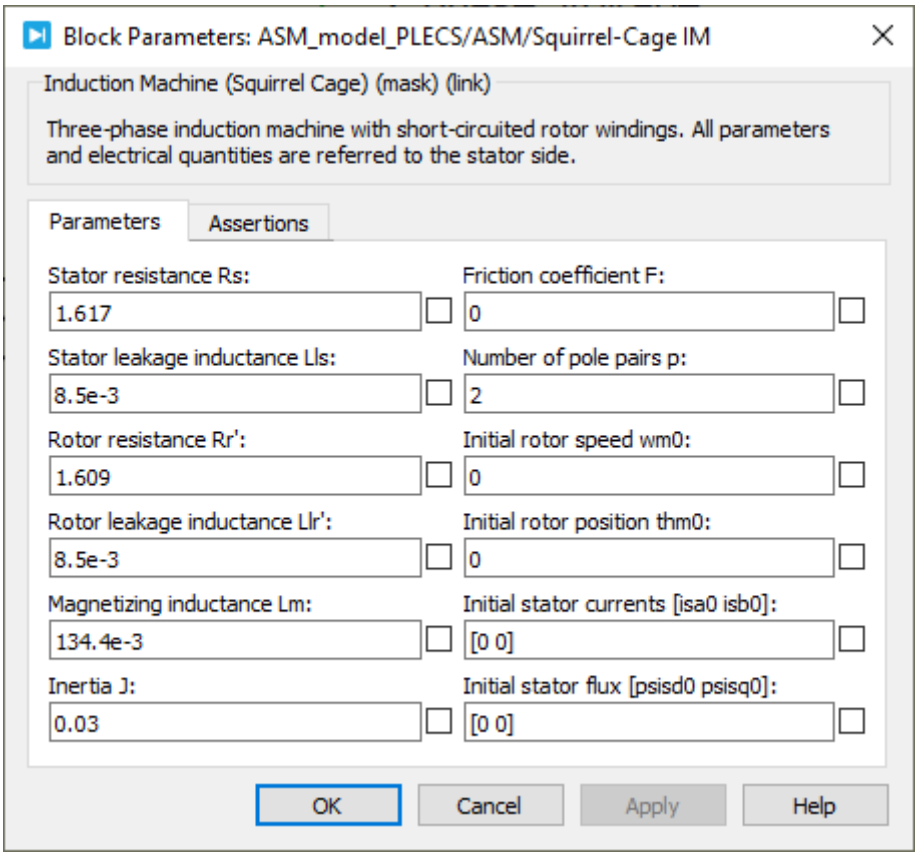
Obr. 6.1: ASM naprázdno přímo připojený ke zdroji napětí, realizace pomocí nástroje PLECS

První testovací příklad slouží k demonstraci funkčnosti matematického modelu, který je využíván balíčkem DynSyPy. Pro ověření byl totožný model vytvořen pomocí Matlab Simulink a také pomocí nástroje PLECS. Výsledky všech tří simulací budou v rámci tohoto testovacího příkladu porovnány.

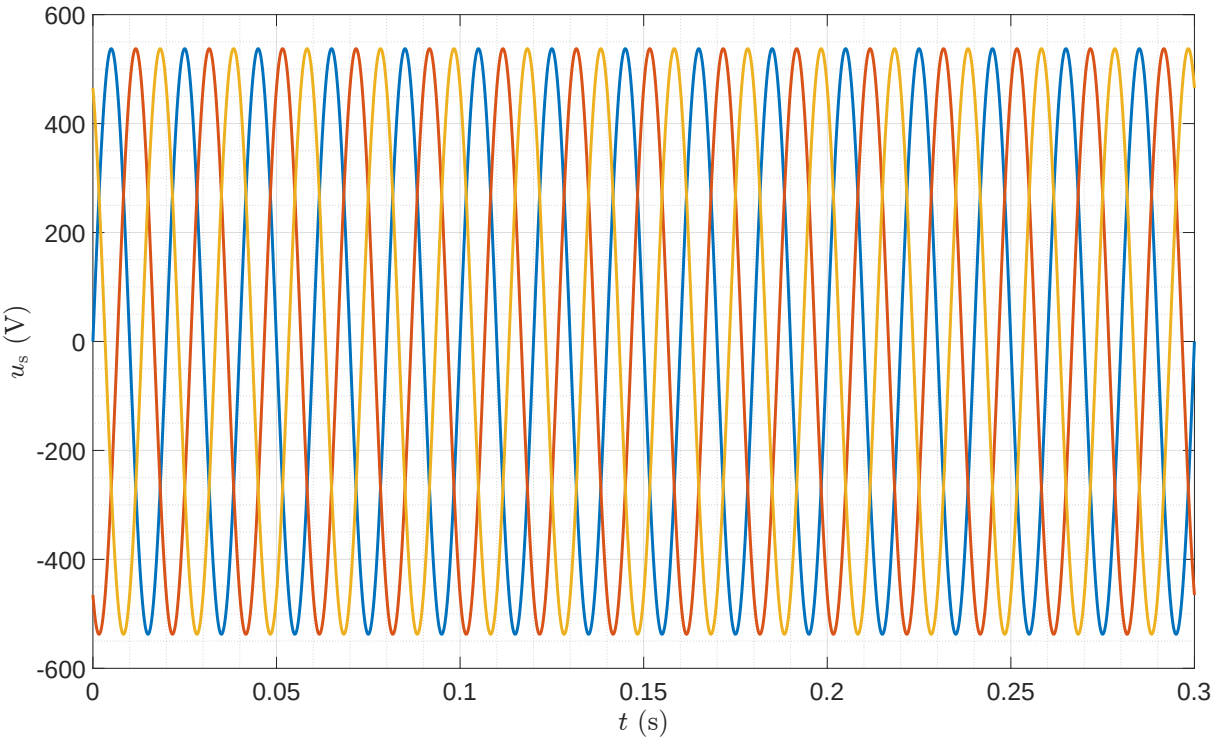
6.1.1 Ověření správnosti matematického modelu ASM

Jako referenční model, podle něhož bude určována správnost modelu použitého v této práci, byl použit model ASM v nástroji PLECS. Realizaci ukazují obrázky 6.1 a A.3, kde obr. 6.1 ukazuje realizaci bloku `ASM` uvedeného na obr. A.3. Nastavení parametrů ASM je ukázáno na obr. 6.2.

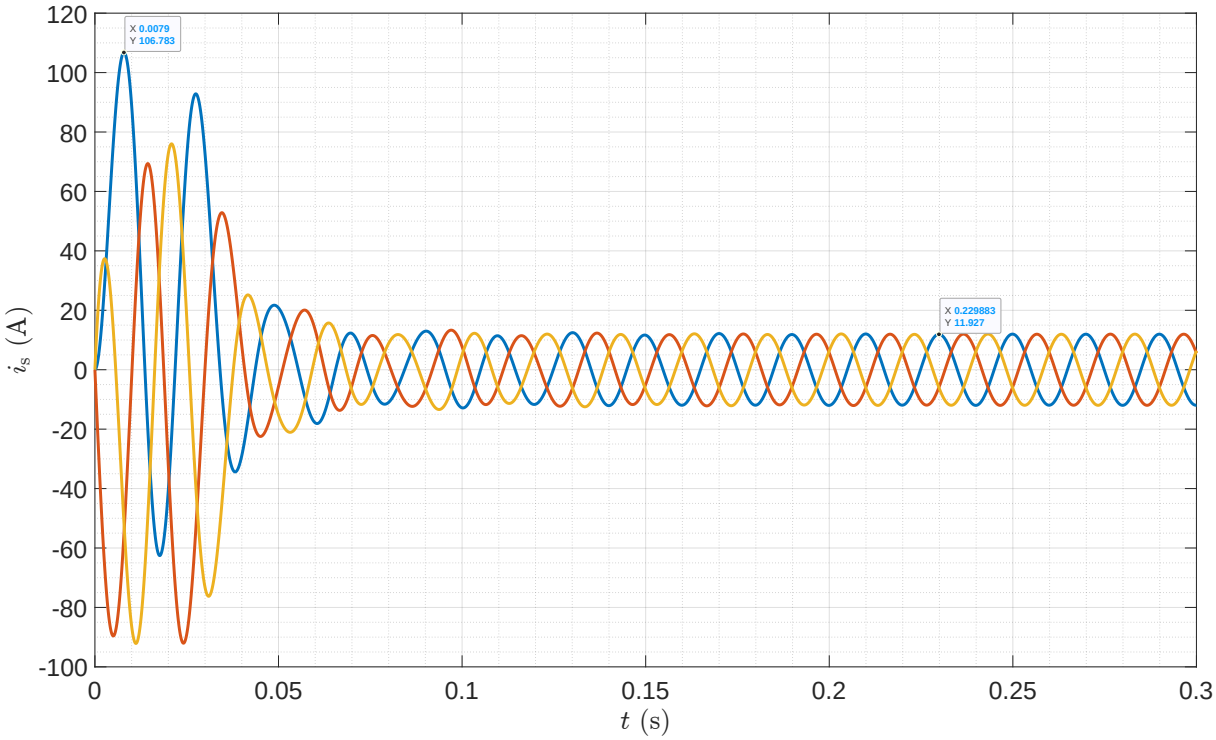
Výsledky simulace takto sestaveného modelu ukazují obrázky 6.3 až 6.5.



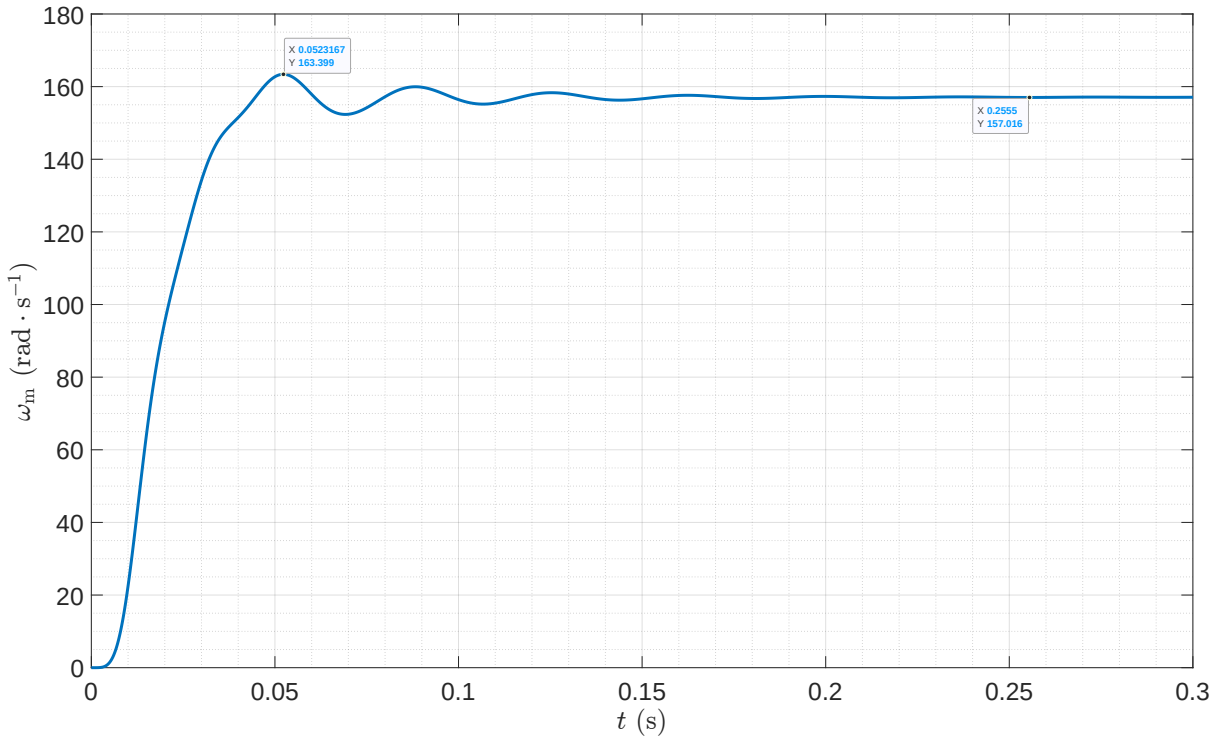
Obr. 6.2: Parametry ASM použité při simulaci pomocí PLECS



Obr. 6.3: Připojení ASM přímo ke zdroji 3f napětí, model pomocí nástroje PLECS, průběh u_{sa}, u_{sb} a u_{sc}



Obr. 6.4: Připojení ASM přímo ke zdroji 3f napětí, model pomocí nástroje PLECS, průběh i_{sa}, i_{sb} a i_{sc}

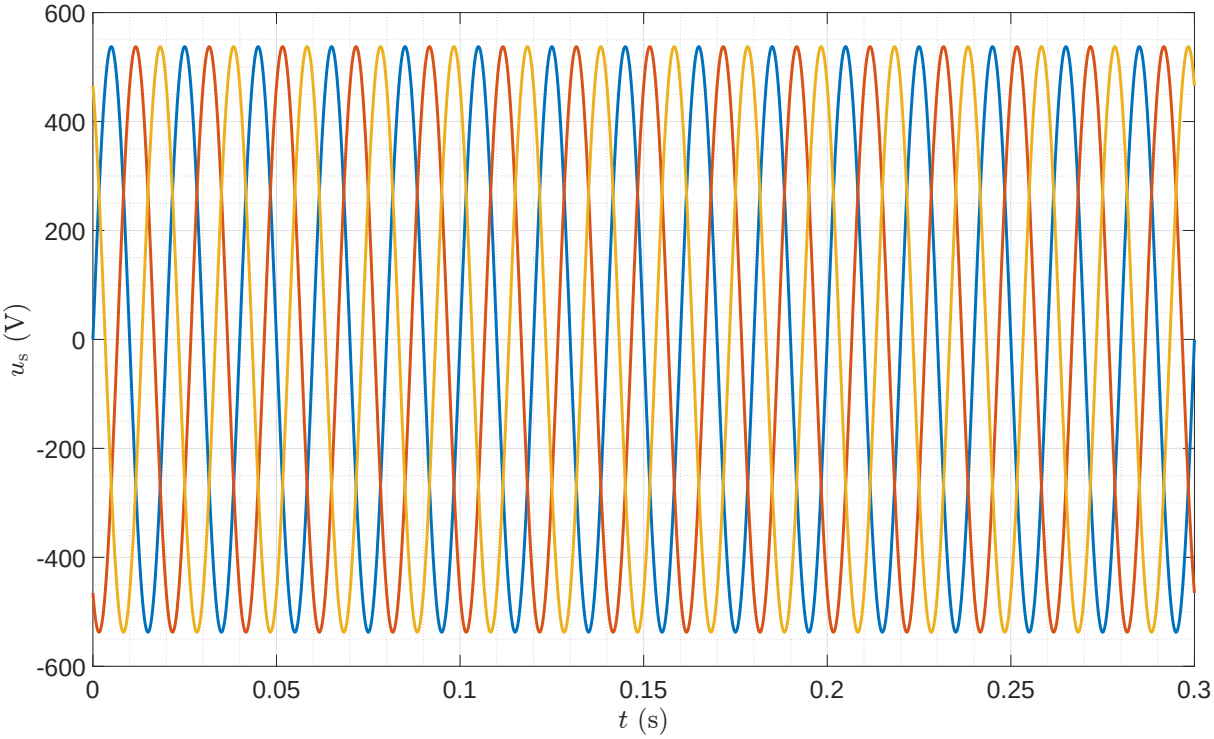


Obr. 6.5: Připojení ASM přímo ke zdroji 3f napětí, model pomocí nástroje PLECS, průběh ω_m

