Fakulta elektrotechnická
Katedra elektrotechniky a počítačového modelování

Bakalářská práce

Modul pro modelování dynamických systémů

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

Plzeň 2021

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta elektrotechnická

Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Martin KADLEC**
Osobní číslo:	**E18B0122P**
Studijní program:	**B2644 Aplikovaná elektrotechnika**
Studijní obor:	**Aplikovaná elektrotechnika**
Téma práce:	**Modul pro modelování dynamických systémů**
Zadávající katedra:	**Katedra výkonové elektroniky a strojů**

Zásady pro vypracování

1. Vytvořte přehled open source nástrojů pro modelování dynamických systémů v jazyce Python.
2. Vytvořte v jazyce Python modul pro modelování dynamických systémů tak, aby jej bylo možné integrovat do platformy Artap.
3. Vytvořte v rámci modulu třídy, které zajistí: zadání LTI systémů stavového popisu a přenosové funkce, analýzu v časové a frekvenční oblasti, jednoduché metody pro návrh regulátorů (pole placement, LQR).

Rozsah bakalářské práce: **30 – 40 stran**
Rozsah grafických prací: **podle doporučení vedoucího**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. Gene F. Franklin, J. David Powell, Abbas Emami-Naeini: Feedback Control of Dynamic Systems, 8th Edition, 2018.

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

Datum zadání bakalářské práce: **9. října 2020**
Termín odevzdání bakalářské práce: **27. května 2021**

L.S.

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

Prof. Ing. Václav Kůs, CSc.
vedoucí katedry

Abstrakt

Bakalářská práce se zabývá možnostmi modelování dynamických systémů s využitím vysokoúrovňového programovacího jazyka Python. Cílem práce je pomocí tohoto jazyka vytvořit balíček, který umožní modelovat lineární časově invariantní systémy (tzv. LTI systémy).

Práce obsahuje základy teorie dynamických systémů. Teorie dynamických systémů zahrnuje možnosti popisu LTI systémů pomocí diferenciálních rovnic a také pomocí přenosu. Dále jsou zde uvedeny základní druhy vazeb mezi LTI systémy. V rámci práce byl také vytvořen přehled open source nástrojů pro modelování dynamických systémů.

Na těchto základech byl vytvořen balíček DynSyPy, který obsahuje třídy a metody, pomocí nichž je schopen vytvářet systémy (LTI systémy a základní zdroje). Tyto systémy je schopen simulovat v časové i frekvenční oblasti.

Funkčnost balíčku DynSyPy byla ověřena na testovacích příkladech stejnosměrného motoru s permanentními magnety a na sériovém RLC obvodu.

Klíčová slova

DynSyPy, LTI systém, simulace, numerická integrace

Abstract

Kadlec, Martin. *Module of dynamic systems modelling* [*Modul pro modelování dynamických systémů*]. Pilsen, 2021. Bachelor thesis (in Czech). University of West Bohemia. Faculty of Electrical Engineering. Department of Electrical Engineering and Computer Modeling. Supervisor: David Pánek

---

The bachelor thesis deals with the possibilities of modelling dynamic systems using the high-level programming language of Python. The aim of the thesis is to use this language in order to create a package that allows modelling linear time invariant systems (so-called LTI systems).

The thesis contains the basics of dynamical systems theory. The theory of dynamic systems includes the possibility of describing LTI systems by means of differential equations and by means of transference. The basic types of couplings between LTI systems are also presented. The thesis includes an overview of open source tools for modelling dynamical systems as well.

The DynSyPy package has been developed on the mentioned basis and contains classes and methods that can be used to create systems (LTI systems and primary sources). It is able to simulate those systems in both time and frequency domains.

The functionality of the DynSyPy package has been verified on test examples of a permanent magnet DC motor and a series RLC circuit.

Keywords

DynSyPy, LTI system, simulation, numerical integration

Prohlášení

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

Prohlašuji, že jsem svou závěrečnou práci vypracoval samostatně pod vedením vedoucího bakalářské 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 bakalářské 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 bakalářské práce, je legální.

V Plzni dne 7. června 2021

Martin Kadlec

. . . . . .

Podpis

Poděkování

Na tomto místě bych rád poděkoval doc. Ing. Davidu Pánkovi Ph.D., vedoucímu bakalářské práce, za odborné vedení, věcné připomínky a podporu při tvorbě této práce.

Obsah

Seznam obrázků **ix**

Seznam symbolů a zkratek **x**

Úvod **1**

1 Úvod do dynamických systémů **3**
1.1 Systém . . . 3
1.2 Dynamický systém . . . 3
1.3 Lineární systém . . . 4
1.3.1 Příklad LTI systému: sériový RLC obvod . . . 5
1.3.2 Přenos systému . . . 6
1.3.3 Spojování systémů . . . 7
1.3.3.1 Paralelní spojení . . . 8
1.3.3.2 Sériové spojení . . . 9
1.3.3.3 Zpětnovazební spojení . . . 10

2 Existující moduly v jazyce Python pro práci s dynamickými systémy **12**
2.1 SciPy . . . 12
2.2 SimuPy . . . 12
2.3 PyDSTool . . . 13
2.4 S-timator . . . 13
2.5 Pynamical . . . 13
2.6 Knihovny PySD a BPTK_Py . . . 14

3 Balíček DynSyPy **15**
3.1 Třída Pool . . . 16
3.1.1 Metoda `add()` . . . 18
3.1.2 Metoda `simulate()` . . . 18
3.2 Třída System . . . 18
3.2.1 Metoda `connect()` . . . 19
3.2.2 Metoda `step()` . . . 20
3.2.3 Metoda `runge_kutta_step()` . . . 21

3.2.4 Metoda `adaptive_step()` . . . 21
3.2.5 Metoda `update_input()` . . . 23
3.2.6 Metoda `select_output()` . . . 23
3.2.7 Metoda `output()` . . . 23
3.2.8 Metoda `linear_regression()` . . . 24
3.2.9 Metoda `null_input()` . . . 25
3.2.10 Metody `equation_of_state()` a `update_output()` . . . 26
3.3 Třída LinearSystem . . . 26
3.3.1 Metoda `equation_of_state()` . . . 27
3.3.2 Metoda `update_output()` . . . 27
3.3.3 Metoda `matrix_controls()` . . . 27
3.3.4 Metoda `value_control()` . . . 28
3.3.5 Metoda `matrix_shape_control()` . . . 28
3.3.6 Metoda `frequency_analysis()` . . . 28
3.4 Třída Source . . . 29
3.4.1 Metody `step()` a `adaptive_step()` . . . 30
3.4.2 Metoda `update_state()` . . . 30
3.4.3 Abstraktní metoda `source_function()` . . . 30
3.4.4 Metoda `update_output()` . . . 31
3.4.5 Metoda `output()` . . . 31
3.5 Abstraktní třída HarmonicFunctions . . . 31
3.6 Třídy Sine a Cosine . . . 32
3.7 Třída UnitStep . . . 32

4 Činnost balíčku DynSyPy **34**
4.1 Testovací příklad – stejnosměrný motor s permanentními magnety . . . 34
4.1.1 Simulace systému pomocí DynSyPy . . . 36
4.1.2 Výsledky simulací . . . 39
4.2 Testovací příklad – Sériový RLC obvod . . . 42
4.2.1 Realizace frekvenční analýzy systému pomocí DynSyPy . . . 43
4.2.2 Výsledky frekvenční analýzy . . . 45

Závěr **48**

Reference, použitá literatura **49**

Přílohy **51**

A Výstupy ze simulace **51**
A.1 Grafy průběhů proudu a otáček stejnosměrného motoru s permanentními magnety – průběhy z nástroje MATLAB Simulink . . . 51

A.2 Amplitudová a fázová frekvenční charakteristika sériového RLC obvodu – průběhy z programu LTspice 56

Seznam obrázků

1.1 Blokové schéma spojitého lineárního dynamického systému 4
1.2 Sériový RLC obvod napájený stejnosměrným zdrojem napětí 5
1.3 Paralelní spojení systémů 8
1.4 Sériové spojení systémů 9
1.5 Zpětnovazební spojení systémů 10

3.1 Diagram tříd (UML) balíčku DynSyPy 15
3.2 Znázornění synchronizace systémů pomocí třídy Pool 16
3.3 Diagram činnosti metody `step()` 20
3.4 Diagram činnosti metody `output()` 24
3.5 Znázornění činnosti metody `matrix_controls()` 27
3.6 Graf závislosti stavu `self.x` zdroje třídy UnitStep na čase `self.t` 33

4.1 Stejnosměrný motor s permanentními magnety 34
4.2 Náhradní schéma stejnosměrného motoru 35
4.3 Vykreslené průběhy pro hodnoty vstupu $u_a = 4\,\mathrm{V}$ a $M_L = 0,1\,\mathrm{Nm}$ 39
4.4 Vykreslené průběhy pro hodnoty vstupu $u_a = 5\,\mathrm{V}$ a $M_L = 0,1\,\mathrm{Nm}$ 39
4.5 Realizace testovacího příkladu 4.1 pomocí MATLAB Simulink 40
4.6 Průběh proudu $i_a(t)$ pro hodnoty vstupu $u_a = 4\,\mathrm{V}$ a $M_L = 0,1\,\mathrm{Nm}$ 41
4.7 Průběh úhlové frekvence $\omega(t)$ pro hodnoty vstupu $u_a = 4\,\mathrm{V}$ a $M_L = 0,1\,\mathrm{Nm}$ 41
4.8 Průběh proudu $i_a(t)$ pro hodnoty vstupu $u_a = 5\,\mathrm{V}$ a $M_L = 0,1\,\mathrm{Nm}$ 42
4.9 Průběh úhlové frekvence $\omega(t)$ pro hodnoty vstupu $u_a = 5\,\mathrm{V}$ a $M_L = 0,1\,\mathrm{Nm}$ 42
4.10 Schéma zapojení RLC obvodu pro frekvenční analýzu 43
4.11 Amplitudová a fázová frekvenční charakteristika pomocí balíčku DynSyPy . 45
4.12 Realizace testovacího příkladu 4.2 pomocí LTspice 46
4.13 Amplitudová frekvenční charakteristika z hodnot získaných pomocí nástroje LTspice 46
4.14 Fázová frekvenční charakteristika z hodnot získaných pomocí nástroje LTspice 47

A.1 Průběh proudu $i_a(t)$ pro hodnoty vstupu $u_a = 4\,\mathrm{V}$ a $M_L = 0,1\,\mathrm{Nm}$ 52
A.2 Průběh úhlové frekvence $\omega(t)$ pro hodnoty vstupu $u_a = 4\,\mathrm{V}$ a $M_L = 0,1\,\mathrm{Nm}$ 53
A.3 Průběh proudu $i_a(t)$ pro hodnoty vstupu $u_a = 5\,\mathrm{V}$ a $M_L = 0,1\,\mathrm{Nm}$ 54
A.4 Průběh úhlové frekvence $\omega(t)$ pro hodnoty vstupu $u_a = 5\,\mathrm{V}$ a $M_L = 0,1\,\mathrm{Nm}$ 55

A.5 Amplitudová a fázová frekvenční charakteristika sériového RLC obvodu . . 57

Seznam symbolů a zkratek

A	Modul přenosu [dB].
$\boldsymbol{A}$	Matice systému o rozměrech ($n \times n$).
A_m	Amplituda harmonické funkce.
$\boldsymbol{B}$	Vstupní matice o rozměrech ($n \times r$).
$\boldsymbol{b}$	Vektor pravé strany.
B_m	Viskózní tření [$\mathrm{N \cdot m \cdot s}$].
C	Kapacita [F].
$\boldsymbol{C}$	Výstupní matice o rozměrech ($m \times n$).
$\boldsymbol{D}$	Matice přímého působení vstupu na výstup o rozměrech ($m \times r$).
f	Frekvence [Hz].
$F(p)$	Přenos lineárního spojitého dynamického systému.
$\boldsymbol{F}(p)$	Přenosová matice o rozměrech ($m \times r$).
h	Velikost integračního kroku.
$\boldsymbol{I}$	Jednotková matice.
i_a	Proud tekoucí obvodem kotvy [A].
i_L	Proud induktorem (stavová veličina) [A].
J	Moment setrvačnosti [$\mathrm{kg \cdot m^2}$].
k	Směrnice přímky.
k_a	Konstanta motoru [$\mathrm{V \cdot s \cdot rad^{-1}}$].
k_a	Konstanta motoru [$\mathrm{N \cdot m \cdot A^{-1}}$].
L	Vlastní indukčnost [H].
L_a	Indukčnost kotvy [H].
M	Moment motoru [Nm].
m	Počet řádků vektoru výstupu.
M_L	Moment zátěže [Nm].
n	Počet řádků vektoru stavu.
q	Posun přímky ve směru osy y.
R	Elektrický odpor [Ω].
R_a	Odpor kotvy [Ω].
R_k	Kritický odpor [Ω].
r	Počet řádků vektoru vstupu.

t	Čas [s].
$\boldsymbol{u}$	Vektor vstupu systému.
U_0	Napětí stejnosměrného zdroje [V].
u_a	Napájecí napětí obvodu kotvy [V].
u_C	Napětí na kapacitoru (stavová veličina) [V].
$U(p)$	Laplaceův obraz vstupní veličiny.
$\boldsymbol{U}(p)$	Laplaceův obraz vektoru vstupu.
$\boldsymbol{x}$	Vektor stavu systému.
$\boldsymbol{y}$	Vektor výstupu systému.
$Y(p)$	Laplaceův obraz výstupní veličiny.
$\boldsymbol{Y}(p)$	Laplaceův obraz vektoru výstupu.
φ	Fázový posun [rad].
$\boldsymbol{\Phi}(p)$	Matice přechodových funkcí (přechodová matice).
ω	Úhlová frekvence $[\,\mathrm{rad} \cdot \mathrm{s}^{-1}\,]$.
ABM	Agent-Based Modeling. Multiagentní modelování.
API	Application Programming Interface. Rozhraní pro programování aplikací.
DAE	Differential-Algebraic Equation. Diferenciálně-algebraická rovnice.
LTI	Linear Time-Invariant (system). Lineární časově nezávislý (systém).
MKP	Metoda konečných prvků.
ndarray	N-dimensional array. Pole o n dimenzích z vědecké knihovny NumPy.
ODE	Ordinary Differential Equation. Obyčejná diferenciální rovnice.
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.
UML	Unified Modeling Language. Grafický jazyk pro vizualizaci, specifikaci, navrhování a dokumentaci programových systémů.

Úvod

Práce se zabývá návrhem balíčku pro modelování a simulaci lineárních časově invariantních (LTI – linear time-invariant) systémů v jazyce Python a možností jeho propojení s již existujícím nástrojem pro optimalizace a robustní návrh Artap, viz [9][13]. V rámci projektu Artap byly implementovány nástroje pro snižování řádu systémů modelovaných pomocí metody konečných prvků (MKP). Nově přidaná funkčnost by měla umožnit pohodlnou práci s těmito redukovanými systémy.

Hlavní cíle práce jsou:

- vytvořit přehled open source nástrojů pro modelování dynamických systémů v jazyce Python,
- vytvořit balíček v jazyce Python, který je schopen modelovat LTI systémy.

Důvody pro vytvoření přehledu open source nástrojů byly následující. Prvním důvodem bylo zhodnocení, zda požadovaná funkčnost není již obsažena v nějakém existujícím balíčku. Druhým důvodem bylo využití již existujících řešení dílčích úkolů.

Nový balíček v jazyce Python by měl splňovat následující kritéria:

- Měl by obsahovat nástroje pro zajištění vytvoření LTI systému.
- Měl by obsahovat nástroje pro zajištění analýzy systémů v časové a frekvenční oblasti.
- Při návrhu se předpokládá, že bude možné simulovat jak systémy spojité v čase, tak systémy časově diskrétní.
- Balíček musí být navržen tak, aby bylo možné pracovat s více vzorkovacími kmitočty zároveň (multi-rate systémy) a aby bylo v budoucnu možné zahrnout nelineární systémy (plánované rozšíření balíčku).
- Navržený balíček musí být možné integrovat do platformy Artap.

První část práce je věnována úvodu do problematiky dynamických systémů. Zde je uvedena definice lineárního dynamického systému, dále možnosti popisu těchto systémů. Také jsou zde popsány možnosti základních vazeb mezi těmito systémy.

Druhá část práce obsahuje vypracovanou rešerši open source nástrojů pro modelování dynamických systémů, kde jsou rozebrány základní funkčnosti těchto nástrojů.

Ve třetí části práce je rozebírán balíček DynSyPy, který byl v rámci bakalářské práce vytvořen. Věnuje se popisu tříd a metod obsažených v balíčku. Každá třída i metoda je zde popsána a vysvětlena jejich funkčnost.

Poslední část se zabývá testováním balíčku DynSyPy. Splnění jednotlivých požadavků na tento balíček je ověřeno v rámci dvou testovacích příkladů. Prvním příkladem je stejnosměrný motor s permanentními magnety, který slouží k ověření funkčnosti metod pro simulaci v časové oblasti. Druhým příkladem je sériový RLC obvod, na němž je ověřována analýza ve frekvenční oblasti.

Kapitola 1

Úvod do dynamických systémů

Cílem bakalářské práce je navrhnout a implementovat knihovnu pro práci s lineárními časově invariantními (LTI – linear time-invariant) dynamickými systémy. V této kapitole budou definovány základní pojmy z teorie systémů, které budou dále využívány v rámci celé práce.

1.1 Systém

Systém je definován jako množina prvků navzájem spolu souvisejících a nějak vázaných na své okolí. Problematika obecných systémů je však příliš rozsáhlá. Pro účely této práce se omezíme pouze na popis dynamických systémů.[16]

1.2 Dynamický systém

Dynamický systém je specifický matematický způsob popisu reálného zařízení sestávající ze soustavy diferenciálních rovnic (stavových) reprezentující popisovanou dynamiku a ze soustavy lineárních rovnic (výstupních), která popisuje jakým způsobem se systém projevuje navenek. Dynamický systém je popsán pomocí rovnic:

$$\dot{\boldsymbol{x}}(t) = f\left(\boldsymbol{x},\ \boldsymbol{u},\ t\right), \tag{1.1}$$

$$\boldsymbol{y}(t) = g\left(\boldsymbol{x},\ \boldsymbol{u},\ t\right), \tag{1.2}$$

kde

$\boldsymbol{u}$ je vektor vstupu o rozměrech $(r \times 1)$,

$\boldsymbol{x}$ je vektor stavu systému o rozměrech $(n \times 1)$,

$\boldsymbol{y}$ je vektor výstupu o rozměrech $(m \times 1)$,

t je čas [s].

Stav systému $\boldsymbol{x}$ je soubor vnitřních veličin systému, který v sobě zahrnuje veškerou informaci o minulém vývoji systému. Za znalosti stavu systému v čase t_0 a vstupu $u(t)$ na intervalu $t \in \langle t_0, t_1 \rangle$ je možné jednoznačně určit vektor výstupních veličin $\boldsymbol{y}(t)$. [14][16]

1.3 Lineární systém

V obecném případě jsou stavové rovnice nelineární a jejich řešení může být velmi obtížné. Jednoduší třídu dynamických systémů představují lineární systémy, popsány rovnicemi

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}(t)\,\boldsymbol{x}(t) + \boldsymbol{B}(t)\,\boldsymbol{u}(t)\,, \tag{1.3}$$

$$\boldsymbol{y}(t) = \boldsymbol{C}(t)\,\boldsymbol{x}(t) + \boldsymbol{D}(t)\,\boldsymbol{u}(t)\,, \tag{1.4}$$

kde

$\boldsymbol{A}$ představuje matici systému o rozměrech $(\,n \times n\,)$,

$\boldsymbol{B}$ je maticí vstupu o rozměrech $(\,n \times r\,)$,

$\boldsymbol{C}$ je maticí výstupu o rozměrech $(\,m \times n\,)$,

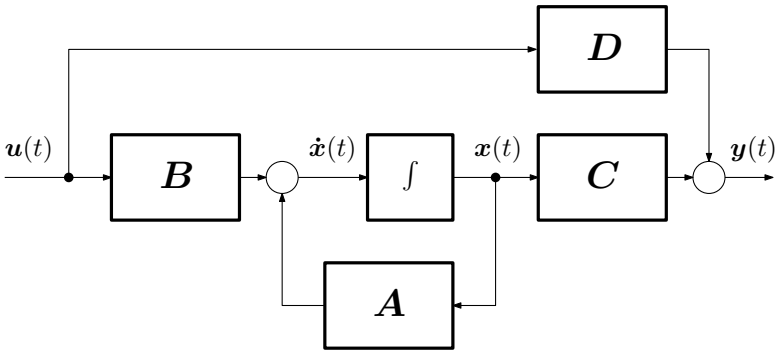
$\boldsymbol{D}$ je maticí přímého působení vstupu na výstup o rozměrech $(\,m \times r\,)$.

Rozměry těchto matic vycházejí z pravidel maticového násobení a jsou tedy závislé na rozměrech vektorů $\boldsymbol{u}$, $\boldsymbol{x}$ a $\boldsymbol{y}$ uvedených v sekci 1.2. [16]

Pravidla pro násobení matic udávají, že počet sloupců první matice musí být stejný, jako je počet řádků druhé matice, aby bylo možné tyto matice mezi sebou vynásobit.

Dále platí pravidlo, že výsledkem součinu dvou matic je matice, jejíž počet řádků je stejný, jako je počet řádků první matice, a jejíž počet sloupců je stejný, jako je počet sloupců druhé matice.

Zároveň platí, že počet rovnic pro stavové veličiny je stejný, jako je počet stavových veličin (počet řádků vektoru $\boldsymbol{x}$), a počet rovnic pro výstupy je stejný, jako je počet výstupů (počet řádků vektoru $\boldsymbol{y}$).



Obr. 1.1: Blokové schéma spojitého lineárního dynamického systému [14]

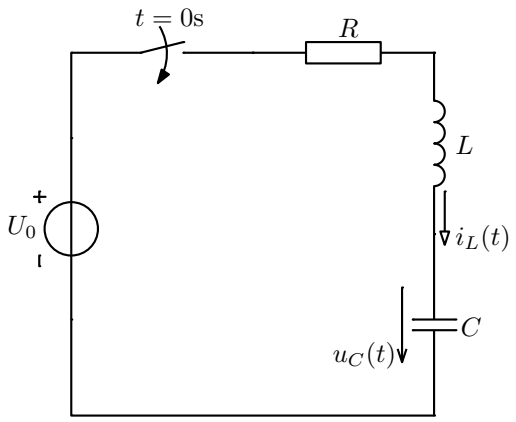
Tato práce je zaměřena na speciální případ lineárních systémů – lineární časově invariantní systémy, tedy takové systémy, kde matice $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ a $\boldsymbol{D}$ jsou konstantní a rovnice lze tedy upravit do tvaru

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\,\boldsymbol{x}(t) + \boldsymbol{B}\,\boldsymbol{u}(t)\,, \tag{1.5}$$

$$\boldsymbol{y}(t) = \boldsymbol{C}\,\boldsymbol{x}(t) + \boldsymbol{D}\,\boldsymbol{u}(t)\,. \tag{1.6}$$

Soustavu diferenciálních rovnic popisující LTI systém je v některých případech možné řešit analyticky. Pro obecný vstup je však rovnice nezbytné řešit numericky. Numerickým metodám řešení soustavy diferenciálních rovnic se věnují kapitoly 3.2.3 a 3.2.4.

1.3.1 Příklad LTI systému: sériový RLC obvod



Obr. 1.2: Sériový RLC obvod napájený stejnosměrným zdrojem napětí

Sériový RLC obvod – viz obr. 1.2, je popsán následující soustavou diferenciálních rovnic:

$$U_0 = Ri_L + L\frac{\mathrm{d}i_L}{\mathrm{d}t} + u_C\ , \tag{1.7}$$

$$i_L = C\frac{\mathrm{d}u_C}{\mathrm{d}t}\ , \tag{1.8}$$

kde první rovnice (1.7) byla sestavena na základě druhého Kirchhoffova zákona a druhá rovnice (1.8) představuje vztah mezi napětím a proudem na kapacitě. Úpravou těchto rovnic vznikne soustava stavových rovnic:

$$\frac{\mathrm{d}i_L}{\mathrm{d}t} = -\frac{R}{L}i_L - \frac{1}{L}u_C + \frac{1}{L}U_0\ , \tag{1.9}$$

$$\frac{\mathrm{d}u_C}{\mathrm{d}t} = \frac{1}{C}i_L\ . \tag{1.10}$$

Tuto soustavu lze zapsat v maticovém tvaru:

$$\begin{bmatrix} \dfrac{\mathrm{d}i_L}{\mathrm{d}t} \\ \\ \dfrac{\mathrm{d}u_C}{\mathrm{d}t} \end{bmatrix} = \overbrace{\begin{bmatrix} -\dfrac{R}{L} & -\dfrac{1}{L} \\ \\ \dfrac{1}{C} & 0 \end{bmatrix}}^{\boldsymbol{A}} \begin{bmatrix} i_L \\ u_C \end{bmatrix} + \overbrace{\begin{bmatrix} \dfrac{1}{L} \\ \\ 0 \end{bmatrix}}^{\boldsymbol{B}} \begin{bmatrix} U_0 \end{bmatrix} . \tag{1.11}$$

Bude-li výstupem tohoto systému napětí na kapacitoru, výstupní rovnice bude mít následující tvar:

$$\begin{bmatrix} u_C \end{bmatrix} = \overbrace{\begin{bmatrix} 0 & 1 \end{bmatrix}}^{\boldsymbol{C}} \begin{bmatrix} i_L \\ u_C \end{bmatrix} + \overbrace{\begin{bmatrix} 0 \end{bmatrix}}^{\boldsymbol{D}} \begin{bmatrix} U_0 \end{bmatrix} . \tag{1.12}$$

Jak je již v rovnicích (1.11) a (1.12) naznačeno, matice $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ a $\boldsymbol{D}$ jsou:

$$\begin{aligned} \boldsymbol{A} &= \begin{bmatrix} -\dfrac{R}{L} & -\dfrac{1}{L} \\ \\ \dfrac{1}{C} & 0 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} \dfrac{1}{L} \\ \\ 0 \end{bmatrix}, \\ \boldsymbol{C} &= \begin{bmatrix} 0 & 1 \end{bmatrix}, \quad \boldsymbol{D} = \begin{bmatrix} 0 \end{bmatrix} . \end{aligned} \tag{1.13}$$

1.3.2 Přenos systému

Přenos spojitého lineárního dynamického systému s jedním vstupem a jedním výstupem je definován jako poměr Laplaceových obrazů výstupní a vstupní veličiny při nulových počátečních podmínkách, tedy

$$F(p) = \frac{\mathcal{L}\{y(t)\}}{\mathcal{L}\{u(t)\}} = \left.\frac{Y(p)}{U(p)}\right|_{\text{n.p.p.}} . \tag{1.14}$$

Bude-li mít systém r vstupů a m výstupů ($\boldsymbol{u}(t)$ a $\boldsymbol{y}(t)$ budou vektory), bude platit

$$\mathcal{L}\{\boldsymbol{y}(t)\} = \boldsymbol{F}(p)\mathcal{L}\{\boldsymbol{u}(t)\} , \tag{1.15}$$

$$\boldsymbol{Y}(p) = \boldsymbol{F}(p)\boldsymbol{U}(p) , \tag{1.16}$$

kde $\boldsymbol{F}(p)$ je přenosová matice o rozměrech $(\, m \times r \,)$ a $\boldsymbol{U}(p)$ a $\boldsymbol{Y}(p)$ jsou Laplaceovými obrazy vektorů $\boldsymbol{u}(t)$ a $\boldsymbol{y}(t)$.