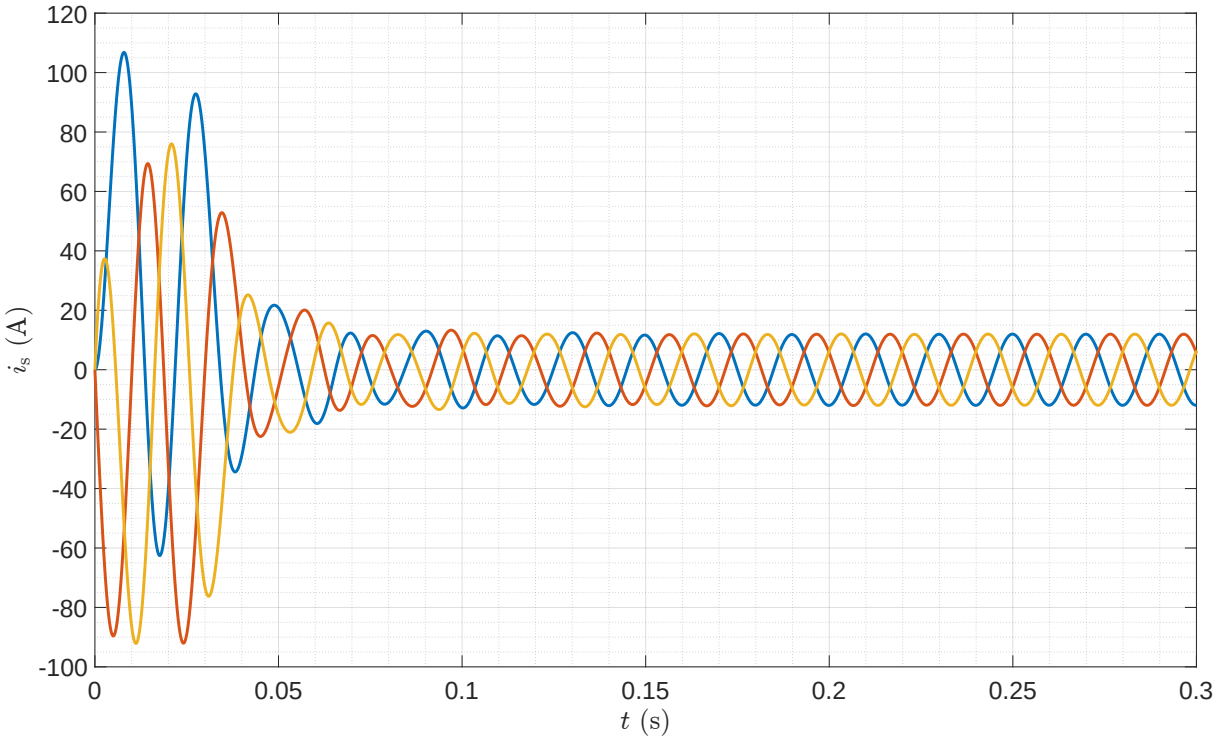
Na obr. 6.4 je ukázán průběh proudu statoru. Ihned po připojení ke zdroji napětí dochází dle očekávání ke strmému nárůstu velikosti proudu. Špičková hodnota proudu je

cca 107 A. Poté dochází k poklesu proudu a postupnému ustálení proudu na efektivní hodnotě cca 8, 5 A. Obrázek 6.5 ukazuje, že motor se po připojení začne rychle roztáčet, vlivem momentu setrvačnosti dochází k překmitu o špičkové hodnotě cca 163 rad $\cdot$ s^{-1} a poté se rychlost postupně ustálí na mírně podsynchronní hodnotě cca 157, 01 rad $\cdot$ s^{-1}.

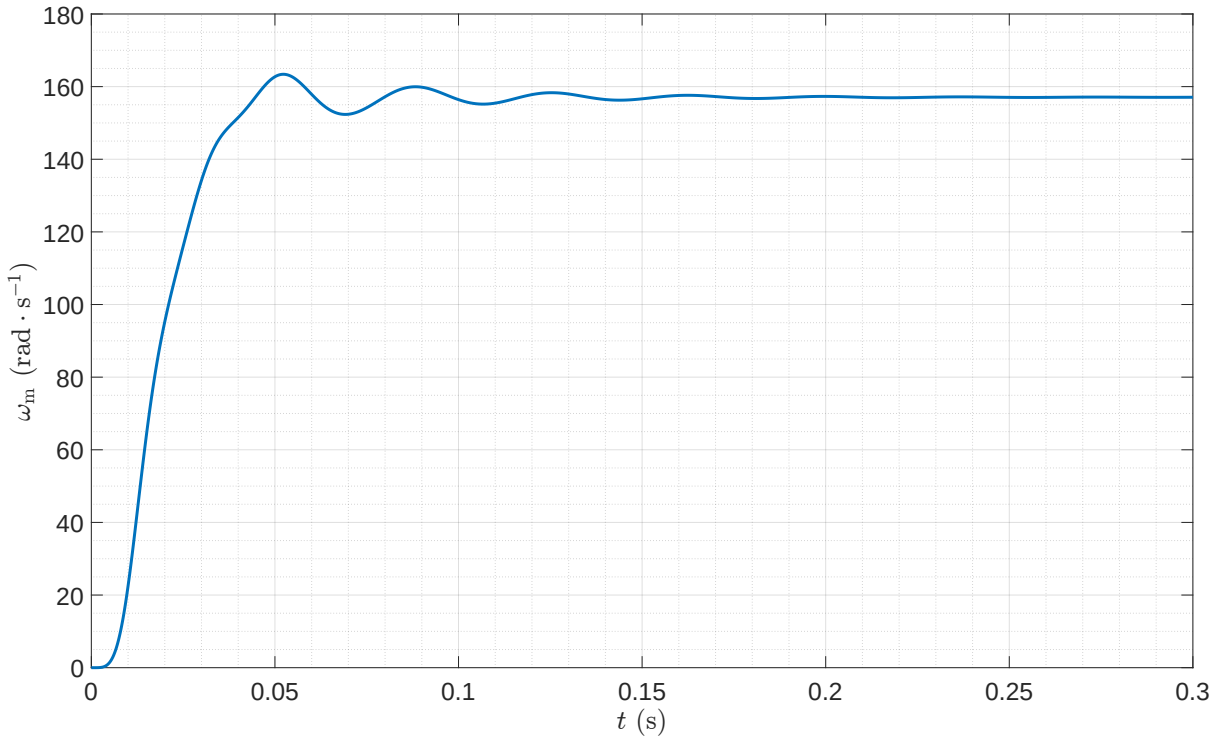
Podle výsledků simulace uvedených výše bylo posuzováno, zda je model ASM proveden správně. Pro ověření správnosti samotného matematického modelu (viz vztah (5.3)) byl tento model nejprve sestaven pomocí Matlab Simulink. Jeho provedení je znázorněno na obrázcích v příloze A.1. Grafické výstupy z tohoto modelu jsou uvedeny na obrázcích 6.6 až 6.8. Porovnáním těchto grafů s výsledky získanými pomocí nástroje PLECS lze konstatovat, že jsou shodné. Matematický model ve tvaru podle vztahu (5.3) je tedy správný.



Obr. 6.6: Připojení ASM přímo ke zdroji 3f napětí, model v Matlab Simulink, průběh u_{sa}, u_{sb} a u_{sc}



Obr. 6.7: Připojení ASM přímo ke zdroji 3f napětí, model v Matlab Simulink, průběh i_{sa}, i_{sb} a i_{sc}



Obr. 6.8: Připojení ASM přímo ke zdroji 3f napětí, model v Matlab Simulink, průběh ω_m

6.1.2 Simulace pomocí DynSyPy

Tato část je věnována ověření správnosti implementace matematického modelu ASM podle vztahu (5.3) v balíku DynSyPy. Aby bylo možné vytvářet instance tříd balíku DynSyPy, je nejprve nutné tento balík importovat do projektu. Spolu s DynSyPy byl importován také balík NumPy, který umožňuje práci s maticemi, a balík matplotlib, díky němuž bude možné výsledky simulace vynést do grafů.

```
import matplotlib.pyplot as plt
import numpy as np

from dynsypy import *
```

Dále je potřeba deklarovat několik proměnných, jako jsou počáteční podmínky, čas ukončení simulace a parametry třífázového zdroje.

```
t0 = 0
t_end = 0.3

x0 = np.array([[0.0],
               [0.0],
               [0.0],
               [0.0],
               [0.0]])

f_s_N = 50
U_s_N = 380

f = f_s_N
U = np.sqrt(2) * U_s_N
number_of_phases = 3
```

Posledním krokem před vytvořením objektů jednotlivých systémů je deklarace proměnných typu `dict` (slovník) obsahujících parametry ASM a třífázového zdroje.

```
motor_params = {
    "R_s": 1.617,
    "R_r": 1.609,
    "L_s_sigma": 8.5e-3,
    "L_r_sigma": 8.5e-3,
```

```
    "L_h": 134.4e-3,
    "p_p": 2,
    "U_s_N": U_s_N,  # 3x380
    "f_s_N": f_s_N,
    "J": 0.03
}

source_params = {
    "amplitude": U,
    "frequency": f,
    "number_of_phases": number_of_phases,
    "phase": 0
}
```

Nyní jsou všechny potřebné proměnné deklarovány a je možné přistoupit k vytváření systémů. Kromě třífázového zdroje a asynchronního motoru je potřeba také vytvořit tři instance třídy UnitStep, které budou sloužit pro generování požadavku na amplitudu a frekvenci třífázového zdroje a také nastaví zátěžný moment motoru na hodnotu 0 (motor naprázdno). Nakonec je nutné vytvořit instanci třídy Pool, která provede simulaci těchto systémů.

```
amplitude = UnitStep(final_value=U, initial_value=0.0, step_time=0)
frequency = UnitStep(final_value=f, initial_value=0.0, step_time=0)
load_torque = UnitStep(final_value=0, initial_value=0.0, step_time=0)

source_3_f = ControlledNPhaseSine(source_params)

motor = SquirrelCageIM(motor_params,
                       x0=x0, number_of_inputs=4, dt0=5e-5)

pool = Pool(1e-2, t_end, t0, True)
```

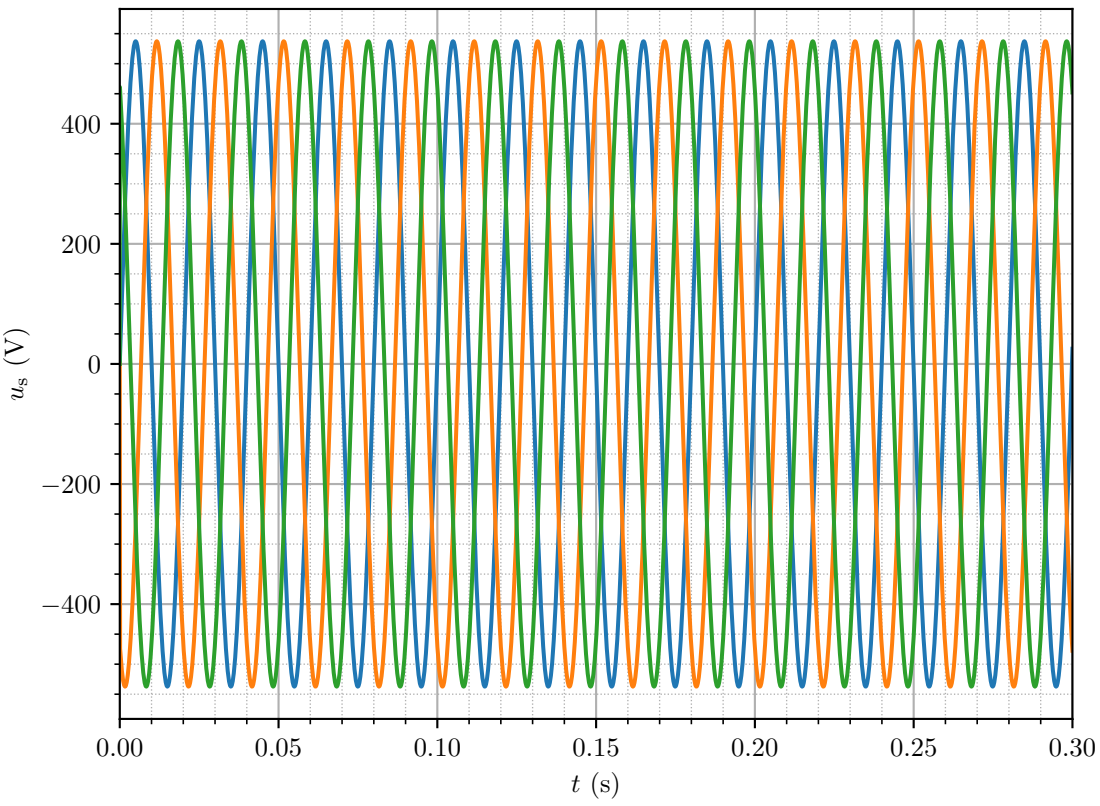
Po vytvoření systémů je potřeba tyto systémy mezi sebou propojit. K tomuto účelu je použita metoda `connect()`.

```
source_3_f.connect(amplitude.output, 0)
source_3_f.connect(frequency.output, 1)

motor.connect(source_3_f.output, 0)
motor.connect(load_torque.output, 1)
```