Má-li být přenos vyjádřen pomocí dynamických rovnic LTI systému (rovnice (1.5) a (1.6)) je nejprve nutné z těchto rovnic vyeliminovat vektor stavu $\boldsymbol{x}$. Toho lze docílit aplikací $\mathcal{L}$-transformace:

$$p\,\boldsymbol{X}(p) - \boldsymbol{x}(0) = \boldsymbol{A}\,\boldsymbol{X}(p) + \boldsymbol{B}\,\boldsymbol{U}(p) , \tag{1.17}$$

$$\boldsymbol{Y}(p) = \boldsymbol{C}\,\boldsymbol{X}(p) + \boldsymbol{D}\,\boldsymbol{U}(p) . \tag{1.18}$$

Jsou-li uvažovány nulové počáteční podmínky, lze z rovnice (1.17) vyjádřit Laplaceův obraz stavového vektoru $\boldsymbol{X}(p)$:

$$(p\boldsymbol{I} - \boldsymbol{A})\boldsymbol{X}(p) = \boldsymbol{B}\,\boldsymbol{U}(p)\ , \tag{1.19}$$

$$\boldsymbol{X}(p) = (p\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B}\,\boldsymbol{U}(p)\ . \tag{1.20}$$

Po dosazení do výstupní rovnice (1.18) za $\boldsymbol{X}(p)$ a vytknutí $\boldsymbol{U}(p)$ vychází:

$$\boldsymbol{Y}(p) = \left[\boldsymbol{C}\,(p\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B} + \boldsymbol{D}\right]\,\boldsymbol{U}(p)\ . \tag{1.21}$$

Při porovnáním tohoto vztahu s (1.16) je zřejmé, že přenosová matice je rovna

$$\boldsymbol{F}(p) = \boldsymbol{C}\,(p\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B} + \boldsymbol{D}\ , \tag{1.22}$$

Zároveň vychází vztah mezi přenosem (vnější popis) a maticemi dynamických rovnic (vnitřní popis).

Tzv. matici přechodových funkcí (též přechodová matice) lze určit jako

$$\boldsymbol{\Phi}(p) = (p\boldsymbol{I} - \boldsymbol{A})^{-1} = \frac{1}{\det(p\boldsymbol{I} - \boldsymbol{A})} \cdot adj(p\boldsymbol{I} - \boldsymbol{A})\ , \tag{1.23}$$

kde $adj(p\boldsymbol{I} - \boldsymbol{A})$ je adjungovaná matice sestavená z algebraických doplňků příslušné transponované matice a $\det(p\boldsymbol{I} - \boldsymbol{A})$ je determinant matice. [14][16]

1.3.3 Spojování systémů

Dynamický systém bývá často složen ze subsystémů, jejichž dynamické vlastnosti mohou být popsány vnitřním či vnějším popisem. Základními vazbami mezi subsystémy jsou **paralelní**, **sériová** a **zpětná vazba**.

Jsou dány systémy S_1 a S_2. Systém S_1 popsán stavovými rovnicemi

$$\dot{\boldsymbol{x}}_1 = \boldsymbol{A}_1\,\boldsymbol{x}_1 + \boldsymbol{B}_1\,\boldsymbol{u}_1\,,$$
$$\boldsymbol{y}_1 = \boldsymbol{C}_1\,\boldsymbol{x}_1 + \boldsymbol{D}_1\,\boldsymbol{u}_1\ ,$$

nebo přenosovou maticí $\boldsymbol{F}_1(p)$

$$\boldsymbol{Y}_1(p) = \boldsymbol{F}_1(p)\,\boldsymbol{U}_1(p)$$

a systém S_2 popsán stavovými rovnicemi

$$\dot{\boldsymbol{x}}_2 = \boldsymbol{A}_2\,\boldsymbol{x}_2 + \boldsymbol{B}_2\,\boldsymbol{u}_2\,,$$
$$\boldsymbol{y}_2 = \boldsymbol{C}_2\,\boldsymbol{x}_2 + \boldsymbol{D}_2\,\boldsymbol{u}_2\ ,$$

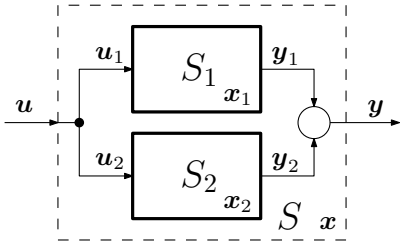
nebo přenosovou maticí $\boldsymbol{F}_2(p)$

$$\boldsymbol{Y}_2(p) = \boldsymbol{F}_2(p)\,\boldsymbol{U}_2(p)\ .$$

Pro řešení stavových rovnic systému S složeného ze dvou subsystémů S_1 a S_2 je zaveden složený stavový vektor

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} . \tag{1.24}$$

1.3.3.1 Paralelní spojení



Obr. 1.3: Paralelní spojení systémů

Aby bylo možné systémy spojit paralelně, musí mít oba systémy stejný počet vstupů ($\dim \boldsymbol{u}_1 = \dim \boldsymbol{u}_2$) a stejný počet výstupů ($\dim \boldsymbol{y}_1 = \dim \boldsymbol{y}_2$).

Rovnice vazeb budou

$$\boldsymbol{u} = \boldsymbol{u}_1 = \boldsymbol{u}_2 , \tag{1.25}$$

$$\boldsymbol{y} = \boldsymbol{y}_1 + \boldsymbol{y}_2 . \tag{1.26}$$

Složený systém S má vnější popis

$$\boldsymbol{Y}(p) = \boldsymbol{F}(p)\,\boldsymbol{U}(p) ,$$

kde $\boldsymbol{Y}(p)$ lze vyjádřit jako

$$\boldsymbol{Y}(p) = \boldsymbol{Y}_1(p) + \boldsymbol{Y}_2(p) = \left[\boldsymbol{F}_1(p) + \boldsymbol{F}_2(p)\right] \boldsymbol{U}(p) . \tag{1.27}$$

Je zřejmé, že přenosová matice $\boldsymbol{F}(p)$ složeného systému S je rovna součtu přenosových matic jednotlivých subsystémů:

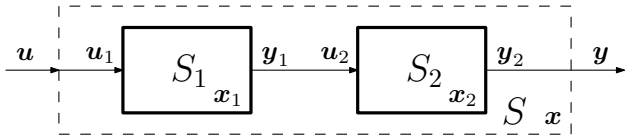
$$\boldsymbol{F}(p) = \boldsymbol{F}_1(p) + \boldsymbol{F}_2(p) . \tag{1.28}$$

Stavové rovnice složeného systému vypadají následujícím způsobem:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{A}_1 & 0 \\ 0 & \boldsymbol{A}_2 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} \boldsymbol{B}_1 \\ \boldsymbol{B}_2 \end{bmatrix} \boldsymbol{u} , \tag{1.29}$$

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{C}_1 & \boldsymbol{C}_2 \end{bmatrix} \boldsymbol{x} + (\boldsymbol{D}_1 + \boldsymbol{D}_2)\,\boldsymbol{u} . \tag{1.30}$$

1.3.3.2 Sériové spojení



Obr. 1.4: Sériové spojení systémů

Aby bylo možné systémy spojit sériově, musí být počet výstupů systému S_1 roven počtu vstupů systému S_2 ($\dim \boldsymbol{y}_1 = \dim \boldsymbol{u}_2$).

Rovnice vazeb budou

$$\boldsymbol{u} = \boldsymbol{u}_1 \ , \tag{1.31}$$

$$\boldsymbol{y}_1 = \boldsymbol{u}_2 \ , \tag{1.32}$$

$$\boldsymbol{y} = \boldsymbol{y}_2 \ . \tag{1.33}$$

Pro přenosovou matici $\boldsymbol{F}(p)$ složeného systému S platí

$$\begin{aligned} \boldsymbol{Y}(p) &= \boldsymbol{F}(p)\,\boldsymbol{U}(p) \ , \\ \boldsymbol{Y}(p) &= \boldsymbol{F}_2(p)\,\boldsymbol{U}_2(p) = \boldsymbol{F}_2(p)\,\boldsymbol{F}_1(p)\,\boldsymbol{U}(p) \ . \end{aligned}$$

Je zřejmé, že přenosová matice $\boldsymbol{F}(p)$ složeného systému S je rovna součinu přenosových matic jednotlivých subsystémů:

$$\boldsymbol{F}(p) = \boldsymbol{F}_2(p)\,\boldsymbol{F}_1(p) \ , \tag{1.34}$$

kde pořadí součinu matic je proti směru šíření signálu (od výstupu ke vstupu). Systémy s jednou vstupní a jednou výstupní veličinou mají přenosové matice $\boldsymbol{F}_1(p)$ a $\boldsymbol{F}_2(p)$ tvořené pouze jedním prvkem a na pořadí součinu potom nezáleží.

Pro vnitřní popisy podsystémů při sériovém spojení platí

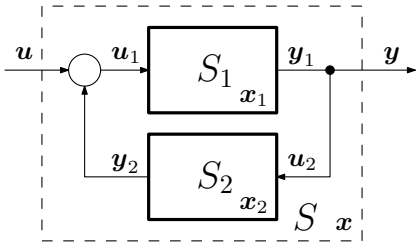
$$\begin{aligned} \dot{\boldsymbol{x}}_1 &= \boldsymbol{A}_1\,\boldsymbol{x}_1 + \boldsymbol{B}_1\,\boldsymbol{u}_1 = \boldsymbol{A}_1\,\boldsymbol{x}_1 + \boldsymbol{B}_1\,\boldsymbol{u} \ , \\ \dot{\boldsymbol{x}}_2 &= \boldsymbol{A}_2\,\boldsymbol{x}_2 + \boldsymbol{B}_2\,\boldsymbol{u}_2 = \boldsymbol{A}_2\,\boldsymbol{x}_2 + \boldsymbol{B}_2\,(\,\boldsymbol{C}_1\,\boldsymbol{x}_1 + \boldsymbol{D}_1\,\boldsymbol{u}) \ , \\ \boldsymbol{y} &= \boldsymbol{y}_2 = \boldsymbol{C}_2\,\boldsymbol{x}_2 + \boldsymbol{D}_2\,\boldsymbol{u}_2 = \boldsymbol{C}_2\,\boldsymbol{x}_2 + \boldsymbol{D}_2\,\boldsymbol{C}_1\,\boldsymbol{x}_1 + \boldsymbol{D}_2\,\boldsymbol{D}_1\boldsymbol{u} \ . \end{aligned}$$

Tyto rovnice lze zapsat maticově s využitím složeného stavového vektoru podle (1.24), čímž vzniknou stavové rovnice složeného systému S

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{A}_1 & 0 \\ \boldsymbol{B}_2\,\boldsymbol{C}_1 & \boldsymbol{A}_2 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} \boldsymbol{B}_1 \\ \boldsymbol{B}_2\,\boldsymbol{D}_1 \end{bmatrix} \boldsymbol{u} \ , \tag{1.35}$$

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{D}_2\boldsymbol{C}_1 & \boldsymbol{C}_2 \end{bmatrix} \boldsymbol{x} + \boldsymbol{D}_2\,\boldsymbol{D}_1\,\boldsymbol{u} \ . \tag{1.36}$$

1.3.3.3 Zpětnovazební spojení



Obr. 1.5: Zpětnovazební spojení systémů

Při zpětnovazebním spojení systémů budou rovnice vazeb budou vypadat následujícím způsobem

$$\boldsymbol{u}_1 = \boldsymbol{u} + \boldsymbol{y}_2 \ , \tag{1.37}$$

$$\boldsymbol{y}_1 = \boldsymbol{u}_2 = \boldsymbol{y} \ . \tag{1.38}$$

Přenosovou matici $\boldsymbol{F}(p)$ lze odvodit následujícím způsobem

$$\boldsymbol{Y}(p) = \boldsymbol{F}_1(p)\,\boldsymbol{U}_1(p) = \boldsymbol{F}_1(p)\,\left[\boldsymbol{U}(p) + \boldsymbol{F}_2(p)\,\boldsymbol{Y}(p)\right] \ ,$$

Odtud

$$\left[\boldsymbol{I} - \boldsymbol{F}_1(p)\,\boldsymbol{F}_2(p)\right]\,\boldsymbol{Y}(p) = \boldsymbol{F}_1(p)\,\boldsymbol{U}(p) \ , \tag{1.39}$$

kde $\boldsymbol{I}$ je jednotková matice. Vyjádřením obrazu výstupu je získán vztah

$$\boldsymbol{Y}(p) = \left[\boldsymbol{I} - \boldsymbol{F}_1(p)\,\boldsymbol{F}_2(p)\right]^{-1}\,\boldsymbol{F}_1(p)\,\boldsymbol{U}(p) \ ,$$

z něhož vyplývá, že přenosová matice je rovna

$$\boldsymbol{F}(p) = \left[\boldsymbol{I} - \boldsymbol{F}_1(p)\,\boldsymbol{F}_2(p)\right]^{-1}\,\boldsymbol{F}_1(p) \ . \tag{1.40}$$

Aby přenosová matice $\boldsymbol{F}(p)$ složeného systému existovala, je nutné, aby matice $(\boldsymbol{I} - \boldsymbol{F}_1\,\boldsymbol{F}_2)$ byla regulární, tedy musí platit

$$\det\left[\boldsymbol{I} - \boldsymbol{F}_1(p)\,\boldsymbol{F}_2(p)\right] \neq 0 \ . \tag{1.41}$$

Vnitřní popis systému lze opět odvodit zavedením stavu $\boldsymbol{x}$ složeného systému podle (1.24). Pro výstup složeného systému S platí

$$\begin{aligned} \boldsymbol{y} = \boldsymbol{y}_1 &= \boldsymbol{C}_1\,\boldsymbol{x}_1 + \boldsymbol{D}_1\,\boldsymbol{u}_1 = \boldsymbol{C}_1\,\boldsymbol{x}_1 + \boldsymbol{D}_1\,\left(\boldsymbol{u} + \boldsymbol{y}_2\right) = \\ &= \boldsymbol{C}_1\,\boldsymbol{x}_1 + \boldsymbol{D}_1\,\boldsymbol{u} + \boldsymbol{D}_1\,\left(\boldsymbol{C}_2\,\boldsymbol{x}_2 + \boldsymbol{D}_2\,\boldsymbol{y}\right) \ . \end{aligned}$$

Převedením všech členů s y na levou stranu vychází

$$\left[\boldsymbol{I} - \boldsymbol{D}_1\,\boldsymbol{D}_2\right]\,\boldsymbol{y} = \boldsymbol{C}_1\,\boldsymbol{x}_1 + \boldsymbol{D}_1\,\boldsymbol{C}_2\,\boldsymbol{x}_2 + \boldsymbol{D}_1\,\boldsymbol{u} \ .$$

Odtud

$$\boldsymbol{y} = \left[\boldsymbol{I} - \boldsymbol{D}_1\,\boldsymbol{D}_2\right]^{-1}\left(\boldsymbol{C}_1\,\boldsymbol{x}_1 + \boldsymbol{D}_1\,\boldsymbol{C}_2\,\boldsymbol{x}_2 + \boldsymbol{D}_1\,\boldsymbol{u}\right)\ ,$$

což lze pomocí stavu $\boldsymbol{x}$ složeného systému (1.24) zapsat maticovém tvaru:

$$\boldsymbol{y} = \left[\ \left(\boldsymbol{I} - \boldsymbol{D}_1\,\boldsymbol{D}_2\right)^{-1}\boldsymbol{C}_1,\ \ \left(\boldsymbol{I} - \boldsymbol{D}_1\,\boldsymbol{D}_2\right)^{-1}\boldsymbol{D}_1\,\boldsymbol{C}_2\ \right]\boldsymbol{x} + \left(\boldsymbol{I} - \boldsymbol{D}_1\,\boldsymbol{D}_2\right)^{-1}\boldsymbol{D}_1\,\boldsymbol{u}\ . \quad (1.42)$$

Pro substavy $\boldsymbol{x}_1$ a $\boldsymbol{x}_2$ složeného systému zde platí

$$\begin{aligned}\dot{\boldsymbol{x}}_1 &= \boldsymbol{A}_1\,\boldsymbol{x}_1 + \boldsymbol{B}_1\,\boldsymbol{u}_1 = \boldsymbol{A}_1\,\boldsymbol{x}_1 + \boldsymbol{B}_1\,\left(\boldsymbol{u} + \boldsymbol{y}_2\right) = \boldsymbol{A}_1\,\boldsymbol{x}_1 + \boldsymbol{B}_1\,\boldsymbol{u} + \boldsymbol{B}_1\,\left(\boldsymbol{C}_2\,\boldsymbol{x}_2 + \boldsymbol{D}_2\,\boldsymbol{y}\right)\ ,\\ \dot{\boldsymbol{x}}_2 &= \boldsymbol{A}_2\,\boldsymbol{x}_2 + \boldsymbol{B}_2\,\boldsymbol{u}_2 = \boldsymbol{A}_2\,\boldsymbol{x}_2 + \boldsymbol{B}_2\,\boldsymbol{y}\ .\end{aligned}$$

Dosazením vztahu (1.42) za výstup y do obou předchozích vztahů lze sestavit zbývající stavovou rovnici

$$\begin{aligned}\dot{\boldsymbol{x}} &= \begin{bmatrix} \boldsymbol{A}_1 + \boldsymbol{B}_1\,\boldsymbol{D}_2\,\left(\boldsymbol{I} - \boldsymbol{D}_1\,\boldsymbol{D}_2\right)^{-1}\boldsymbol{C}_1 & \boldsymbol{B}_1\,\boldsymbol{C}_2 + \boldsymbol{B}_1\,\boldsymbol{D}_2\,\left(\boldsymbol{I} - \boldsymbol{D}_1\,\boldsymbol{D}_2\right)^{-1}\boldsymbol{D}_1\boldsymbol{C}_2 \\ \boldsymbol{B}_2\,\left(\boldsymbol{I} - \boldsymbol{D}_1\,\boldsymbol{D}_2\right)^{-1}\boldsymbol{C}_1 & \boldsymbol{A}_2 + \boldsymbol{B}_2\,\left(\boldsymbol{I} - \boldsymbol{D}_1\,\boldsymbol{D}_2\right)^{-1}\boldsymbol{D}_1\boldsymbol{C}_2 \end{bmatrix}\boldsymbol{x} +\\ &+ \begin{bmatrix} \boldsymbol{B}_1 + \boldsymbol{B}_1\,\boldsymbol{D}_2\,\left(\boldsymbol{I} - \boldsymbol{D}_1\,\boldsymbol{D}_2\right)^{-1}\boldsymbol{D}_1 \\ \boldsymbol{B}_2\,\left(\boldsymbol{I} - \boldsymbol{D}_1\,\boldsymbol{D}_2\right)^{-1}\boldsymbol{D}_1 \end{bmatrix}\boldsymbol{u}\ . \end{aligned} \quad (1.43)$$

Za předpokladu, že platí podmínka podle vztahu (1.41), vyplývají ze vztahů (1.42) a (1.43) matice $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ a $\boldsymbol{D}$ složeného systému. [14]

V rámci balíčku DynSyPy nejsou tyto vazby použity, jelikož jejich užití je možné pouze v případě, že spojované systémy jsou lineární. Tento způsob realizace systémových vazeb zároveň znemožňuje práci s multi-rate systémy.

Způsob použitý v rámci balíčku DynSyPy je popsán v sekci 3.2.1. Umožňuje spojení různých systémů s různou vzorovací frekvencí. K synchronizaci multi-rate systémů byla do balíčku implementována třída Pool, viz sekce 3.1.

Kapitola 2

Existující moduly v jazyce Python pro práci s dynamickými systémy

2.1 SciPy

Knihovna SciPy je jedním ze základních balíčků, které tvoří zásobník SciPy. Poskytuje mnoho uživatelsky přívětivých a efektivních numerických metod, například metody pro numerickou integraci, interpolaci, optimalizaci, lineární algebru a statistiku.

Dynamickým systémům se v balíčku SciPy věnuje submodul `scipy.signal`. Ten obsahuje třídy, jejichž objekty představují LTI systémy, které mohou být spojité i nespojité v čase (diskrétní systémy). LTI systémy je možné definovat několika způsoby. Buď ve formě přenosové funkce, nebo ve formě tzv. nul, pólů a zesílení, nebo ve formě stavového prostoru.

LTI systémy je možné pomocí dostupných metod simulovat a analyzovat. Jsou zde obsaženy metody pro simulaci výstupů lineárních systémů a simulaci jejich odezvy na skok či impuls. Je zde také možné provedení analýzy ve frekvenční oblasti, konkrétně výpočet hodnot pro vynesení amplitudové a fázové frekvenční charakteristiky a také pro vynesení tzv. Nyquistovy charakteristiky. [17]

2.2 SimuPy

SimuPy je rámec pro simulaci propojených modelů dynamických systémů a poskytuje open source nástroj založený na jazyce python, který lze použít v pracovních postupech návrhu a simulace založených na modelech a systémech. Modely dynamických systémů lze zadat jako objekt s rozhraním popsaným v dokumentaci API (viz [10]). Modely lze také konstruovat pomocí symbolických výrazů. [6]

2.3 PyDSTool

PyDSTool je integrovaný balíček pro simulaci, modelování a analýzu dynamických systémů. Je nezávislý na platformě, napsaný převážně v jazyce Python. Pro některé výpočetně náročné úlohy lze vyvolat kód napsaný v jazycích C a Fortran (bez zásahu nebo přispění uživatele na této úrovni).

Ve velké míře jsou zde využívány knihovny numpy a scipy. PyDSTool podporuje symbolickou matematiku, optimalizaci, analýzu fázové roviny, analýzu kontinuity a bifurkace, analýzu dat a další nástroje pro modelování – zejména pro biologické aplikace.

Je zde kladen důraz na podporu analýzy dat a přizpůsobování modelů, což je klíčová součást procesu modelování založeného na datech. Balíček je zaměřen na modely zahrnující obyčejné diferenciální rovnice (dále ODE – Ordinary Differential Equation), diferenciálně-algebraické rovnice (dále DAE – Differential-Algebraic Equation) a diskrétní mapy. [3]

2.4 S-timator

S-timator je knihovna v jazyce Python, sloužící k analýze modelů popsaných pomocí obyčejných diferenciálních rovnic, jež jsou také známy jako dynamické nebo kinetické modely.

S-timator je vybaven funkcemi pro řešení obyčejných diferenciálních rovnic, což umožňuje základní analýzu dynamického systému. Dále jsou zde dostupné nástroje pro odhad parametrů a výběr modelu. Při zadání experimentálních dat ve formě časových řad a při omezení pracovních rozsahů modelu mohou vestavěné numerické optimalizátory najít hodnoty parametrů a pomoci při experimentálním návrhu s výběrem modelu.

V rámci tohoto balíčku je možné modely vkládat jako prostý text podle pravidel velmi jednoduchého, člověkem čitelného jazyka. [4]

2.5 Pynamical

Pynamical je výukový balíček v jazyce Python, který slouží k modelování, simulaci, vizualizaci a animaci diskrétních nelineárních dynamických systémů a chaosu se zaměřením na jednorozměrné mapy (například logistickou mapu a kubickou mapu). Pro svou činnost využívá balíčky pandas, numpy, numba a matplotlib.

Součástí balíčku jsou předdefinované logistické mapy (logistické zobrazení), Singerova mapa a kubická mapa. Modely lze spouštět s různými hodnotami parametrů v řadě časových kroků. Výsledný numerický výstup je vrácen jako pandas DataFrame, což je rychlý a efektivní objekt pro manipulaci s daty s integrovaným indexováním. Pandas je open source knihovna s licencí BSD, která poskytuje vysoce výkonné a snadno použitelné datové struktury a nástroje pro analýzu dat v programovacím jazyce Python. [15]

Tento výstup je možné pomocí Pynamical různými způsoby vizualizovat, např. pomocí bifurkačních diagramů, dvourozměrných fázových diagramů, trojrozměrných fázových diagramů a pavučinových grafů. Tyto vizualizace umožňují jednoduché kvalitativní posouzení chování systému včetně fázových přechodů, bifurkačních bodů, atraktorů a limitních cyklů, oblastí přitažlivosti a fraktálů.