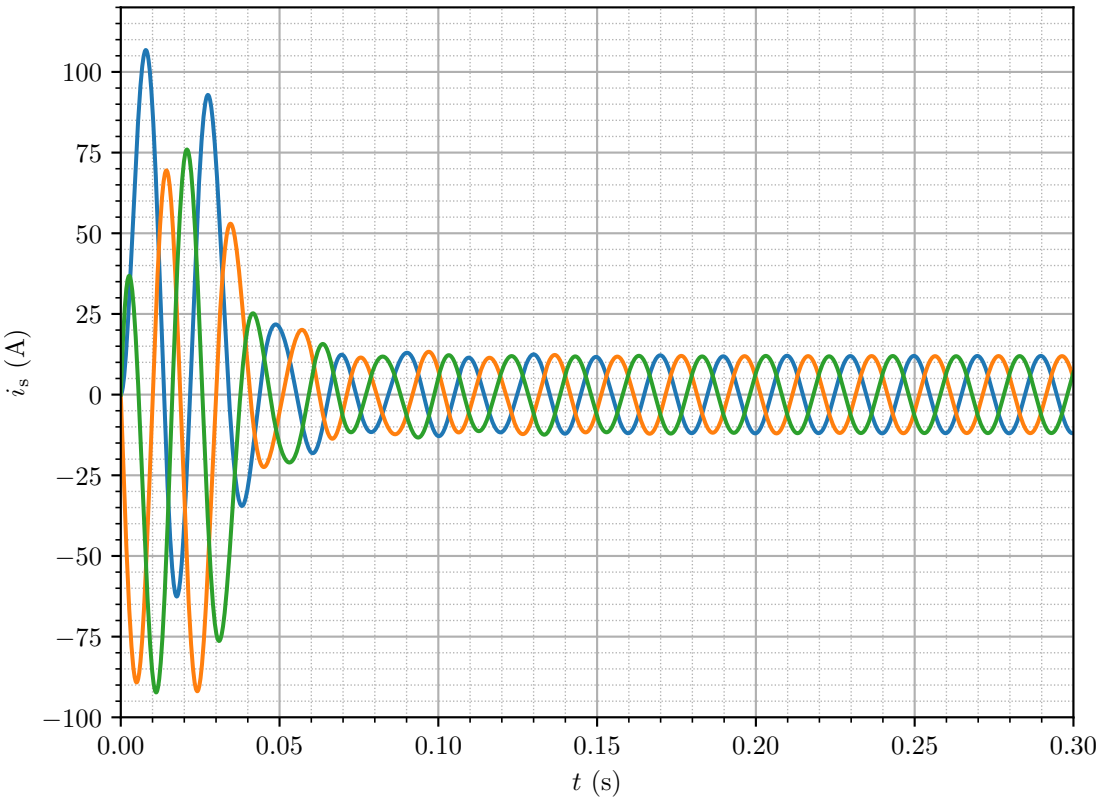
Nyní jsou všechny systémy vytvořeny a propojeny. Posledním krokem je předat systémy instanci třídy Pool, aby mohly být odsimulovány. To je provedeno pomocí metody `add()`. Po předání všech systémů je možné spustit simulaci zavoláním metody `simulate()`.

```
pool.add(amplitude)
pool.add(frequency)

pool.add(load_torque)

pool.add(source_3_f)

pool.add(motor)

pool.simulate()
```

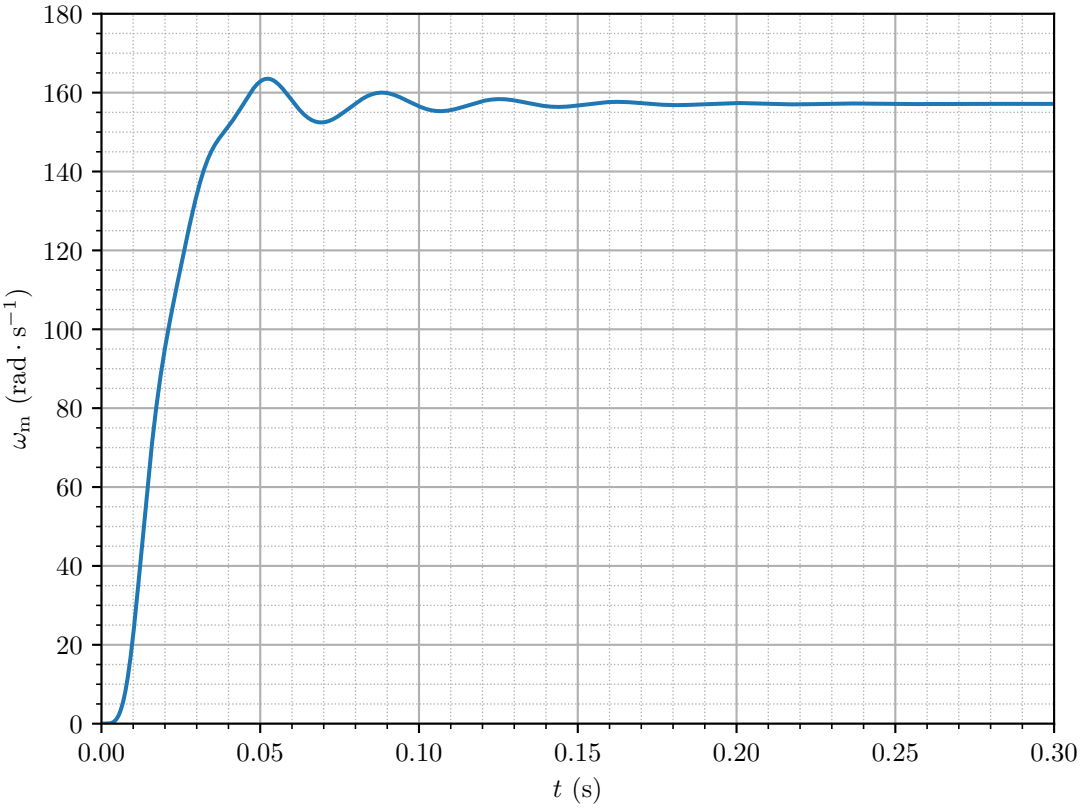
Po provedení simulace byly do grafů vyneseny stejné hodnoty jako u simulací pomocí nástrojů PLECS a Matlab Simulink a jsou uvedeny na obrázcích 6.9 až 6.11. Porovnáním těchto grafů s výsledky získanými pomocí PLECS a Matlab Simulink bylo zjištěno, že průběhy se shodují. Lze tedy konstatovat, že implementace matematického modelu asynchronního stroje s klecovou kotvou v balíčku DynSyPy je provedena správně.



Obr. 6.9: Připojení ASM přímo ke zdroji 3f napětí, model v DynSyPy, průběh u_{sa}, u_{sb} a u_{sc}



Obr. 6.10: Připojení ASM přímo ke zdroji 3f napětí, model v DynSyPy, průběh i_{sa}, i_{sb} a i_{sc}



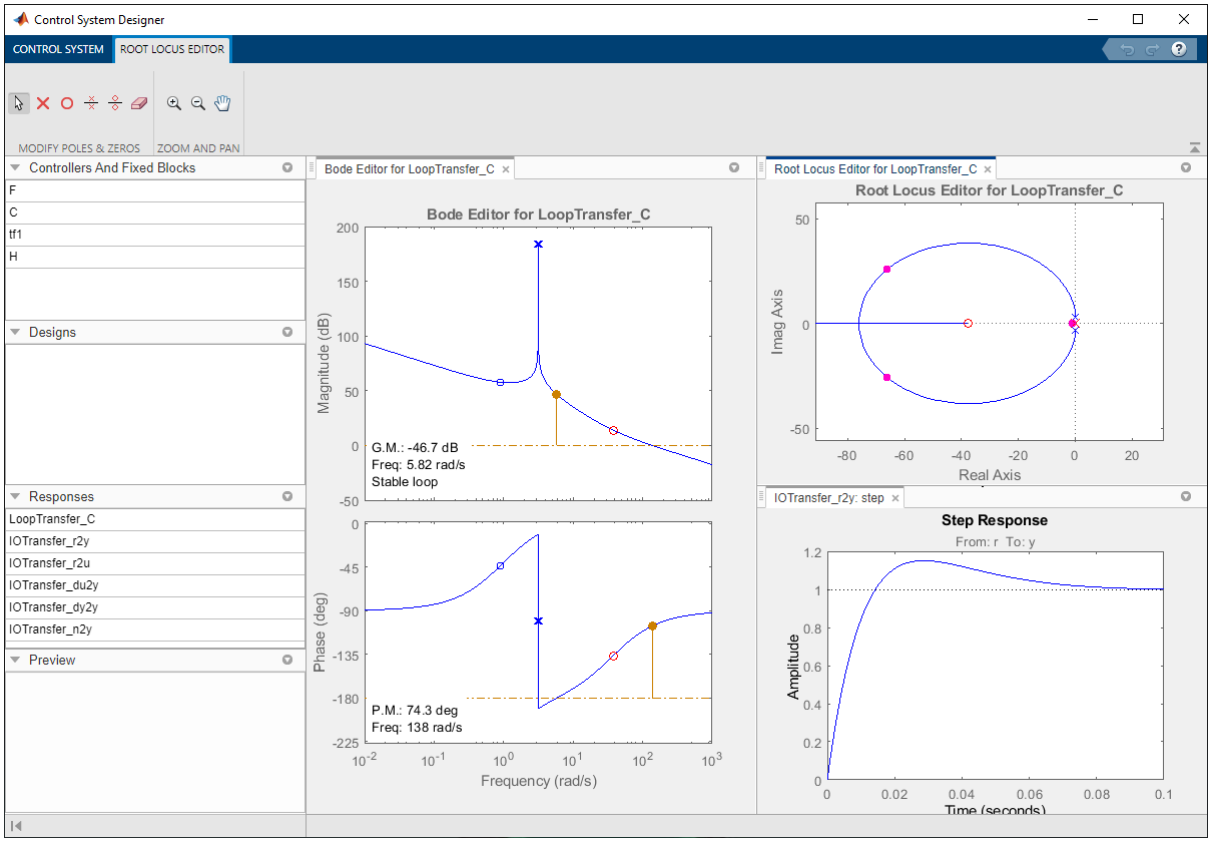
Obr. 6.11: Připojení ASM přímo ke zdroji 3f napětí, model v DynSyPy, průběh ω_m

6.2 Testovací příklad – skalární řízení ASM

Cílem druhého testovacího příkladu je ověřit, zda je možné pomocí balíčku DynSyPy realizovat složitější úlohy. Skalární řízení je úloha komplikovanější nejen z důvodu většího množství najednou simulovaných systémů, ale hlavně z důvodu simulace systémů spojených do uzavřené smyčky. Protože bylo ověřeno, že výstupy simulací modelu z PLECS i z Matlab Simulink se shodují, není tedy potřeba uvádět výsledky z obou nástrojů. Výstupy ze simulace pomocí DynSyPy budou tedy dále porovnávány pouze s modelem v PLECS.

6.2.1 Návrh PI regulátoru otáček ASM

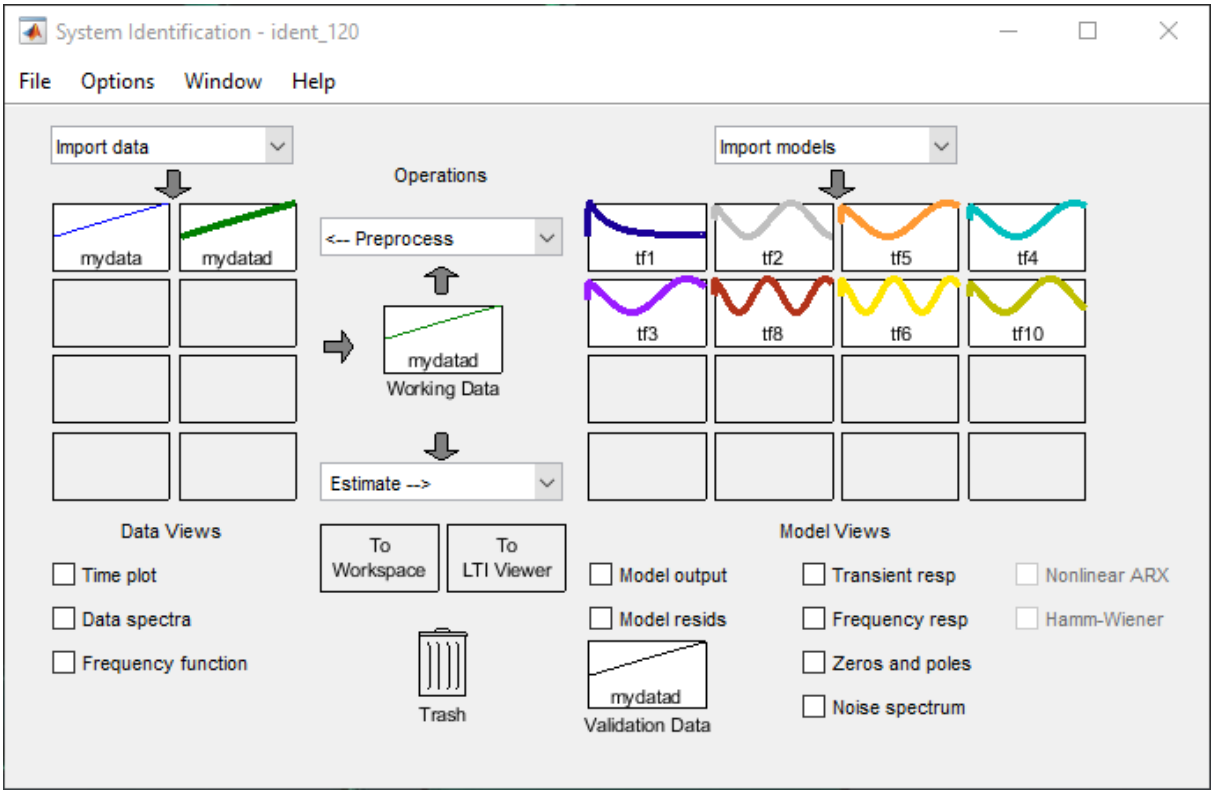
Parametry PI regulátoru byly nejprve zjištěny metodou ručního ladění pomocí Matlab Simulink. Tímto postupem byly stanoveny hodnoty $K = 2/3$ a $T_i = 0.05$. Pro tyto parametry regulátoru bylo možné systém uřídit v celém rozsahu otáček. Byly však rozdíly v kvalitě regulace na nízkých a na vysokých otáčkách, kdy na nižších otáčkách byly přechodné děje více kmitavé. Z tohoto důvodu bylo přistoupeno k ověření správnosti navrženého regulátoru pomocí nástroje Control System Designer dostupného v rámci programu Matlab.