Pynamical usnadňuje definování diskrétních jednorozměrných nelineárních modelů jako funkcí jazyka Python s just-in-time kompilací, což vede ke zrychlení simulace. [1][2]

2.6 Knihovny PySD a BPTK_Py

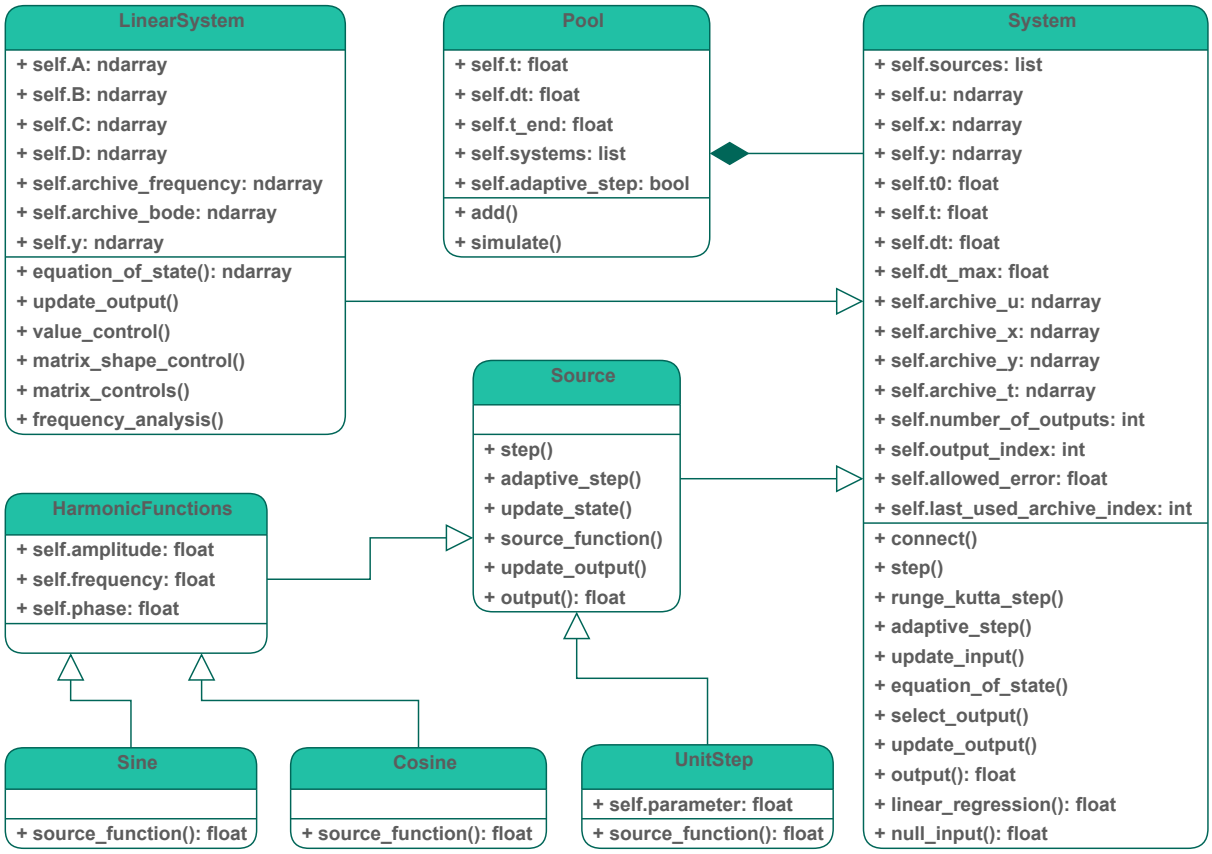
V rámci této rešerše byly nalezeny také knihovny PySD a BPTK_Py. Nebyly k nim však nalezeny dostatečné informace, proto zde o nich bude pouze stručná zmínka.

PySD je jednoduchá knihovna pro spouštění modelů systémové dynamiky v jazyce python, jejímž cílem je zlepšit integraci Big Data a strojového učení do pracovních postupů systémové dynamiky. Umí překládat soubory modelu Vensim nebo XMILE do modulů pythonu a poskytuje metody pro úpravu, simulaci a sledování těchto přeložených modelů. [8]

Knihovna BPTK_Py představuje jednoduchý rámec pro modelování založené na systémové dynamice a také modelování založené na agentech (tzv. ABM – Agent-Based Modeling). [7]

Kapitola 3

Balíček DynSyPy



Obr. 3.1: Diagram tříd (UML) balíčku DynSyPy

Z obrázku 3.1 je patrné, že balíček DynSyPy je tvořen třídou System a jejími dceřinnými třídami (potomky) a třídou Pool.

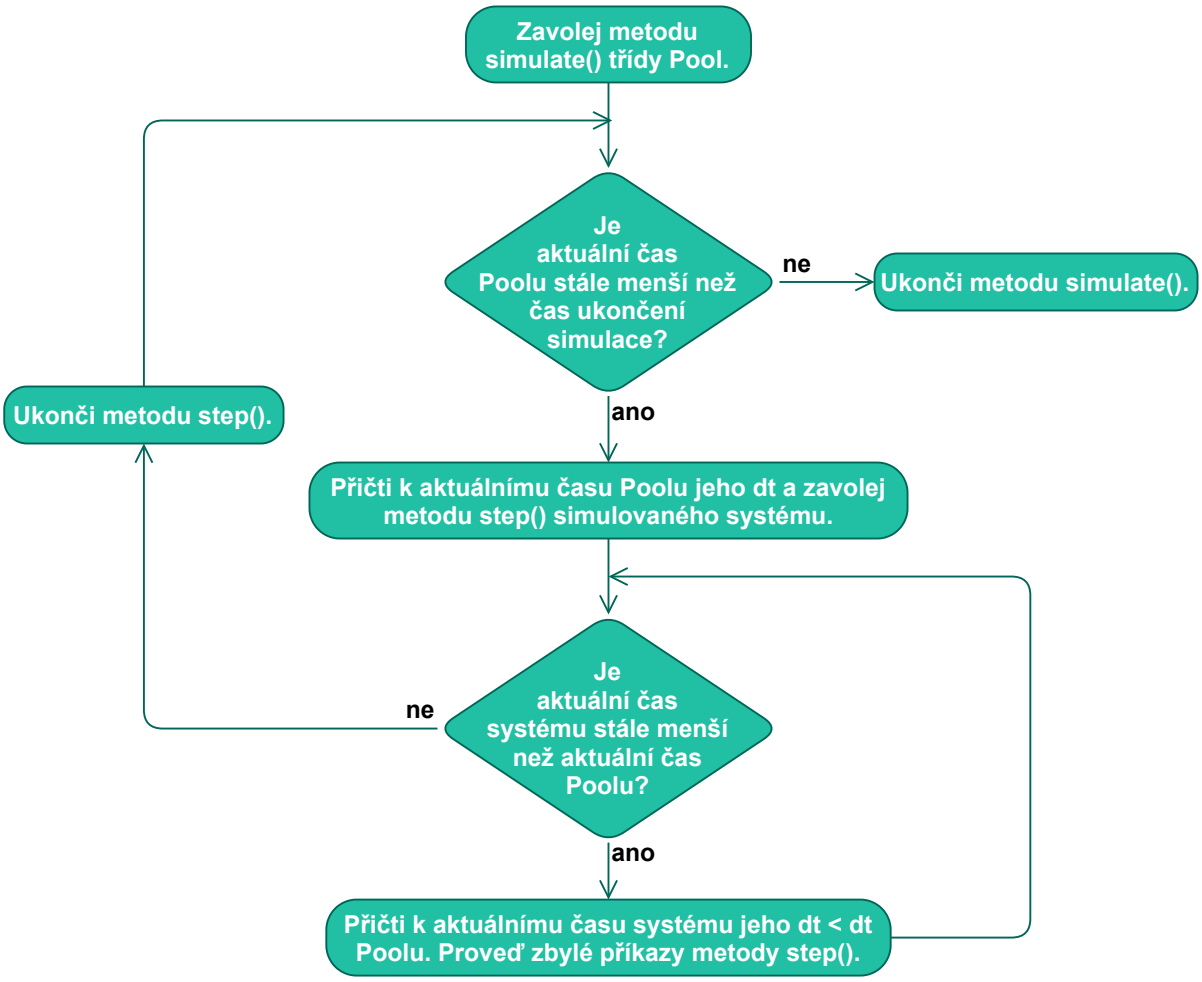
Rodičovská třída System definuje obecný systém, resp. atributy a metody, které jsou pro všechny dynamické systémy společné. Jejími potomky jsou třídy LinearSystem a Source. Třída LinearSystem definuje tzv. lineární dynamický systém a třída Source slouží k definici zdrojů. Od třídy Source dále dědí další třídy, jež představují konkrétní zdroje (např. harmonické funkce nebo jednotkový skok). V rámci dalšího vývoje bude

balíček pravděpodobně rozšířen např. o třídu po vytváření nelineárních dynamických systémů, což ovšem není předmětem této bakalářské práce.

Poslední třídou v tomto balíčku je třída Pool, jejíž atributy a metody slouží k nastavení a realizaci simulace systému (popř. systémů).

Zbytek této kapitoly je věnován bližšímu popisu těchto tříd a jejich atributů a metod.

3.1 Třída Pool



Obr. 3.2: Znázornění synchronizace systémů pomocí třídy Pool

Aby bylo možné současně simulovat více systémů, z nichž každý by pro simulaci využíval metody s adaptivním krokem, je potřeba vytvořit prostředí, které zajistí, že budou jednotlivé systémy v průběhu simulace synchronizovány. Tento problém je v rámci balíčku DynSyPy řešen pomocí třídy Pool.

Instanci této třídy je při vytváření nastaven časový krok `self.dt` a hodnota `self.t_end`, což je čas, kdy má být simulace ukončena. Velikost tohoto časového kroku je vhodné nastavovat na větší hodnotu než je maximální velikost časového kroku simulovaných systémů. Násobky hodnoty `self.dt` představují okamžiky, ve kterých jsou časy

simulovaných systémů synchronizovány. Jednotlivé systémy tedy v rámci jednoho kroku instance třídy Pool začínají i končí ve stejném čase. Proces této synchronizace je znázorněn na obr. 3.2.

Simulace systémů je zajištěna pomocí metody `simulate()`. Systémy, které mají být touto metodou simulovány, je nutné nejprve uložit do seznamu `self.systems`. K tomuto účelu slouží metoda `add()`. Obě tyto metody jsou podrobněji popsány níže.

Konstruktor třídy Pool má dva povinné a dva nepovinné parametry. Nepovinný parametr má přednastavenou hodnotu, tudíž není-li při volání metody (popř. funkce) tento parametr vyplněn, zůstane mu jeho přednastavená hodnota. Jedná se o následující parametry:

- `dt`
  - požadovaná velikost časového kroku systému v sekundách,
- `t_end`
  - hodnota času, kdy má být simulace ukončena, v sekundách,
- `t0=0`
  - čas počátku simulace v sekundách,
- `adaptive_step=True`
  - hodnota typu `bool`,
  - na základě této hodnoty je v metodě `simulate()` v sekci 3.1.2 rozhodováno, zda budou systémy modelovány s pevným nebo s adaptivním krokem.

V konstruktoru jsou hodnoty těchto parametrů uloženy do proměnných a také je vytvořen prázdný seznam určený pro ukládání systémů určených k simulaci:

```
self.t = t0
self.dt = dt
self.t_end = t_end

self.systems = []

self.adaptive_step = adaptive_step
```

Tato třída slouží pouze pro simulaci systémů a jejich synchronizaci. Kdyby byly všechny systémy simulovány se stejným krokem, byla by tato třída pravděpodobně zbytečná. Implementací této třídy je však umožněno užití metod s adaptivním krokem, což vede ke zpřesnění simulací a k jejich výraznému zrychlení.

3.1.1 Metoda `add()`

Metoda `add()` má za úkol při zavolání přidat do seznamu `self.systems` systém, jenž je metodě předán jako parametr. V tomto seznamu jsou uloženy systémy, které mají být simulovány synchronně. Tento seznam je následně procházen metodou `simulate()`.

Při vytváření objektu třídy Pool se seznam `self.systems` pouze vytvoří, ale žádný systém v něm není. Z tohoto důvodu, pokud není před samotnou simulací metoda `add()` zavolána alespoň jednou, k simulaci vůbec nedojde.

3.1.2 Metoda `simulate()`

Tato metoda zajišťuje provedení samotné simulace. Při každém kroku pomocí cyklu `for()` projde seznam systémů `self.systems` a pro každý z nich zavolá jeho metodu `step()` případně `adaptive_step()` a v parametru jí předá čas, kdy se má program vrátit ze systému zpět do metody `simulate()`.

Rozhodování o tom, zda bude u simulovaných systémů zavolána metoda `step()` nebo `adaptive_step()`, je prováděno na základě hodnoty proměnné `self.adaptive_step`. Pokud má hodnotu `True`, je zavolána metoda `adaptive_step()`, jinak metoda `step()`.

3.2 Třída System

Třída System je základním kamenem pro vytváření dynamických systémů. Jedná se o abstraktní třídu (nelze vytvářet instance této třídy) a obsahuje metody a proměnné, které buď přímo používají nebo si podle svých požadavků modifikují její dceřinné třídy (zatím jsou to třídy LinearSystem a Source). Jedná se např. o metodu `connect()` pro „připojení“ zdroje na vstup systému. Dále důležité metody `step()` a její „sesterská“ metoda `adaptive_step()`, které zajišťují aktualizaci a archivaci hodnot času, vstupu, stavu a výstupu. Rozdíl mezi těmito dvěma metodami je velikost časového kroku. U metody `step()` je tento krok konstantní, metoda `adaptive_step()` si krok upravuje podle toho, jak prudce se systém mění v čase, což vede k zrychlení i zpřesnění simulace.

Konstruktor třídy System nemá žádný povinný parametr a sedm nepovinných parametrů:

- `dt0=1.5e-5`
  - požadovaná velikost časového kroku systému v sekundách při použití metody `step()`,
  - při použití metody `adaptive_step()` tento krok nemusí být vůbec použit,
- `t0=0`
  - čas počátku simulace v sekundách,

- `x0=0`
  - vektor počátečních podmínek stavových veličin v základních jednotkách těchto veličin,
  - tato proměnná je typu `numpy.ndarray`, což je datový typ, který umožňuje maticové a vektorové operace (dále již jen `ndarray`),
  - tento parametr musí být vyplněn, má-li systém více než jednu stavovou veličinu, nebo je-li stavová veličina v počátku nenulová,
- `number_of_inputs=1`
  - informace o počtu vstupů (představuje počet řádků vektoru vstupu $\boldsymbol{u}$),
  - tento parametr musí být vyplněn, má-li systém více než jeden vstup,
- `number_of_outputs=1`
  - informace o počtu výstupů (představuje počet řádků vektoru výstupu $\boldsymbol{y}$),
  - tento parametr musí být vyplněn, má-li systém více než jeden výstup,
- `allowed_error=1e-6`
  - udává maximální velikost chyby integrace při použití adaptivní integrace,
- `dt_max=1e-2`
  - udává maximální krok, který může být použit při použití adaptivní integrace.

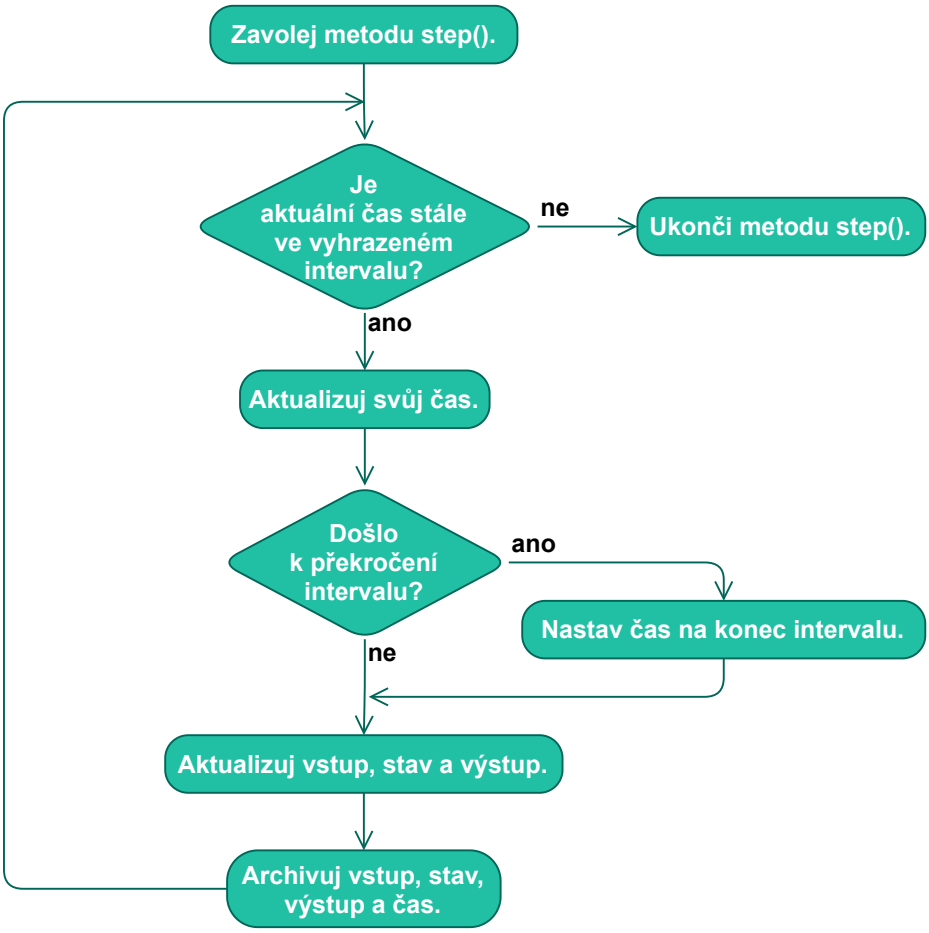
V rámci této práce byl balíček tvořen tak, aby byl schopen modelovat lineární dynamické systémy. Je tedy velmi pravděpodobné, že po přidání třídy pro vytváření nelineárních dynamických systémů bude nutné třídu System modifikovat nebo rozšířit.

3.2.1 Metoda `connect()`

Metoda `connect()` je metodou třídy System a umožňuje k systému „připojit“ zdroj nebo zdroje. V parametru je metodě předán zdroj a jeho pozice ve vektoru zdrojů. Metoda nejprve „zkusí“ zdroj přidat na zadanou pozici v seznamu `self.source`. Pokud tato pozice v seznamu není (seznam je kratší), je zdroj „přilepen“ na konec seznamu a seznam se tím zvětší.

Připojování zdrojů je nutné provádět v pořadí, ve kterém mají být umístěny ve vektoru zdrojů, neboť ve chvíli, kdy je zdroj přidáván na konec seznamu (požadovaná pozice v seznamu neexistuje), již není přidáván na pozici předanou v parametru, ale pouze na konec seznamu. Tato pozice ovšem nemusí být shodná s pozicí požadovanou a může tímto způsobem dojít k záměně pozic jednotlivých zdrojů.

3.2.2 Metoda `step()`



Obr. 3.3: Diagram činnosti metody `step()`

Metoda `step()` po zavolání provede n integračních kroků podle toho, jaký krok má systém nastaven a podle toho, jak velký krok má nastavena instance třídy Pool, jejíž metoda `simulate()` volá metodu `step()`. V každém integračním kroku metoda aktualizuje čas, vstup, stav a výstup.

Čas je aktualizován přičtením hodnoty `self.dt` ke stávající hodnotě `self.t`. Hodnota `self.dt` je při použití této metody konstantní. Pouze v případě, že by po přičtení posledního kroku intervalu byl interval překročen, se nastaví čas na konec intervalu a krok je v tuto chvíli menší. Proměnná `self.dt` však zůstává nezměněna.

Aktualizaci vstupu, stavu a výstupu zajišťují následující metody:

- `update_input()`,
- `runge_kutta_step()`,
- `update_output()`.

Jejich činnost je podrobněji popsána níže.

Následně jsou aktualizované hodnoty uloženy do proměnných typu `ndarray`. Jedná se o proměnné:

- `self.archive_x`,
- `self.archive_y`,
- `self.archive_u`,
- `self.archive_t`.

Tyto „matice“ potom lze využít pro vykreslení grafů.

3.2.3 Metoda `runge_kutta_step()`

Tato metoda slouží jako integrátor s konstantním krokem. Její princip je založen na metodách numerické integrace, konkrétně používá tzv. klasickou metodu Runge-Kutta, která také bývá označována jako RK4, neboli metoda Runge-Kutta 4. řádu.

Stav systému v následujícím kroku je pomocí současného stavu popsán pomocí následující rovnice:

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \frac{1}{6}h\left(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4\right) . \tag{3.1}$$

Každý krok vyžaduje použití následujících čtyř hodnot (obecně se jedná o vektory):

$$\begin{aligned} \boldsymbol{k}_1 &= f\left(t_i,\ \boldsymbol{x}_i\right) , \\ \boldsymbol{k}_2 &= f\left(t_i + \frac{1}{2}h,\ \boldsymbol{x}_i + \frac{1}{2}h\boldsymbol{k}_1\right) , \\ \boldsymbol{k}_3 &= f\left(t_i + \frac{1}{2}h,\ \boldsymbol{x}_i + \frac{1}{2}h\boldsymbol{k}_2\right) , \\ \boldsymbol{k}_4 &= f\left(t_i + h,\ \boldsymbol{x}_i + h\boldsymbol{k}_3\right) . \end{aligned} \tag{3.2}$$

Tato metoda je zde použita za účelem aktualizace stavu systému, popsaného diferenciální rovnicí, popř. soustavou diferenciálních rovnic. Systémy popsané pomocí funkce času (v rámci tohoto balíčku se jedná o zdroje – objekty potomků třídy Source) používají pro aktualizaci stavu metodu `update_state()`, která neprovádí integraci, ale přímo zavolá funkci času popisující daný systém.

3.2.4 Metoda `adaptive_step()`

Metoda `adaptive_step()` funguje stejně jako metoda `step()`. Také nejprve provede aktualizaci času, vstupu, stavu a výstupu a následně tyto hodnoty archivuje. Na rozdíl od metody `step()` je však integrátor zabudován v ní. Jedná se o integrátor s adaptivním krokem. Kód integrátoru je upravený kód metody `rkf()` z modulu diffeq.py dostupného na webu (viz [11]).

Tento integrátor je také založen na principu numerické integrace, konkrétně je zde použita integrační metoda Runge-Kutta-Fehlberg (zkráceně RKF). Ta zjišťuje, zda je používána vhodná velikost kroku. V každém kroku jsou pro řešení vytvořeny a porovnány dvě různé aproximace. Pokud jsou výsledky obou aproximací shodné, nebo je jejich rozdíl v toleranci, je krok buď zvětšen, nebo je použit stávající krok. Metoda se snaží použít největší možný krok takový, aby výsledek byl stále v toleranci. Pokud rozdíl výsledků aproximací není v toleranci, je vypočítán nový menší krok.

Každý krok vyžaduje použití následujících šesti hodnot:

$$\begin{aligned}
\boldsymbol{k}_1 &= hf\left(t_i,\ \boldsymbol{x}_i\right)\ , \\
\boldsymbol{k}_2 &= hf\left(t_i + \frac{1}{4}h,\ \boldsymbol{x}_i + \frac{1}{4}\boldsymbol{k}_1\right)\ , \\
\boldsymbol{k}_3 &= hf\left(t_i + \frac{3}{8}h,\ \boldsymbol{x}_i + \frac{3}{32}\boldsymbol{k}_1 + \frac{9}{32}\boldsymbol{k}_2\right)\ , \\
\boldsymbol{k}_4 &= hf\left(t_i + \frac{12}{13}h,\ \boldsymbol{x}_i + \frac{1932}{2197}\boldsymbol{k}_1 - \frac{7200}{2197}\boldsymbol{k}_2 + \frac{7296}{2197}\boldsymbol{k}_3\right)\ , \\
\boldsymbol{k}_5 &= hf\left(t_i + h,\ \boldsymbol{x}_i + \frac{439}{216}\boldsymbol{k}_1 - 8\boldsymbol{k}_2 + \frac{3680}{513}\boldsymbol{k}_3 - \frac{845}{4104}\boldsymbol{k}_4\right)\ , \\
\boldsymbol{k}_6 &= hf\left(t_i + \frac{1}{2}h,\ \boldsymbol{x}_i - \frac{8}{27}\boldsymbol{k}_1 + 2\boldsymbol{k}_2 - \frac{3544}{2565}\boldsymbol{k}_3 + \frac{1859}{4104}\boldsymbol{k}_4 - \frac{11}{40}\boldsymbol{k}_5\right)\ .
\end{aligned} \tag{3.3}$$

Aproximace řešení počáteční úlohy je řešena metodou Runge-Kutta řádu 4:

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \frac{25}{216}\boldsymbol{k}_1 + \frac{1408}{2565}\boldsymbol{k}_3 + \frac{2197}{4101}\boldsymbol{k}_4 - \frac{1}{5}\boldsymbol{k}_5\ . \tag{3.4}$$

Aby bylo možné zjistit chybu aproximace, je hodnota řešení počítána také metodou Runge-Kutta řádu 5:

$$\boldsymbol{z}_{i+1} = \boldsymbol{x}_i + \frac{16}{135}\boldsymbol{k}_1 + \frac{6656}{12,825}\boldsymbol{k}_3 + \frac{28,561}{56,430}\boldsymbol{k}_4 - \frac{9}{50}\boldsymbol{k}_5 + \frac{2}{55}\boldsymbol{k}_6\ . \tag{3.5}$$

Rozdíl výsledků aproximace pomocí metod 4. a 5. řádu je výsledná chyba, jíž lze využít pro výpočet nového kroku.

Optimální velikost kroku lze určit vynásobením hodnoty s aktuální velikostí kroku h. Hodnotu s lze určit následujícím způsobem:

$$s = \left(\frac{\mathrm{tol}\,h}{2\left\|\,|\boldsymbol{z}_{i+1} - \boldsymbol{x}_{i+1}|\,\right\|_\infty}\right)^{\frac{1}{4}} \approx 0,84\left(\frac{\mathrm{tol}\,h}{\left\|\,|\boldsymbol{z}_{i+1} - \boldsymbol{x}_{i+1}|\,\right\|_\infty}\right)^{\frac{1}{4}}\ . \tag{3.6}$$

Výpočet optimálního kroku je potom realizován následujícím způsobem:

$$h = h \cdot 0.84\left(\frac{\mathrm{tol}\,h}{\left\|\,|\boldsymbol{z}_{i+1} - \boldsymbol{x}_{i+1}|\,\right\|_\infty}\right)^{\frac{1}{4}}\ , \tag{3.7}$$

kde hodnota tol představuje akceptovanou velikost chyby aproximace.[12]

3.2.5 Metoda `update_input()`

Tato metoda se stará o aktualizaci vstupu (zdrojů). Při každém zavolání metoda pomocí cyklu `for()` projde seznam připojených zdrojů `self.source`.

Tento seznam obsahuje funkce nikoli číselné hodnoty. Každá funkce je postupně zavolána a v parametru je jí předána hodnota aktuálního času `self.t`. Hodnota, kterou funkce vrátí, je uložena do pomocného seznamu `source_t` (hodnoty zdrojů v čase `self.t`). Tento pomocný seznam je v posledním kroku přiřazen do proměnné `self.u`. Jelikož tato proměnná je typu `ndarray`, je před přiřazením seznam přetypován na `ndarray` a následně transponován. Tím vznikne sloupcový vektor (vektor zdrojů), se kterým již lze provést simulaci.

3.2.6 Metoda `select_output()`

Tato metoda má uplatnění, pokud je modelován systém, na jehož vstup je potřeba připojit jiný systém, který má více než jeden výstup.

Metodě je v parametru (proměnná `index`) předána pozice požadovaného výstupu a ta je následně přiřazena do proměnné `self.output_index` (její hodnota je při vytváření objektu nastavena na 0).

Následně metoda kontroluje, jestli zadaná pozice není mimo rozsah seznamu výstupů. Pokud ano, dojde k výjimce.

3.2.7 Metoda `output()`

Metoda `output()` má za úkol vrátit hodnotu výstupu v požadovaném čase, který byl zvolen pomocí metody `select_output()`. Hodnota času je jí předána v parametru jako proměnná `t`.