Obr. 6.12: Ukázka návrhu regulátoru pomocí nástroje Control System Designer

Control System Designer však umí pracovat pouze s lineárními systémy. Bylo tedy

potřeba nelineární model ASM linearizovat. V kapitole 2.2 je popsán postup, jak linearizovat nelineární systém v okolí pracovního bodu, kde systémové matice **A**, **B**, **C** a **D** jsou určovány pomocí Jacobiových matic. Tento postup platí v okamžiku, kdy dojde k ustálení systému a kdy vstup systému je konstantní. Jelikož v tomto případě je ASM popsán modelem ve stojícím souřadném systému α, β, pro jakýkoliv ustálený stav při nenulových otáčkách rotoru platí, že vstupy u_α a u_β jsou harmonické funkce času a není možné splnit podmínku konstantního vstupu. První možností, jak toto vyřešit, bylo přepočítat model ASM do rotujícího souřadného systému d, q, čímž by bylo docíleno konstantního vstupu. Další možností bylo počítat linearizovaný model v systému α, β s tím, že nebudou platit zjednodušení uvedená v kapitole 2.2. V obou těchto případech by bylo samozřejmě potřeba dále dopočítat celkový přenos od výstupu regulátoru otáček po výstup z ASM („měření“ rychlosti otáčení rotoru). Výsledkem tohoto procesu by byl přesný linearizovaný model systému v okolí požadovaného pracovního bodu.



Obr. 6.13: Ukázka identifikace systému pomocí nástroje System Identification Toolbox

Vzhledem k náročnosti tohoto procesu bylo přistoupeno k dalšímu v praxi běžně používanému způsobu získávaní linearizovaného modelu systému: experimentální identifikace. Pro tento způsob je možné využít nástroje System Identification Toolbox, který je také dostupný v programu Matlab. Ten ke své činnosti potřebuje data vstupující do identifikovaného systému a k nim odpovídající data vystupující ze systému. Díky nim je schopen určit linearizovaný model systému.

Provedení experimentální identifikace spočívá v ustálení systému v požadovaném pracovním bodě a v následném vybuzení dynamického chování v tomto pracovním bodě.

Realizace tohoto problému byla provedena pomocí Matlab Simulink (viz příloha B.1). K ustálení systému na požadovaných otáčkách byl použit ručně navržený regulátor. Po ustálení systému byl výstup regulátoru „zmrazen“ a na tuto hodnotu byl poté superponován signál získaný pomocí bloku Random Number. Tento experimentální signál není ideální, vhodnějším signálem by byla série skoků o náhodné amplitudě s náhodným časovým rozestupem. Byl však použit z důvodu relativní jednoduchosti realizace.

Tímto způsobem získaná data byla vložena do nástroje System Identification Toolbox a byly z nich určeny linearizované modely (přenosové funkce) pro ustálené stavy při $\omega_\mathrm{m} = 30, 60, 90, 120$ a 150 rad $\cdot$ s^{-1}. Pro takto získané přenosové funkce byly navrženy regulátory pomocí nástroje Control System Designer. V tabulce 6.1 jsou uvedeny parametry regulátorů získané ručním laděním a pomocí experimentální identifikace.

Tab. 6.1: Parametry navržených regulátorů

	K	T_i
Ruční ladění	$0,\overline{6}$	0,05
Experimentální identifikace, $\omega_\mathrm{m} = 30$ rad $\cdot$ s^{-1}	1,9332	0,0288
Experimentální identifikace, $\omega_\mathrm{m} = 60$ rad $\cdot$ s^{-1}	1,2167	0,0265
Experimentální identifikace, $\omega_\mathrm{m} = 90$ rad $\cdot$ s^{-1}	0,2006	0,0926
Experimentální identifikace, $\omega_\mathrm{m} = 120$ rad $\cdot$ s^{-1}	0,1917	0,0956
Experimentální identifikace, $\omega_\mathrm{m} = 150$ rad $\cdot$ s^{-1}	13,1054	0,0473

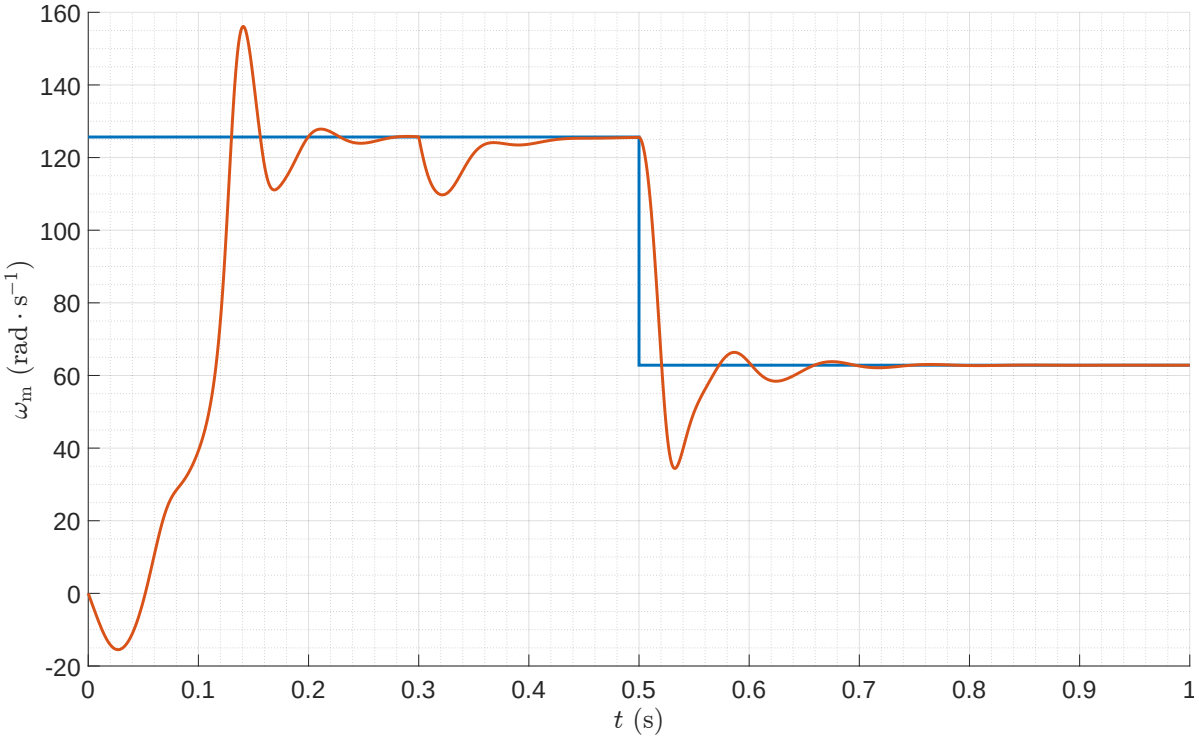
Z navržených regulátorů v celém pracovním rozsahu funguje regulátor získaný pomocí ručního ladění a dále regulátory navržené pro pracovní body $\omega_\mathrm{m} = 30$ rad $\cdot$ s^{-1} a $\omega_\mathrm{m} = 60$ rad $\cdot$ s^{-1}. Ostatní regulátory nefungovaly spolehlivě v celém rozsahu. Regulátor pro pracovní bod $\omega_\mathrm{m} = 150$ rad $\cdot$ s^{-1} paradoxně nevyhověl při regulaci vyšších otáček, kde docházelo k netlumeným kmitům. To je pravděpodobně způsobeno tím, že při identifikaci byla deaktivována saturace požadavku na amplitudu statorového napětí. Ostatní regulátory neuregulovaly systém v okolí nulových otáček. Pro realizaci testovacího příkladu byl nakonec ponechán původní regulátor, jelikož doba ustálení výstupu byla při jeho použití nejkratší.

6.2.2 Simulace pomocí Matlab Simulink

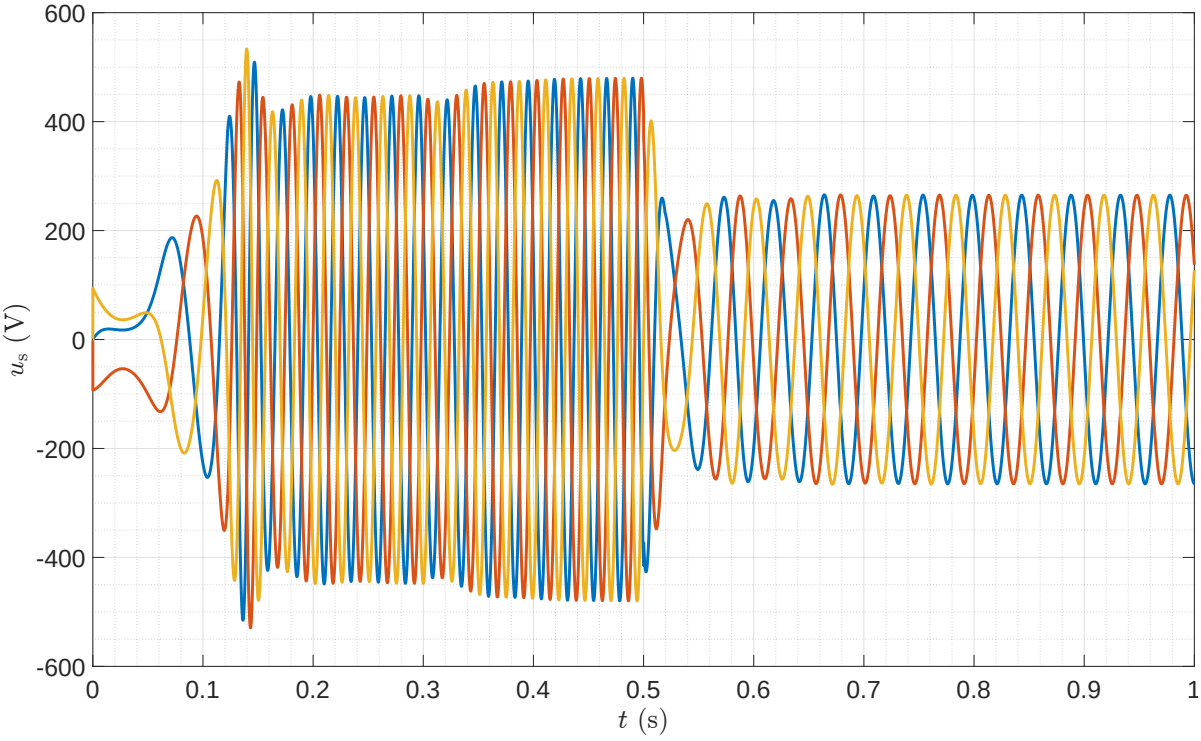
Stejně jako u prvního testovacího příkladu byla simulace provedena nejprve pomocí programu Matlab Simulink s využitím modelu ASM z nástroje PLECS.

Nejprve je vhodné zmínit, jakým způsobem byl proveden experiment, kterým byl tento systém testován. Na začátku byla nastavena velikost zátěžného momentu na hodnotu 25 Nm a požadovaná úhlová rychlost otáčení rotoru na hodnotu $2 \cdot \pi \cdot 20$ rad $\cdot$ s^{-1}. Saturace výstupu regulátoru, který představuje rotorovou frekvenci, byla nastavena na hodnotu 5 Hz tedy asi $31,4$ rad $\cdot$ s^{-1}, aby nemohlo dojít k překročení tzv. kritické rotorové frekvence

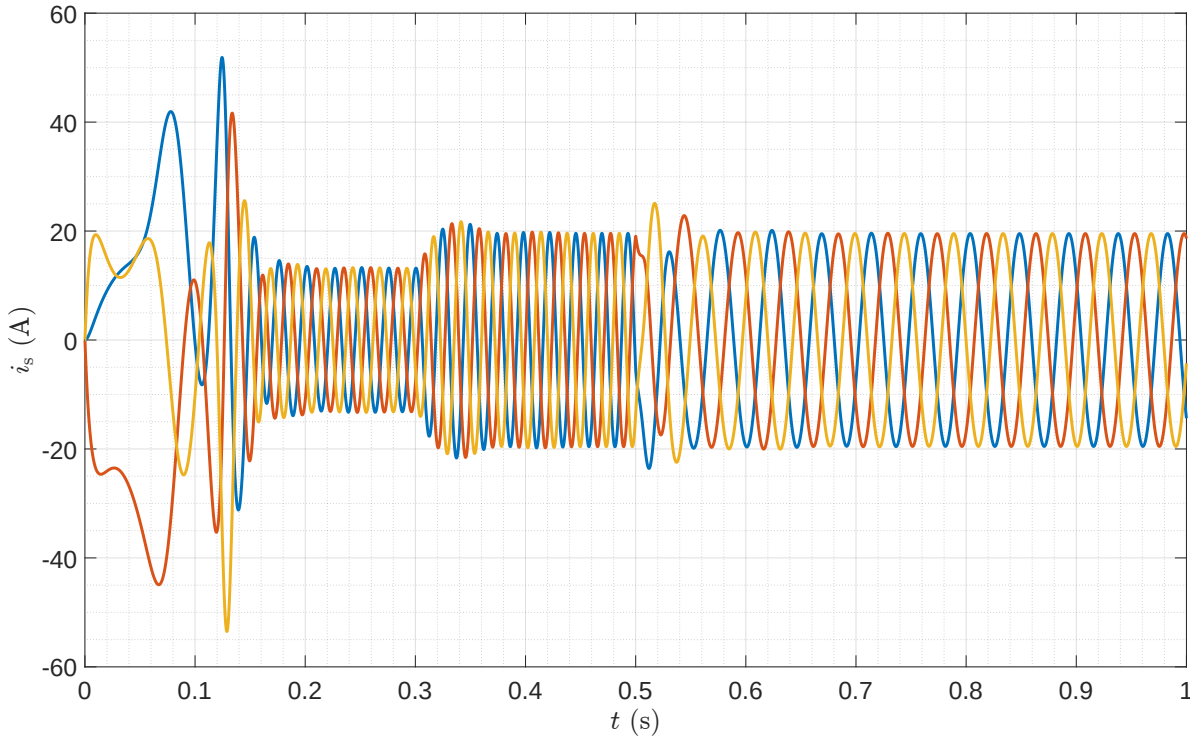
f_{rk}. Saturace požadované amplitudy statorového napětí byla nastavena na hodnotu $380 \cdot \sqrt{2}$ V. V čase $0,3$ s dojde ke skokové změně zátěžného momentu na hodnotu 70 Nm. Nakonec v čase $0,5$ s dochází ke skokové změně požadavku úhlové rychlosti otáčení rotoru na hodnotu $2 \cdot \pi \cdot 10$ rad $\cdot$ s^{-1}. Celý experiment je poté ukončen v čase 1 s.