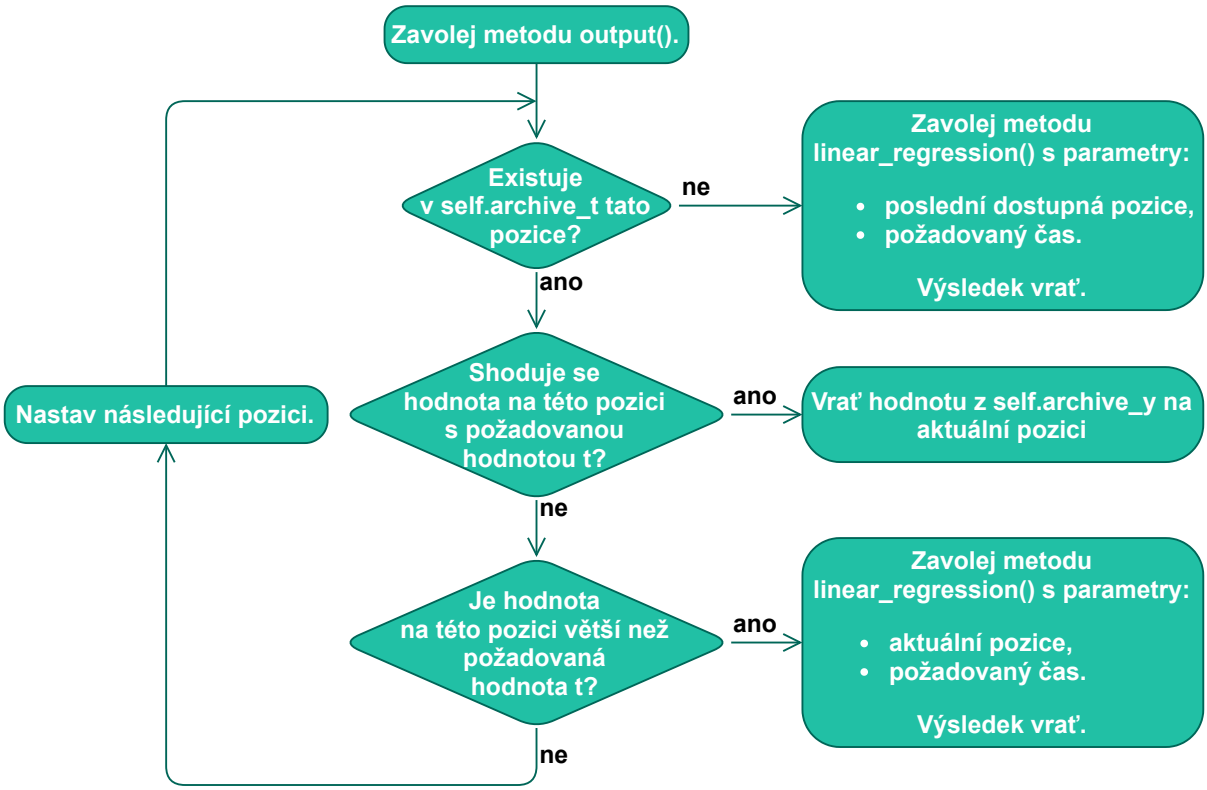
V průběhu simulace jsou hodnoty výstupů průběžně ukládány do archivu výstupu systému `self.archive_y` a hodnoty času do archivu `self.archive_t`. Je zde využita skutečnost, že hodnota výstupu je do archivu `self.archive_y` vždy uložena na stejnou pozici, jako jí odpovídající hodnota času do archivu `self.archive_t` (jejich indexy jsou stejné).

Metoda pomocí cyklu `for()` prochází archiv `self.archive_t` a porovnává hodnoty v něm s hodnotou předanou v parametru. Při shodě se uloží pozice času v archivu do proměnné `self.last_used_archive_index`, aby se příště nemusel archiv procházet od začátku, a poté metoda rovnou vrátí hodnotu výstupu na této pozici v archivu `self.archive_y`.

Pokud shodná hodnota v archivu neexistuje, hledá metoda první hodnotu, která je větší než požadovaný čas. Toto je situace, kdy je požadovaná hodnota mezi hodnotou na současné pozici v archivu a hodnotou na pozici předchozí. Pomocí metody `linear_regression()` je z hodnot na této a předchozí pozici požadovaná hodnota

dopočítána. Před zavoláním metody `linear_regression()` je ovšem ještě uložena předchozí pozice do proměnné `self.last_used_archive_index`.

Činnost této metody je znázorněna na obr. 3.4.



Obr. 3.4: Diagram činnosti metody `output()`

3.2.8 Metoda `linear_regression()`

Metodě je v parametru předán čas, ve kterém je zjišťována hodnota výstupu, a pozice první hodnoty z archivu `self.archive_t`, jejíž hodnota je větší než požadovaný čas.

Nejprve jsou metodou vypočítány hodnoty k a q, což jsou prvky tzv. směrnicové rovnice přímky

$$y(x) = k\ x + q\,, \tag{3.8}$$

kde

k je směrnice přímky,

q je posun přímky ve směru osy y.

Základem pro výpočet těchto hodnot je numerický výpočet soustavy lineárních rovnic (v tomto případě soustava dvou lineárních rovnic o dvou neznámých) v maticovém tvaru:

$$\boldsymbol{A}\ \boldsymbol{x} = \boldsymbol{b}\,, \tag{3.9}$$

kde

$\boldsymbol{A}$ představuje tzv. systémovou matici (v tomto případě matice 2×2),

$\boldsymbol{x}$ je vektor neznámých (sloupcový vektor v tomto případě o dvou řádcích),

$\boldsymbol{b}$ je tzv. vektor pravé strany (sloupcový vektor v tomto případě o dvou řádcích).

Pro výpočet neznámých je rovnice upravena do následujícího tvaru:

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}\,. \tag{3.10}$$

Hodnoty v těchto maticích a vektorech jsou uspořádány následujícím způsobem:

$$\boldsymbol{x} = \begin{pmatrix} k \\ q \end{pmatrix}, \tag{3.11}$$

$$\boldsymbol{A} = \begin{pmatrix} \texttt{self.archive\_t[i-1]} & 1 \\ \texttt{self.archive\_t[i]} & 1 \end{pmatrix}, \tag{3.12}$$

$$\boldsymbol{b} = \begin{pmatrix} \texttt{self.archive\_y[self.output\_index][i-1]} \\ \texttt{self.archive\_y[self.output\_index][i]} \end{pmatrix}. \tag{3.13}$$

Dosazením systémové matice $\boldsymbol{A}$ (3.12) a vektoru pravých stran $\boldsymbol{b}$ (3.13) do rovnice (3.10) je vypočítán vektor neznámých $\boldsymbol{x}$. V kódu je to realizováno následujícím způsobem:

```
vector_x = np.linalg.inv(system_matrix) @ vector_of_right_sides
```

Vypočítané prvky vektoru $\boldsymbol{x}$ je nyní možné dosadit do směrnicové rovnice přímky (3.8). V kódu je toto řešeno následujícím způsobem:

```
y = vector_x[0] * t + vector_x[1]
```

Hodnota `y` je návratová hodnota a jedná se o přibližnou hodnotu výstupu v čase `t`.

3.2.9 Metoda `null_input()`

Statická metoda null_input() vždy vrací hodnotu `0.0`. Je volána v kostruktoru při vytváření objektu, kdy má za úkol do seznamu `self.sources` uložit na pozici `0` hodnotu `0.0`.

Při vytváření objektu ještě není k systému připojen žádný zdroj, protože metodu `connect()` pro daný objekt (systém) je možné zavolat až poté, co je objekt (systém) vytvořen. Seznam `self.sources` by byl tedy prázdný. Jelikož se pomocí tohoto seznamu nastavují další atributy, nesmí být tento seznam prázdný, což zajišťuje právě metoda `null_input()`.

3.2.10 Metody `equation_of_state()` a `update_output()`

Tyto dvě metody jsou ve třídě System pouze definovány, ale nic nedělají. Jejich chování si dceřinné třídy upraví (tzv. overriding – překrytí metody rodiče).

Metoda `equation_of_state()` slouží k popisu stavu systému. Jelikož stav je u různých druhů systémů popisován různými způsoby, je potřeba, aby si každý systém tuto metodu upravil. Tato metoda ovšem není abstraktní, tudíž není bezpodmínečně nutné, aby ji každý potomek přepisoval. Důvodem je, že tuto metodu lze použít pouze u systémů, jejichž stav je popsán diferenciální rovnicí, kterou je nutné následně integrovat. Třída Source a její dceřinné třídy (zdroje) pro popis stavu používají jinou metodu pouze s jedním parametrem. Stav zdrojů je totiž popsán přímo funkcí času.

Metoda `update_output()` již je abstraktní metodou, neboť každý systém potřebuje mít nějakým způsobem definován výstup, ale různé druhy systémů jej mohou mít definován různým způsobem. Každý potomek třídy System si tedy metodu `update_output()` musí předefinovat.

3.3 Třída LinearSystem

Třída LinearSystem je potomkem třídy System. Instancí této třídy (objektem) je abstraktní (matematický model), orientovaný a kauzální lineární dynamický systém. Takový systém je možné popsat pomocí dynamických rovnic (1.3) (stavová rovnice) a (1.4) (výstupní rovnice). V případě LTI systémů je možné použití dynamických rovnic (1.5) (stavová rovnice) a (1.6) (výstupní rovnice). Na základě těchto rovnic je lineární dynamický systém popsán pomocí matic $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ a $\boldsymbol{D}$.

Konstruktor této třídy má všechny nepovinné parametry své rodičovské třídy System a také čtyři povinné parametry typu `ndarray`, které představují matice $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ a $\boldsymbol{D}$. Po zavolání konstruktoru rodiče a po kontrole těchto matic jsou tyto matice uloženy do proměnných.

```
    super().__init__(dt0, t0, x0,
                     number_of_inputs, number_of_outputs,
                     allowed_error, dt_max)

    self.matrix_controls((A, B, C, D))

    self.A = np.array(A)
    self.B = np.array(B)
    self.C = np.array(C)
    self.D = np.array(D)

    self.y = self.C @ self.x + self.D @ self.u
```

V konstruktoru je také na základě počátečních podmínek a matic $\boldsymbol{C}$ a $\boldsymbol{D}$ vypočítána počáteční hodnota výstupu `self.y`.

3.3.1 Metoda `equation_of_state()`

Jak již bylo zmíněno výše (sekce 3.2.10), tuto metodu si musí každý potomek třídy System, jehož stav je popsán pomocí diferenciální rovnice, upravit.

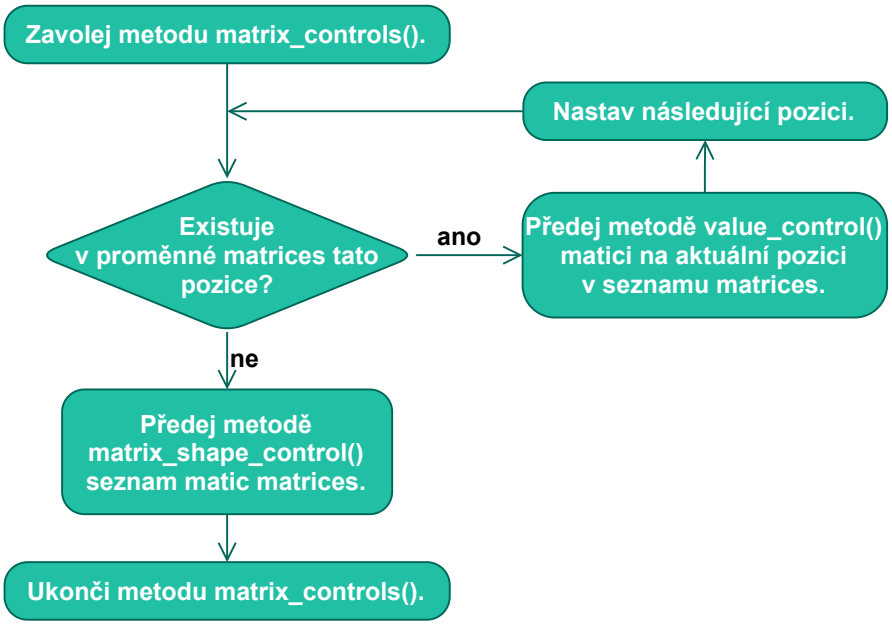
Stav lineárního dynamického systému popisuje rovnice (1.3), popř., jde-li o LTI systém, lze jeho stav popsat pomocí rovnice (1.5). Tato metoda dosazením do stavové rovnice číselně vypočítá hodnotu `dx` (derivace stavu), jež je následně metodou vrácena. Jelikož tato metoda vrací derivaci stavu, je nutné v kombinaci s ní použít jednu z dostupných integračních metod (metody `runge_kutta_step()` a `adaptive_step()`).

3.3.2 Metoda `update_output()`

Metoda `update_output()` ve třídě System je abstraktní. Je tedy potřeba ji zde předefinovat.

Činnost této metody spočívá pouze v aktualizaci hodnot vektoru výstupu $\boldsymbol{y}$ a přiřazení tohoto vektoru do proměnné `self.y`. Výstup lineárního dynamického systému je popsán rovnicí (1.4), případně rovnicí (1.6), jedná-li se o LTI systém. Metoda tedy pouze dosadí do této rovnice aktuální hodnoty stavu a vstupu (vektory $\boldsymbol{x}$ a $\boldsymbol{u}$) a výsledek přiřadí do proměnné `self.y`.