Obr. 6.14: Skalární řízení, model pomocí nástroje PLECS, průběh ω_{m}



Obr. 6.15: Skalární řízení, model pomocí nástroje PLECS, průběh u_{sa}, u_{sb} a u_{sc}



Obr. 6.16: Skalární řízení, model pomocí nástroje PLECS, průběh i_{sa}, i_{sb} a i_{sc}

Z grafu na obr. 6.14 je patrné, že použitý regulátor systém uřídí bez výraznějších potíží. Reakce na změnu požadovaných otáček ovšem vyvolá kmitavý děj. Regulátor lépe reaguje na změnu zatěžovacího momentu. Kdyby se jednalo o reálnou aplikaci pohonu, bylo by vhodné zvážit použití složitějšího řídicího obvodu. V tomto případě by se nabízelo použití 2DoF regulátoru (regulátor s dvěma stupni volnosti), u něhož je možné nastavit jinou reakci na změnu referenční hodnoty (v tomto případě požadavek na rychlost rotoru) a jinou reakci na poruchy (zde změna zátěžného momentu). Případně vzhledem k nelineárnímu charakteru asynchronního stroje by bylo možné pro zkvalitnění regulace navrhnout např. dva regulátory, kde první by byl navržen pro regulaci otáček blízkých nule a druhý by sloužil k regulaci vysokých otáček.

Na obr. 6.15 jsou vykresleny hodnoty statorových napětí. Je patrné, že amplituda napětí roste se zvyšující se frekvencí a naopak. Princip skalárního řízení je zde tedy funkční. Jelikož výstup regulátoru byl omezen na max. hodnotu cca $31,4\ \mathrm{rad \cdot s^{-1}}$, nedojde např. na začátku simulace ke skokovému nárůstu frekvence (a tím i amplitudy) statorového napětí, ale tento nárůst je pozvolný. Saturace výstupu regulátoru je zde tedy také plně funkční.

Průběhy statorových proudů na obr. 6.16 vypadají také dle očekávání. Při rozběhu motoru dochází k nárůstu proudu na špičkovou hodnotu až cca 53 A, následně dojde k ustálení amplitudy proudu na hodnotě přibližně 13 A. Po změně zatěžovacího momentu dochází ke zvýšení amplitudy proudu na hodnotu cca 20 A, která zůstává i po zpomalení rotoru na hodnotu úhlové rychlosti $2 \cdot \pi \cdot 10\ \mathrm{rad \cdot s^{-1}}$.

6.2.3 Simulace pomocí DynSyPy

Nyní je potřeba experiment o stejné konfiguraci provést také za využití vytvořeného balíčku DynSyPy. První část kódu je možné ponechat stejnou, jako je provedena v části 6.1.2, tedy import balíků, deklarace základních proměnných, počátečních podmínek a slovníků s parametry. Ke stávajícím dvěma slovníkům deklarujícím parametry asynchronního stroje a třífázového zdroje napětí je potřeba ještě přidat deklaraci třetího slovníku pro nastavení parametrů PI regulátoru.

```
PI_controller_params = {
    "K": 2 / 3,
    "T_i": 0.05,
    "saturation_value": 2 * np.pi * 5
}
```

Deklarace proměnných je tímto dokončena a je možné přistoupit k vytváření objektů potřebných systémů. Opět bude potřeba vytvořit dvě instance třídy UnitStep, které budou sloužit pro generování hodnot požadované úhlové rychlosti otáčení rotoru a zátěžného momentu. Dále zde budou vytvořeny instance tříd PIController a ASMScalarControl pro regulaci a implementaci principu skalárního řízení. Nyní zbývá vytvořit pouze třífázový zdroj a asynchronní motor a pro provedení simulace těchto systémů ještě musí být vytvořena instance třídy Pool.

```
required_speed = UnitStep(step_time=0.5, initial_value=2 * np.pi * 20,
                          final_value=2 * np.pi * 10)
load_torque = UnitStep(step_time=0.3, initial_value=25, final_value=70)

controller = PIController(PI_controller_params)
scalar_control = ASMScalarControl(motor_params)

source_3_f = ControlledNPhaseSine(source_params)

motor = AsynchronousMachine(motor_params,
                            x0=x0, number_of_inputs=4, dt0=5e-5)

pool = Pool(1.3e-4, t_end, t0, False)
```

Systémy je po vytvoření potřeba mezi sebou správně propojit. Opět je to provedeno pomocí metody `connect()`.

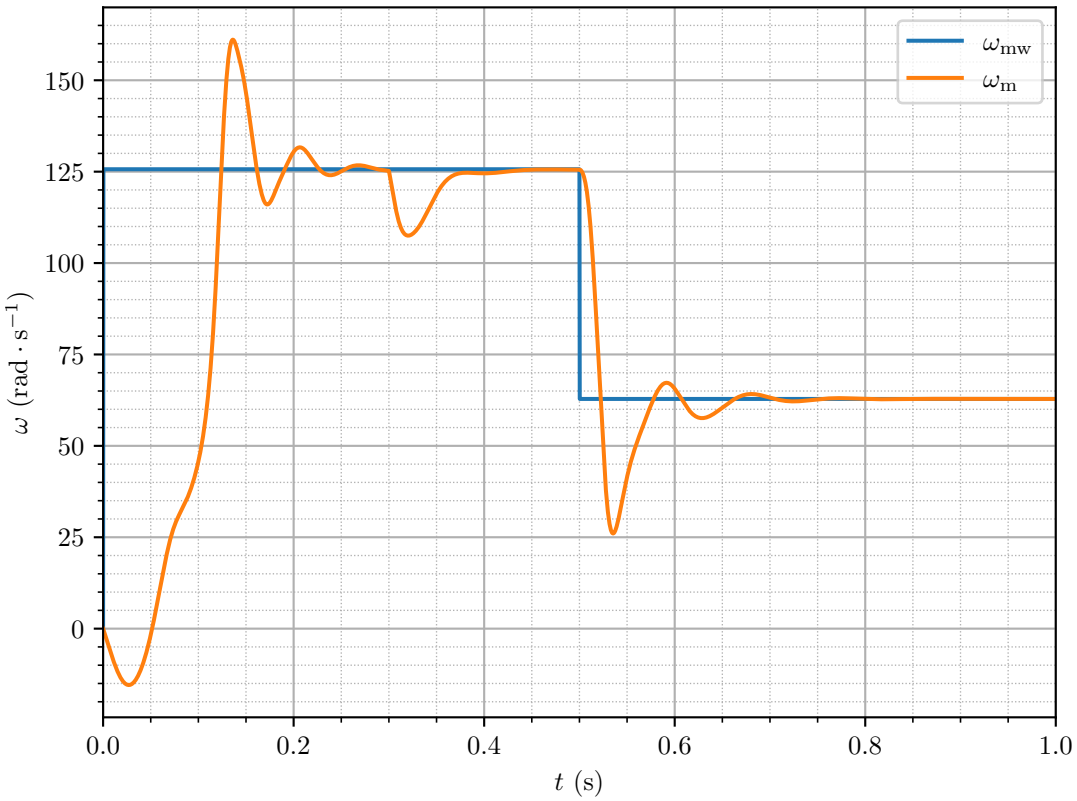
```
controller.connect(required_speed.output, 0)
controller.connect(motor.output, 1, [3])

scalar_control.connect(controller.output, 0)
scalar_control.connect(motor.output, 1, [3])

source_3_f.connect(scalar_control.output, 0)

motor.connect(source_3_f.output, 0)
motor.connect(load_torque.output, 1)
```

Po správném propojení systémů mezi sebou jsou všechny systémy opět pomocí metody `add()` předány instanci třídy Pool a zavoláním metody `simulate()` je spuštěna simulace.

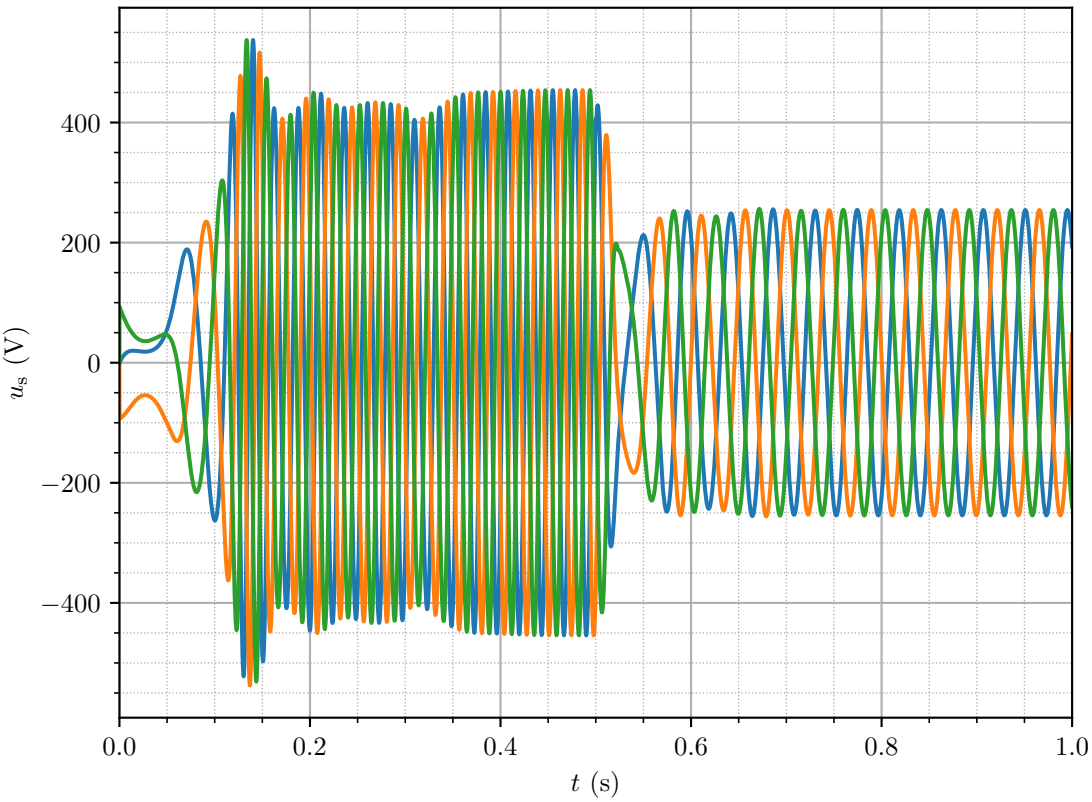
```
pool.add(required_speed)
pool.add(load_torque)

pool.add(controller)
pool.add(scalar_control)

pool.add(source_3_f)

pool.add(motor)

pool.simulate()
```

Po ukončení simulace byly za využití balíku matplotlib.pyplot vykresleny průběhy stejných veličin jako při simulaci pomocí Matlab Simulink v části 6.2.2. Tyto grafické výstupy jsou uvedeny na obrázcích 6.17 až 6.19.

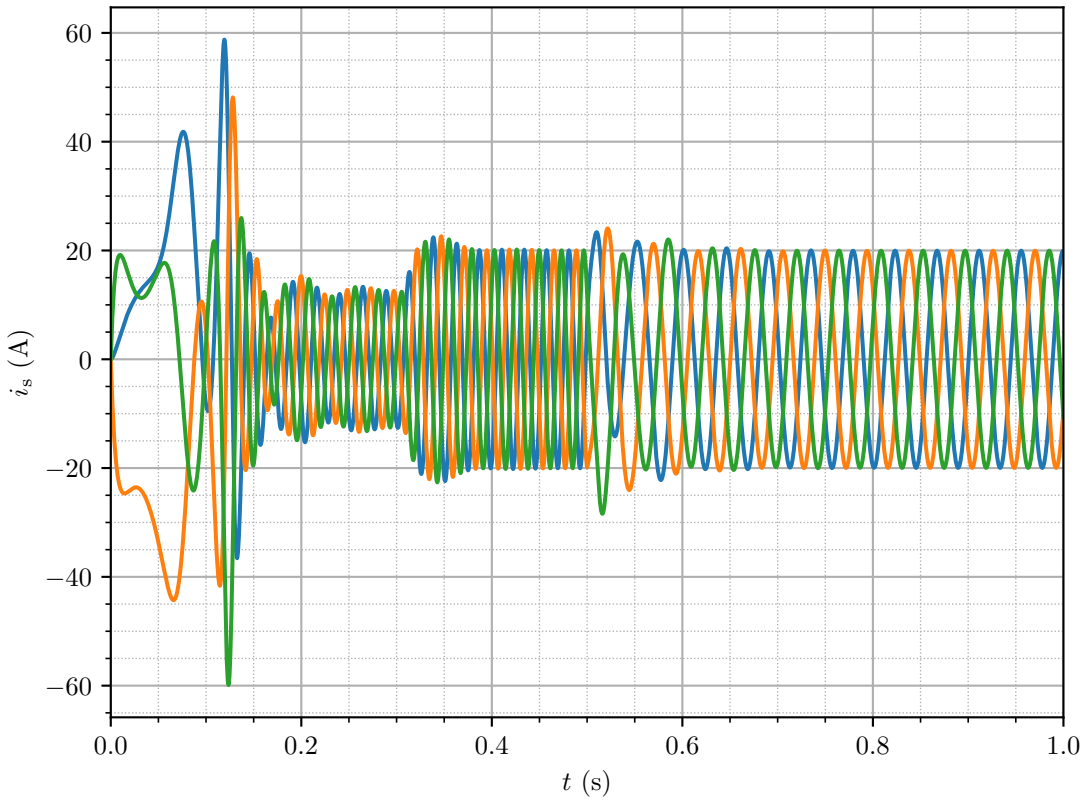
Při porovnání těchto grafických výstupů s výstupy z Matlab Simulink je nutno konstatovat, že se mírně liší. Rozdíly jsou patrné při přechodných dějích, v ustálených stavech jsou průběhy téměř shodné. Z obrázků 6.18 a 6.19 je patrné, že princip skalárního řízení je plně funkční, saturace výstupu regulátoru také zpomaluje rychlost nárůstu frekvence a amplitudy statorového napětí. Je tedy možné prohlásit, že struktura regulátoru i skalárního řízení je v balíčku DynSyPy je implementována správně. Problém je zřejmě v použité metodě simulace systémů, která je sice schopna simulovat systémy s různou délkou simulačního kroku, případně s adaptivním krokem, ale pouze řazených za sebou. Při uzavření zpětnovazební smyčky tento systém selže.



Obr. 6.17: Skalární řízení, model v DynSyPy, průběh ω_m



Obr. 6.18: Skalární řízení, model v DynSyPy, průběh u_{sa}, u_{sb} a u_{sc}



Obr. 6.19: Skalární řízení, model v DynSyPy, průběh i_{sa}, i_{sb} a i_{sc}

Závěr

Tato diplomová práce se zabývala dynamickým modelováním elektrických strojů za využití vysokoúrovňového programovacího jazyka Python. Cílem této práce bylo vytvořit balíček, ve kterém bude možné vytvářet modely elektrických strojů. Tento nástroj byl vytvořen při zpracování diplomové práce na základě dříve vyvinutého balíčku DynSyPy.

Původní balík DynSyPy byl vybaven solvery pro numerické řešení diferenciálních rovnic s konstantním (RK4) i s adaptivním krokem (RKF) a také třídami pro modelování a simulaci LTI systémů a jednoduchých neřízených zdrojů (např. jednotkový skok). Před implementací tříd umožňujících modelovat elektrické stroje bylo potřeba vyřešit několik nedostatků původního balíčku, jež se týkaly převážně problematiky spojování systémů. Jejich vyřešením byl odstraněn problém s připojením třífázového zdroje na vstup asynchronního stroje.

Do DynSyPy byly v rámci diplomové práce přidány třídy nezbytné pro modelování asynchronního stroje, pro návrh jeho řízení, ale také abstraktní třídy pro docílení lepší modularity balíku a pro usnadnění dalšího vývoje a rozšiřování. Třídy Machine a SquirrelCageIM definují základní atributy a metody potřebné pro práci s modelem stroje. Model použitý ve třídě SquirrelCageIM byl v rámci této práce také odvozen. Pro implementaci modelů strojů v maticovém tvaru byla vytvořena třída Matrix, která řeší problematiku v čase se měnících matic. Dále byla vytvořena třída ControlledNPhaseSine, která umožňuje simulovat harmonický zdroj o libovolném počtu fází. Jedná se vlastně o velmi zjednodušený model měniče. Zde bylo potřeba vyřešit, aby při skokové změně požadavku na frekvenci nedocházelo ve výstupním signálu ke skokům ve fázi. Aby bylo možné balíček otestovat na příkladu skalárního řízení, byly do něho implementovány třídy IMScalarControl a PIController. Pomocí balíku DynSyPy je tedy možné simulovat skalární řízení s regulací otáček.

Činnost balíčku DynSyPy byla ověřována na dvou testovacích příkladech. Na simulaci připojení asynchronního stroje s klecovou kotvou přímo ke zdroji napětí bylo demonstrováno, že model stroje implementovaný v balíčku funguje správně. Výstupy ze simulace byly porovnány s výstupy modelu sestaveného pomocí Matlab Simulink a také s výstupy modelu nástroje PLECS. Ve všech třech případech bylo dosaženo totožných výsledků. Tvary průběhů při přechodných dějích, špičkové hodnoty i hodnoty v ustáleném stavu jsou shodné, jak ukazují grafy na obrázcích 6.3 až 6.11. Doba simulace v DynSyPy však trvala výrazně déle, což je způsobeno tím, že kód pro práci s daty v průběhu simulace

zatím není plně optimalizován.

Druhým testovacím příkladem, skalárním řízením asynchronního stroje s klecovou kotvou, byla ověřována schopnost balíčku DynSyPy simulovat složitější struktury systémů. V tomto testovacím příkladu byl také proveden návrh PI regulátoru otáček ASM a to dvěma způsoby. Nejprve byl regulátor navržen pomocí ručního ladění na modelu vytvořeném v Matlab Simulink (na předchozím příkladu bylo prokázáno, že se modely shodují). Návrh byl také proveden použitím nástroje Control System Designer dostupného v programu Matlab. K tomu bylo potřeba získat linearizované modely nelineárního systému (motor + měnič + algoritmus skalárního řízení) v několika pracovních bodech. To bylo provedeno pomocí experimentální identifikace a s využitím nástroje System Identification Toolbox také dostupného v programu Matlab. Z navržených regulátorů v celém pracovním rozsahu vyhověl regulátor navržený pomocí ručního ladění a potom regulátory pro linearizované systémy identifikované v pracovních bodech $\omega_\mathrm{m} = 30\ \mathrm{rad} \cdot \mathrm{s}^{-1}$ a $\omega_\mathrm{m} = 60\ \mathrm{rad} \cdot \mathrm{s}^{-1}$. V případě, že by šlo o návrh regulátoru pro reálnou aplikaci, bylo by vhodné zvážit nasazení 2DoF regulátoru, případně navrhnout různé regulátory pro různé pracovní oblasti. V případě stroje s těmito parametry by bylo také vhodnější nasazení vektorového řízení namísto skalárního.

Výstupy druhého testovacího příkladu byly také vyneseny do grafů a jsou uvedeny na obrázcích 6.14 až 6.19. Z nich je zřejmé, že aplikované skalární řízení je v obou nástrojích plně funkční. Úměrně ke zvyšující se frekvenci dochází ke zvyšování amplitudy statorového napětí a naopak. Z grafů je také patrné, že omezení rotorové frekvence na výstupu z regulátoru na maximální hodnotu 5 Hz způsobuje pozvolný nárůst frekvence a amplitudy statorového napětí. Je však také patrné, že výsledky získané pomocí DynSyPy se od výstupů z Matlab Simulink mírně liší ve tvaru přechodných dějů, jako např. jiné velikosti překmitů v mechanické úhlové rychlosti rotoru. Metody použité v balíčku DynSyPy pravděpodobně nejsou dostatečné pro provádění simulací systémů spojených do uzavřené smyčky.

Výsledný balíček v jazyce Python je schopen spolehlivě modelovat LTI systémy, asynchronní stroj s klecovou kotvou, jednoduché neřízené zdroje, jako jsou jednotkový skok a harmonické funkce, a řízený n-fázový harmonický zdroj. Balíček je také doplněn o třídy pro realizaci skalárního řízení a PI regulátoru s anti-windup ochranou. Všechny tyto systémy je možné mezi sebou propojovat a simulovat tyto systémy pohromadě s pevným nebo proměnným krokem simulace. Při simulacích systémů spojených do uzavřené smyčky však dochází k problémům.

V rámci dalšího vývoje balíčku DynSyPy je v plánu zobecnění implementovaného modelu ASM (např. doplnění koeficientu tření) a doplnění modelu v rotující soustavě d, q. Dále by bylo vhodné rozšíření modelů strojů např. o synchronní stroje. Vzhledem k výsledkům druhého testovacího příkladu bude však nutné nejprve odstranit problémy se simulací v uzavřené smyčce.

Použitá literatura

[1] *_TINY_ACMSIMC*. [online]. GitHub repository. 6. květ. 2020. URL: https://github.com/tinyko/_TINY_ACMSIMC [cit. 28. 03. 2023].

[2] *ACMSimPy*. [online]. GitHub repository. 19. zář. 2022. URL: https://github.com/horychen/ACMSimPy [cit. 28. 03. 2023].

[3] ÅSTRÖM, Karl Johan a Tore HÄGGLUND. *PID Controllers: Theory, Design, and Tuning*. Research Triangle Park, North Carolina: ISA - The Instrumentation, Systems a Automation Society, 1995. ISBN: 1-55617-516-7.

[4] BOEING, Geoff. „Visual Analysis of Nonlinear Dynamical Systems: Chaos, Fractals, Self-Similarity and the Limits of Prediction“. In: *Systems* 4.4 (2016), s. 37. ISSN: 2079-8954. DOI: 10.3390/systems4040037.

[5] Robert Clewley et al. *PyDSTool, a software environment for dynamical systems modeling*. [online]. 2007. URL: http://pydstool.sourceforge.net [cit. 08. 12. 2022].

[6] HOFFSTADT, Jonathan a Preston COTHREN. *Dear PyGui*. [online]. 2023. URL: https://dearpygui.readthedocs.io/_/downloads/en/latest/pdf/ [cit. 29. 03. 2023].

[7] CHEN, Jiahao et al. „Overview of Fundamental Frequency Sensorless Algorithms for AC Motors: A Unified Perspective“. In: *IEEE Journal of Emerging and Selected Topics in Power Electronics* 11.1 (2023), s. 915–931. DOI: 10.1109/JESTPE.2022.3194520.

[8] Plotly Technologies Inc. *Collaborative data science*. Montreal, QC, 2015. URL: https://plot.ly.

[9] KADLEC, Martin. „Modul pro modelování dynamických systémů“. Bakalářská práce. Plzeň: Západočeská univerzita v Plzni. Fakulta elektrotechnická, 2021. URL: http://hdl.handle.net/11025/44851.

[10] KADLEC, Martin. *DynSyPy*. [online]. GitHub repository. 11. květ. 2023. URL: https://github.com/kadlec99/DynSyPy [cit. 17. 02. 2023].

[11] MARINO, Riccardo, Patrizio TOMEI a Cristiano M. VERRELLI. *Induction Motor Control Design*. London: Springer-Verlag, 2010. ISBN: 978-1-84996-284-1.

[12] MELICHAR, Jiří a Martin GOUBEJ. *Lineární systémy 1.* Plzeň: Západočeská univerzita v Plzni, 2017.

[13] MELICHAR, Jiří a Martin GOUBEJ. *Lineární systémy 2.* Plzeň: Západočeská univerzita v Plzni, 2019.

[14] *Mnohofázové soustavy.* [online]. Západočeská univerzita v Plzni. 18. ún. 2022. URL: https://courseware.zcu.cz/CoursewarePortlets2/DownloadDokumentu?id=214223 [cit. 26. 04. 2023].

[15] ONG, Chee-Mun. *Dynamic Simulation of Electric Machinery Using Matlab/Simulink.* New Jersey: Prentice Hall PTR, 1998. ISBN: 978-0-13-723785-2.

[16] PEROUTKA, Zdeněk a kol. *Pohony a výkonová elektronika 2, přednáška č. 1. Úvod, modely a transformace.* [online]. Plzeň: Západočeská univerzita v Plzni, 2020. URL: https://portal.zcu.cz/CoursewarePortlets2/DownloadDokumentu?id=187827 [cit. 06. 03. 2023].

[17] PEROUTKA, Zdeněk a kol. *Pohony a výkonová elektronika 2, přednáška č. 8. Obecná teorie střídavých el. strojů, modely asynchronního stroje a úvod do jeho řízení.* [online]. Plzeň: Západočeská univerzita v Plzni, 2020. URL: https://portal.zcu.cz/CoursewarePortlets2/DownloadDokumentu?id=192819 [cit. 07. 01. 2023].

[18] PEROUTKA, Zdeněk a kol. *Pohony a výkonová elektronika 2, přednáška č. 9. Řízení pohonů s asynchronním motorem – část 1/2.* [online]. Plzeň: Západočeská univerzita v Plzni, 2020. URL: https://portal.zcu.cz/CoursewarePortlets2/DownloadDokumentu?id=193353 [cit. 13. 03. 2023].

[19] *Pohony a výkonová elektronika 2. Courseware ZČU.* [online]. Západočeská univerzita v Plzni. 2020. URL: https://portal.zcu.cz/portal/studium/courseware/kev/pve2 [cit. 23. 11. 2022].

[20] RAVEENDRA, M. a M. ASWINI. „Speed Control of Induction Motors using PI Controllers“. In: *International Journal of Scientific Engineering and Technology Research* 07 (03 2018), s. 361–364. ISSN: 2319-8885.

[21] ŠTECHA, Jan a Vladimír HAVLENA. *Teorie dynamických systémů.* 2. vyd. Praha: České vysoké učení technické, 1999. ISBN: 80-01-01971-3.

[22] TŮMA, František. *Teorie řízení.* Plzeň: Západočeská univerzita v Plzni, 1997. ISBN: 80-7082-341-0.

[23] VIRTANEN, Pauli et al. „SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python“. In: *Nature Methods* 17 (2020), s. 261–272. DOI: 10.1038/s41592-019-0686-2.

[24] ZEMAN, Karel, Zdeněk PEROUTKA a Martin JANDA. *Automatická regulace pohonů s asynchronními motory.* Plzeň: Západočeská univerzita v Plzni, 2007. ISBN: 978-80-7043-350-8.

Příloha A

Podklady pro simulace

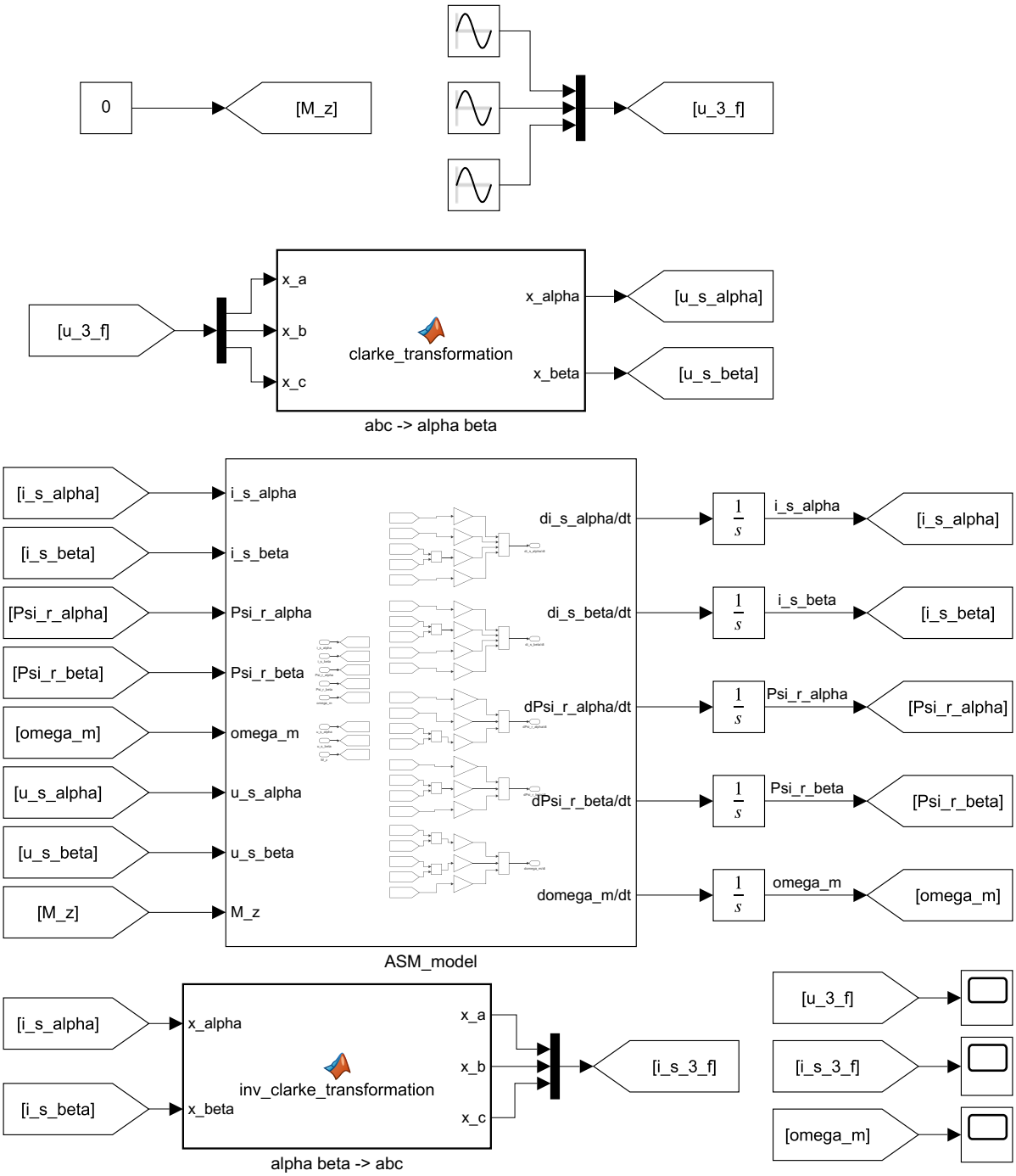
A.1 Model ASM v Matlab Simulink

Realizace funkce `clarke_transformation` viz obr. A.1 a B.2:

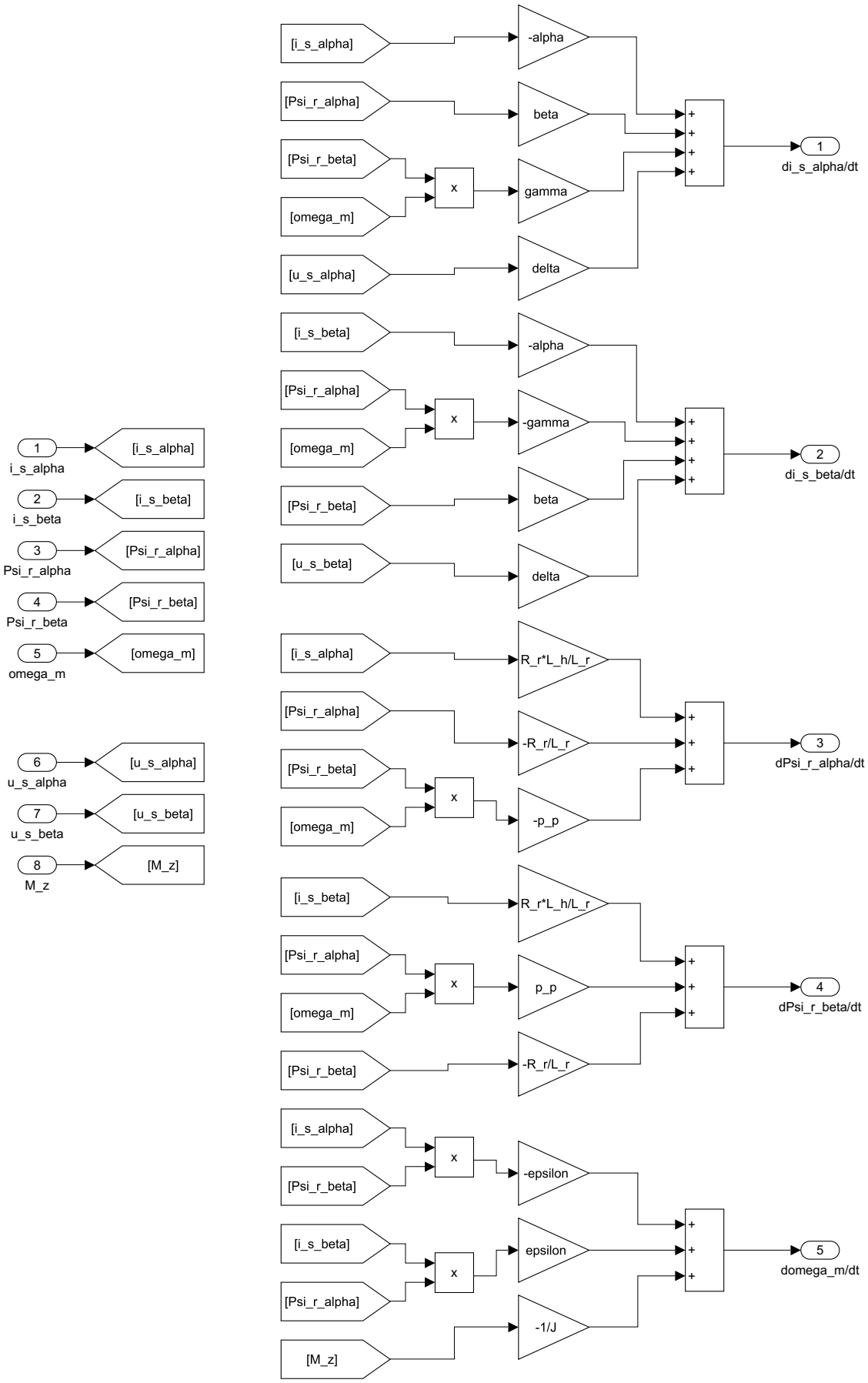
```
function [x_alpha, x_beta] = clarke_transformation(x_a, x_b, x_c)

C_p = [ 1,      -1 / 2,         -1 / 2
        0,      sqrt(3) / 2,    -sqrt(3) / 2    ];

x = 2 / 3 * C_p * [x_a; x_b; x_c];

x_alpha = x(1);
x_beta = x(2);
```

Realizace funkce `inv_clarke_transformation` viz obr. A.1 aB.2:

```
function [x_a, x_b, x_c] = inv_clarke_transformation(x_alpha, x_beta)

C_p_inv = [ 1,          0
            -1 / 2,     sqrt(3) / 2
            -1 / 2,     -sqrt(3) / 2    ];

x = C_p_inv * [x_alpha; x_beta];

x_a = x(1);
x_b = x(2);
x_c = x(3);
```

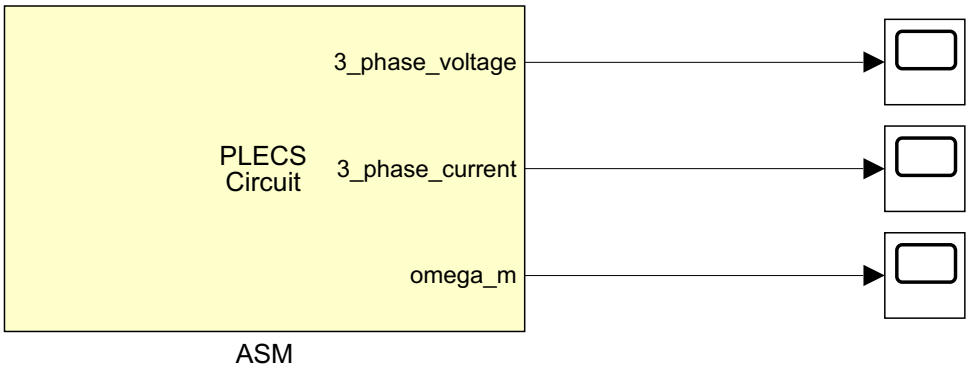


Obr. A.1: Model ASM pomocí Matlab Simulink, přímé připojení ASM ke zdroji napětí



Obr. A.2: Realizace bloku `ASM_model` pomocí Matlab Simulink, viz obr. A.1 a B.2

A.2 Model ASM v Matlab Simulink pomocí PLECS

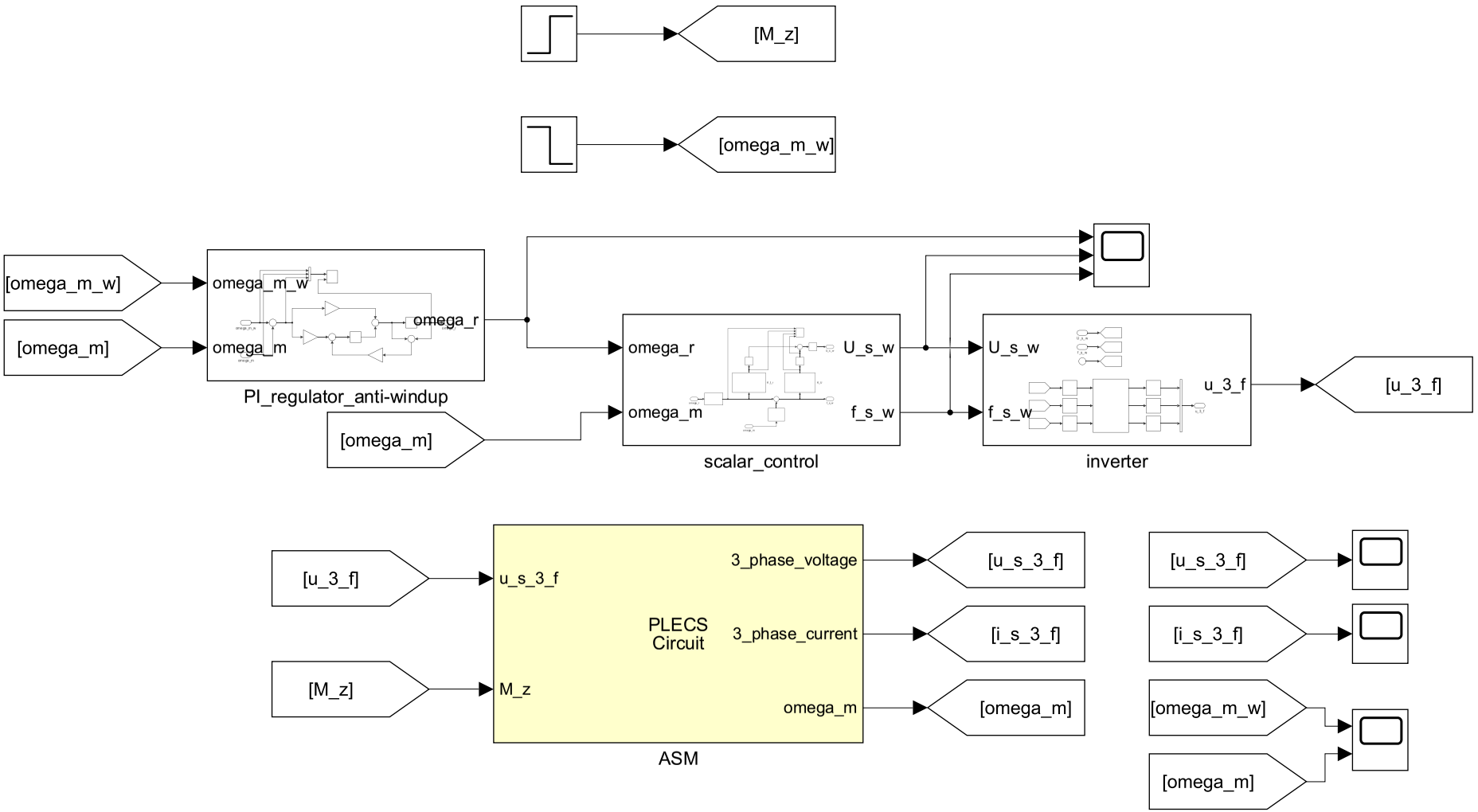


Obr. A.3: Model ASM pomocí nástroje PLECS, obvod v bloku `ASM` je uveden na obr. 6.1

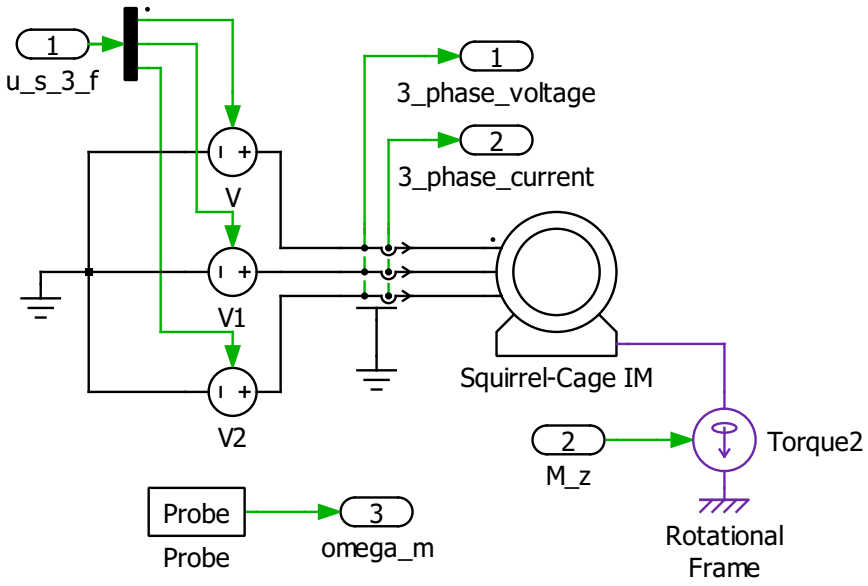
A.3 Model skalárního řízení ASM v Matlab Simulink pomocí PLECS

Realizace funkce `sine_3_phase` viz obr. A.8:

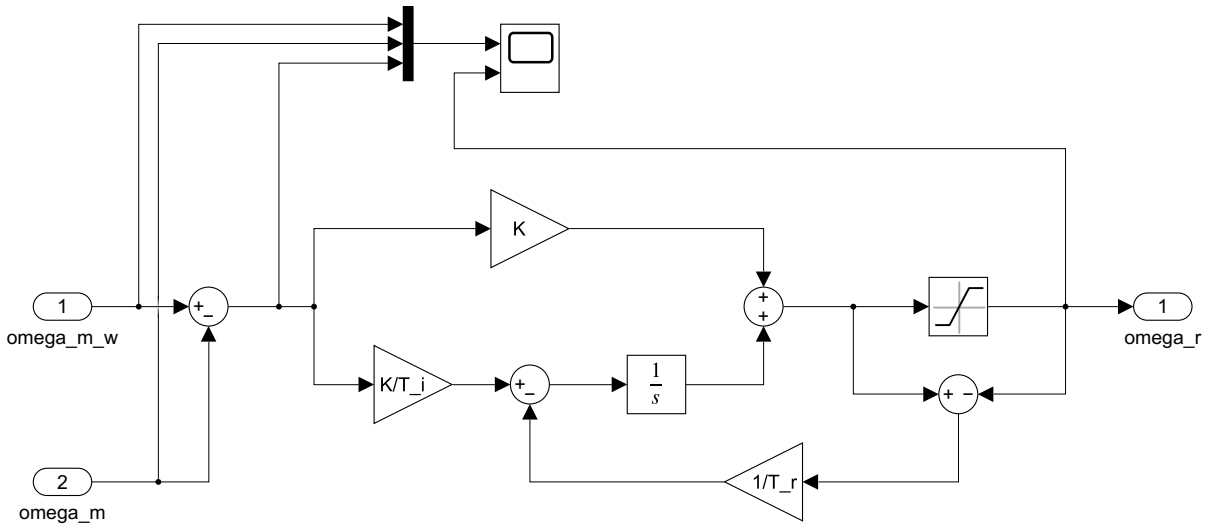
```
function [u_a, u_b, u_c] = sine_3_phase(U_s_w, f_s_w, t)

persistent beta;
persistent t_prev;

if isempty(t_prev)
    t_prev = 0;
end

if isempty(beta)
    beta = 0;
else
    delta_t = t - t_prev;
    beta = beta + 2 * pi * f_s_w * delta_t;
end

if beta > pi
    beta = beta - 2 * pi;
end

if beta < -pi
    beta = beta + 2 * pi;
end

u_a = U_s_w * sin(beta);
u_b = U_s_w * sin(beta - 2 * pi / 3);
u_c = U_s_w * sin(beta + 2 * pi / 3);

t_prev = t;
```



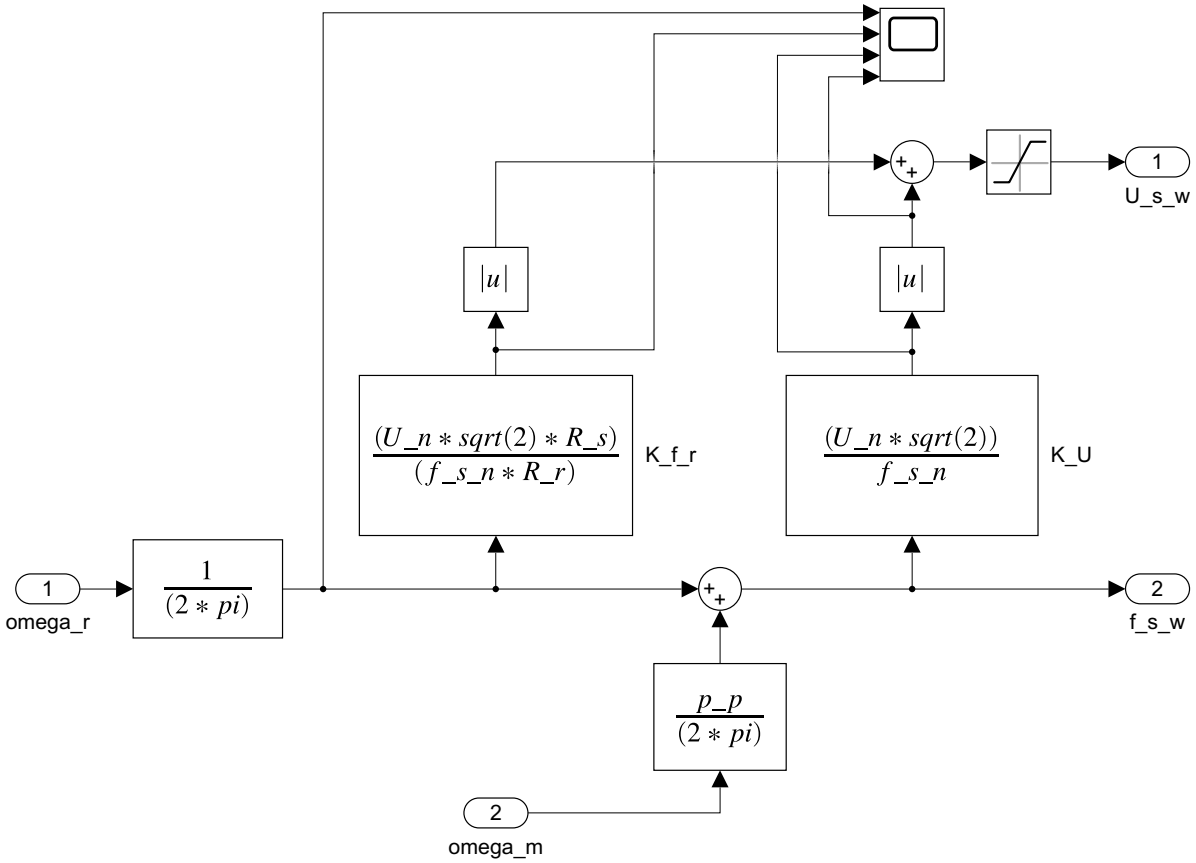
Obr. A.4: Realizace skalárního řízení ASM v Matlab Simulink pomocí PLECS



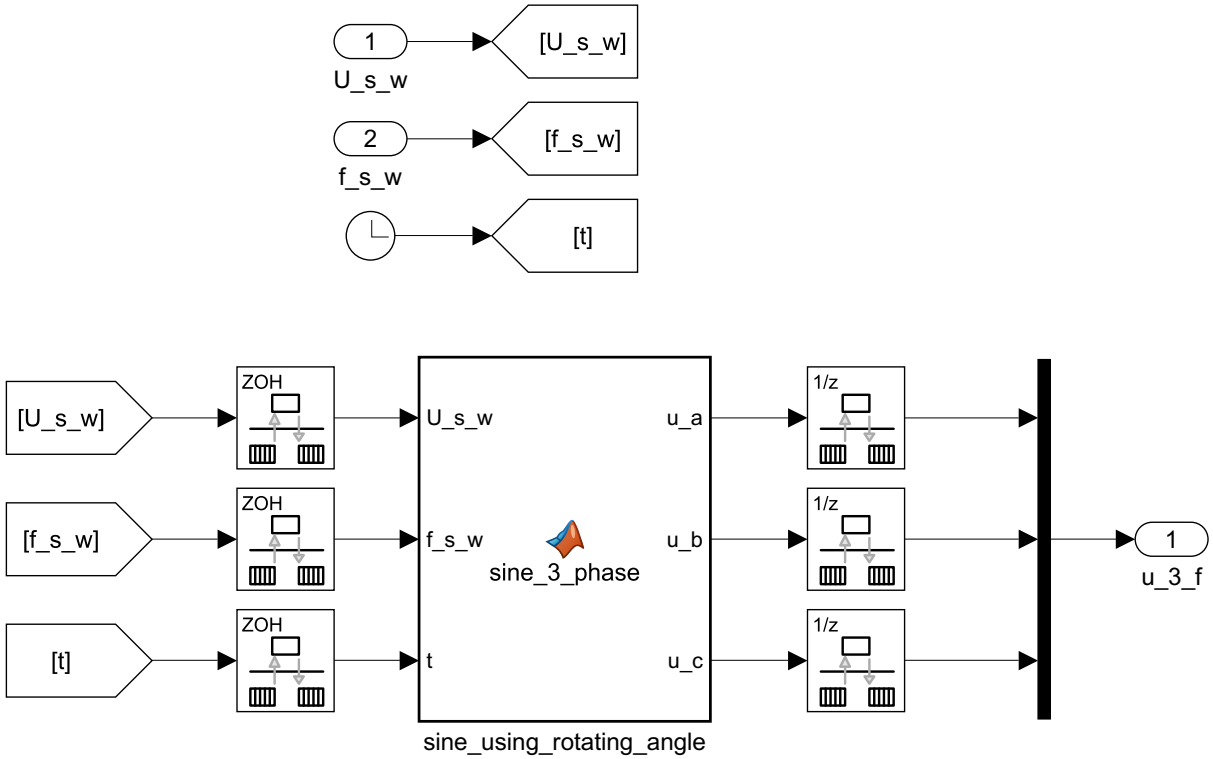
Obr. A.5: Realizace modelu ASM pro skalární řízení pomocí PLECS, blok `ASM` viz obr. A.4



Obr. A.6: Realizace bloku `PI_regulator_anti-windup`, viz obr. A.4 a B.1



Obr. A.7: Realizace bloku `scalar_control`, viz obr. A.4 a B.2



Obr. A.8: Realizace bloku `inverter`, viz obr. A.4 a B.2

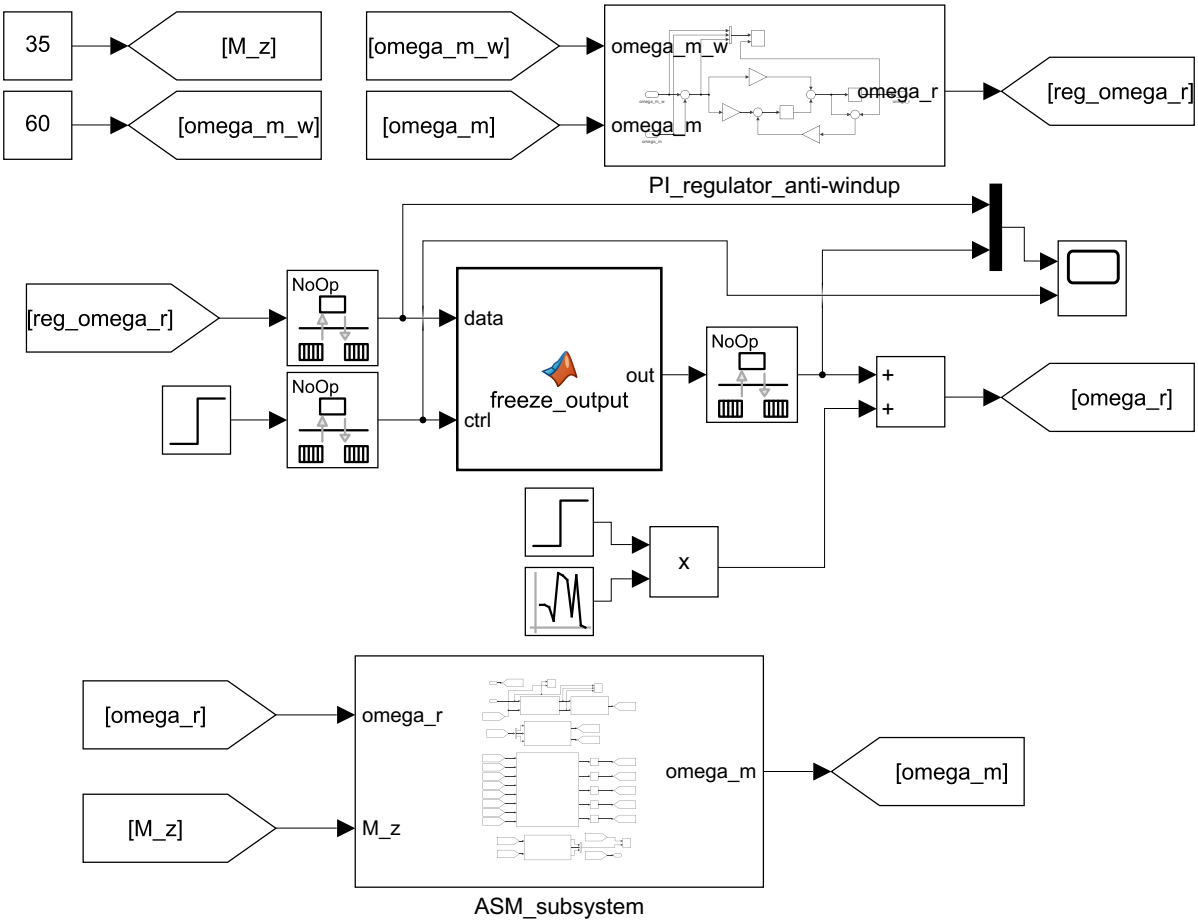
Příloha B

Podklady pro realizaci experimentální identifikace

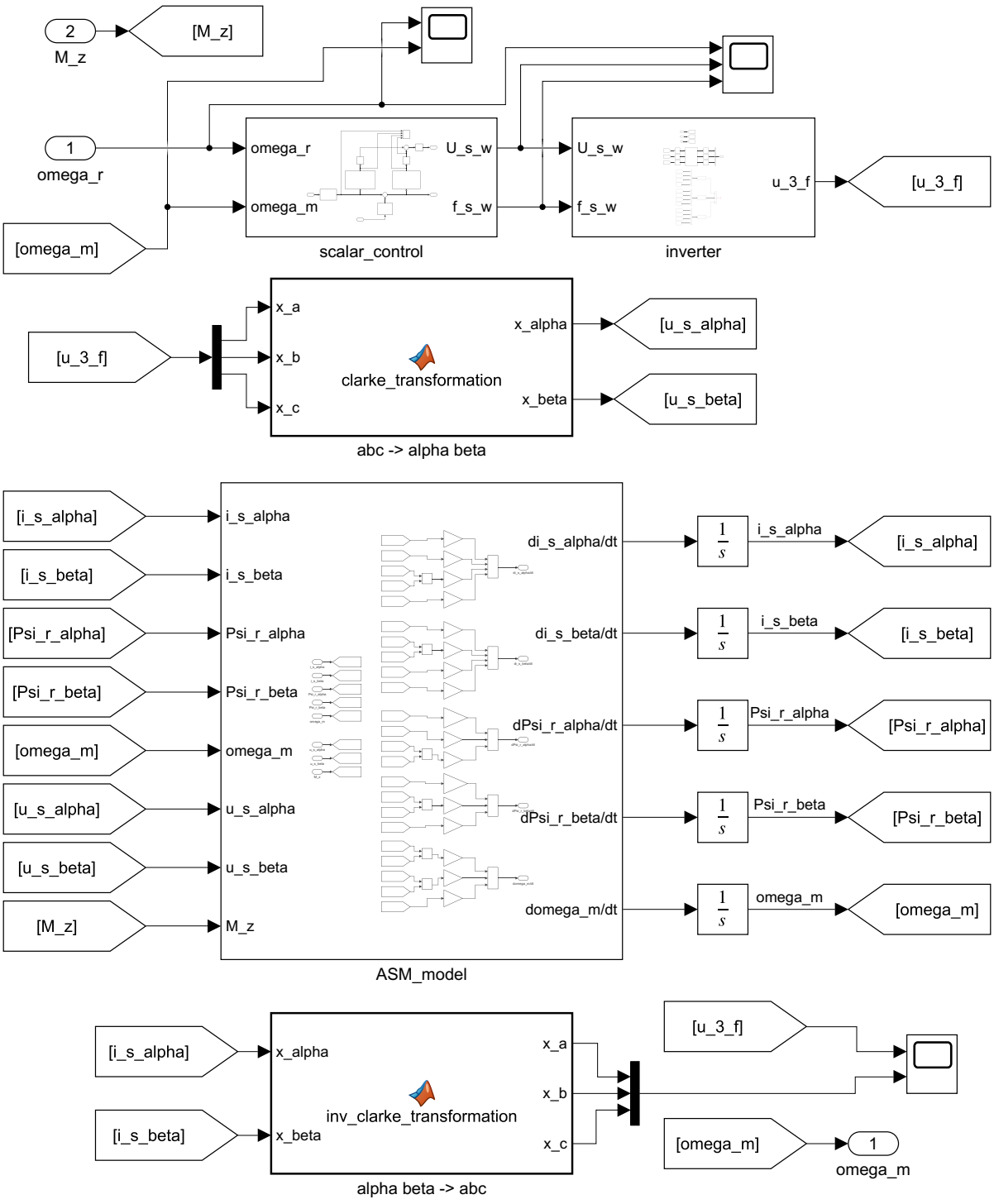
B.1 Model v Matlab Simulink pro experimentální identifikaci

Realizace funkce `freeze_output` viz obr. B.1:

```
function out = freeze_output(data, ctrl)

persistent memory

if isempty(memory)
    memory = data;
end

if ctrl <= 0
    memory = data;
end

out = memory;
```



Obr. B.1: Model v Matlab Simulink použitý pro experimentální identifikaci



Obr. B.2: Realizace bloku `ASM_subsystem`, viz obr. B.1