3.3.3 Metoda `matrix_controls()`



Obr. 3.5: Znázornění činnosti metody `matrix_controls()`

Metoda `matrix_controls()` je volána v konstruktoru třídy LinearSystem. Před vytvořením objektu lineárního dynamického systému je nejprve nutné zkontrolovat, zda bude možné tento objekt simulovat. Je tedy potřeba zkontrolovat, zda mají systémové matice správný tvar, a zda nejsou v maticích nepovolené hodnoty (např. textové znaky).

Ke kontrole tvaru matic slouží metoda `matrix_shape_control()` (sekce 3.3.5) a ke kontrole hodnot v maticích slouží metoda `value_control()` (sekce 3.3.4). Obě metody jsou podrobněji popsány níže.

Metodě `matrix_controls()` je v parametru předána n-tice `matrices`, která obsahuje systémové matice. V první fázi kontroly je postupně pomocí cyklu `for()` vybrána vždy jedna matice a je předána v parametru metodě `value_control()`. Po kontrole hodnot (nedojde-li k výjimce) je celá n-tice předána metodě `matrix_shape_control()` a následně je metoda `matrix_controls()` ukončena.

3.3.4 Metoda `value_control()`

Jak již bylo zmíněno výše, metoda `value_control()` zjišťuje, zda nejsou v maticích nepovolené hodnoty, jelikož simulace systémů je možná pouze tehdy, jsou-li v maticích čísla.

V parametru je metodě předána vždy jedna matice. Matice je následně prvek po prvku procházena pomocí dvou vnořených cyklů `for()`. U každého prvku metoda zkusí provést přetypování na `float`. Není-li přetypovávaný prvek číslem, dojde k výjimce.

3.3.5 Metoda `matrix_shape_control()`

Před výpočtem je potřeba zkontrolovat, zda matice $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ a $\boldsymbol{D}$ předané konstruktoru mají správný tvar, aby bylo možné případné výjimky „odchytit“ ještě před vytvořením objektu. Tvar matic vychází z pravidel pro maticové násobení a je určen tvarem vektorů $\boldsymbol{u}$, $\boldsymbol{x}$ a $\boldsymbol{y}$. Tyto podmínky již byly popsány v kapitole 1.3. Metoda `matrix_shape_control()` zajišťuje kontrolu těchto podmínek. Pokud není některá z nich splněna dojde k výjimce.

3.3.6 Metoda `frequency_analysis()`

Metoda `frequency_analysis()`, jak její název napovídá, provádí frekvenční analýzu LTI systému. K tomuto účelu využívá metodu `signal.bode()` z balíčku SciPy.

Tato metoda má dva nepovinné parametry:

- `omega_range=None`,
- `n=100`.

To jsou zároveň nepovinné parametry metody `signal.bode()`, jenž je zde používána.

Po zavolání této metody dojde k vytvoření objektu třídy `scipy.signal.StateSpace`. Do konstruktoru jsou předány matice `self.A`, `self.B`, `self.C` a `self.D`. Tento objekt je nakonec předán metodě `signal.bode()`.

Metoda `signal.bode()` vrací n-tici, která obsahuje hodnoty úhlové frekvence $\omega\,[\,\mathrm{rad}\cdot\mathrm{s}^{-1}\,]$ a jí odpovídající hodnoty přenosu $A\,[\,\mathrm{dB}\,]$ a fáze $\varphi\,[\,^\circ\,]$ typu `ndarray`. Hodnoty úhlové frekvence jsou převedeny na frekvenci $f\,[\,\mathrm{Hz}\,]$ vydělením konstantou $2\,\pi$ a následně jsou uloženy do archivu `self.archive_frequency`. Hodnoty přenosu a fáze jsou přímo uloženy do archivu `self.archive_bode`.

Pomocí hodnot v těchto dvou archivech je možné vykreslit amplitudovou a fázovou frekvenční charakteristiku.

3.4 Třída Source

Třída Source je abstraktní třídou a je potomkem třídy System. Instancemi této třídy (a jejích potomků) jsou zdroje. Tyto zdroje jsou zatím řešeny jako neřízené (nemají žádný vstup), s jedním výstupem a jejich chování je přesně popsáno matematickou funkcí času. Další vývoj balíčku počítá s možností vytváření řízených zdrojů.

Konstruktor třídy Source má pouze dva nepovinné parametry, jejichž výchozí hodnoty jsou nastaveny stejně jako u konstruktoru třídy System. Jsou to parametry

- `dt0=1.5e-5`,
- `t0=0`,

kde `dt0` si lze představit jako vzorkovací periodu. Udává časový rozestup mezi jednotlivými hodnotami, které jsou ukládány do archivů. Jeho velikost má vliv pouze na kvalitu vykreslení průběhů, nikoliv na přesnost hodnot. Hodnota `t0` má stejný význam jako ve třídě System (sekce 3.2).

Ostatní parametry konstruktoru třídy System jsou v konstruktoru třídy Source nastaveny na následující hodnoty:

- `x0=0`,
- `number_of_inputs=0`,
- `number_of_outputs=1`,
- `allowed_error=1e-6`,
- `dt_max=1e-2`.

Jelikož je zdroj popsán funkcí času, lze v libovolném čase zjistit přesnou (neaproximovanou) hodnotu stavu nebo výstupu zdroje. To umožňuje zjednodušení metod zdrojů pro aktualizaci stavu a výstupu.

3.4.1 Metody `step()` a `adaptive_step()`

Metoda `step()` ve třídě Source je téměř stejná jako ve třídě System. Nejprve aktualizuje čas, stav a výstup a poté tyto hodnoty archivuje. Vzhledem k tomu, že zatím není uvažována možnost řízených zdrojů, není tedy potřeba použití metody pro aktualizaci vstupu a tudíž ani jeho archivace.

Dalším rozdílem je použití jiné metody pro aktualizaci stavu. Není zde použita metoda `runge_kutta_step()` (numerická integrace), ale metoda `update_state()`, která na rozdíl od metody `runge_kutta_step()` neprovádí integraci, ale volá přímo funkci konkrétního zdroje v požadovaném čase. Podrobněji je tato metoda popsána níže.

Jelikož je možné zjistit přesnou hodnotu stavu zdroje v libovolném čase, je potřeba úprava kroku jen kvůli dostatečné kvalitě vykreslení průběhů. Z tohoto důvodu je zde adaptivní krok zbytečný a nebyl řešen. Metoda `adaptive_step()` pouze zavolá metodu `step()`. Smysl této realizace spočívá v tom, že metoda `simulate()` třídy Pool (sekce 3.1) potřebuje mít umožněno pro jakýkoliv systém zavolat buď metodu `step()` nebo metodu `adaptive_step()`. Metoda `adaptive_step()` zde tedy musí být k dispozici ve variantě pro zdroje, aby byla zajištěna správná funkčnost. Rozhodování o tom, zda se použije metoda `step()` nebo `adaptive_step()`, realizuje objekt třídy Pool pro všechny systémy (tedy i zdroje), jejichž simulaci tento objekt zajišťuje.

3.4.2 Metoda `update_state()`

Metoda `update_state()` nahrazuje metodu `runge_kutta_step()`. Jelikož numerická metoda Runge-Kutta je metodou integrační, bylo vhodné tuto metodu pro aktualizaci stavu pojmenovat jinak.

Činnost této metody spočívá pouze v zavolání funkce popisující požadovaný zdroj v požadovaném čase. Vrácená hodnota je následně uložena do proměnné `self.x`.

3.4.3 Abstraktní metoda `source_function()`

Metoda `source_function()` představuje funkci času popisující daný zdroj a nahrazuje metodu `equation_of_state()` z třídy System, která udává diferenciální rovnici popisující systém. Metodě `source_function()` k aktualizaci stavu stačí pouze předat hodnotu času, v němž je potřeba zjistit stav systému, a není potřeba provádět integraci. Má tedy pouze jeden parametr na rozdíl od metody `equation_of_state()`, které je kromě času potřeba předat také aktuální hodnotu stavu (vektor stavu).

Ve třídě Source je metoda `source_function()` pouze definována. Jde o abstraktní metodu, protože třída Source nepředstavuje žádný konkrétní zdroj, a tudíž neexistuje funkce času, která by jej popisovala. Dceřinné třídy si tuto metodu musí předefinovat.

3.4.4 Metoda `update_output()`

Výstup zdroje popisuje, stejně jako jeho stav, metoda `source_function()`. Aktualizace hodnoty výstupu tedy v případě zdrojů spočívá pouze v zavolání metody `source_function()`, přičemž je jí v parametru předána hodnota aktuálního času `self.t`, a v uložení návratové hodnoty do proměnné `self.y`.

Metoda `update_output()` také přepisuje abstraktní metodu ze třídy System.

3.4.5 Metoda `output()`

Metoda `output()` přepisuje metodu ze třídy System, kde je hodnota výstupu získávána, popř. aproximována, pomocí archivovaných hodnot výstupu a času (proměnné `self.archive_y` a `self.archive_t`). V případě třídy Source toto lze realizovat přesněji (bez aproximace) a rychleji (bez procházení archivů).

Jak již bylo zmíněno v sekci 3.4.4, výstup zdroje je popsán funkcí času, jíž představuje metoda `source_function()`. Hodnota výstupu v požadovaném čase je tedy získána zavoláním metody `source_function()` s požadovaným časem v parametru a je následně vrácena.

3.5 Abstraktní třída HarmonicFunctions

Tato třída má za úkol nadefinovat obecný harmonický zdroj. Harmonický průběh je možné matematicky popsat pomocí následujících vztahů:

$$f(t) = A_m \sin(\omega t + \varphi) , \tag{3.14}$$

$$f(t) = A_m \cos(\omega t + \varphi) , \tag{3.15}$$

kde ω představuje tzv. úhlovou frekvenci, která je definována, jako

$$\omega = 2\pi f . \tag{3.16}$$

Vztahy popisující harmonický průběh lze tedy přepsat do tvaru

$$f(t) = A_m \sin(2\pi f t + \varphi) , \tag{3.17}$$

$$f(t) = A_m \cos(2\pi f t + \varphi) , \tag{3.18}$$

kde

A_m představuje amplitudu harmonického průběhu,

f je frekvence kmitů [Hz],

φ je fázový posun harmonického průběhu [rad].

Třída HarmonicFunctions je potomkem třídy Source a jejími potomky jsou třídy Sine a Cosine. Instancemi těchto tříd jsou harmonické funkce sinus nebo cosinus. Konstruktory těchto tříd mají následující parametry:

- `amplitude`,
- `frequency`,
- `phase`,
- `dt0=1.5e-5`,
- `t0=0`.

První tři parametry jsou povinné a slouží přímo k matematickému popisu harmonické funkce (vztahy (3.17) a (3.18)). Konstruktor třídy HarmonicFunctions nejprve zavolá konstruktor rodiče, jemuž předá hodnoty nepovinných parametrů `dt0` a `t0`. Následně jsou hodnoty povinných parametrů uloženy do proměnných.

```
    super().__init__(dt0, t0)

    self.amplitude = amplitude
    self.frequency = frequency
    self.phase = phase
```

Objekt následně tyto proměnné využívá pro výpočet hodnot stavu a výstupu. Zbylé dva parametry jsou nepovinné a jejich význam byl již vysvětlen (sekce 3.4).

V této třídě nejsou definovány ani přepisovány žádné metody. Slouží pouze k nadefinování výše zmíněných atributů. Jelikož nepředstavuje žádný konkrétní zdroj, je třída HarmonicFunctions abstraktní.

3.6 Třídy Sine a Cosine

Jak již bylo uvedeno, třídy Sine a Cosine jsou potomky třídy HarmonicFunctions. Jejich konstruktory pouze zavolají konstruktor rodičovské třídy.

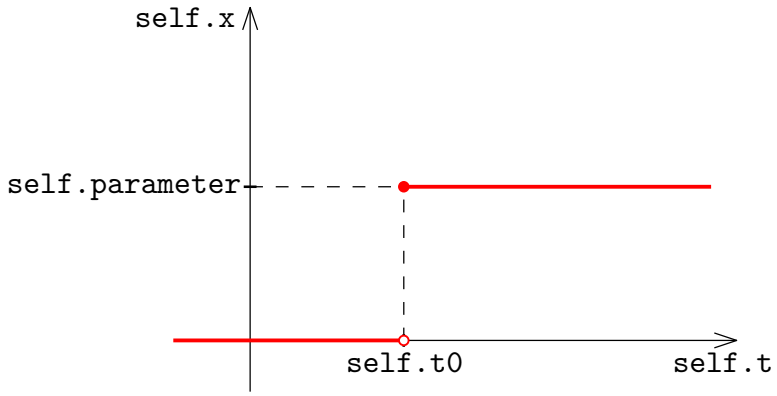
V těchto třídách je pouze přepsána metoda `source_function()`. Ve třídě Sine je tato metoda upravena podle vztahu (3.17), ve třídě Cosine podle vztahu (3.18).

3.7 Třída UnitStep

Třída UnitStep definuje zdroj, jehož stav a výstup má do času `self.t0` hodnotu `0.0` a v čase `self.t0` „skočí“ na hodnotu `self.parameter`. Tato hodnota je poté až do ukončení simulace konstantní. Konstruktor této třídy má tři nepovinné parametry

- `parameter=1`,
- `dt0=1.5e-5`,
- `t0=0`.

Hodnota parametru `parameter` je v konstruktoru uložena do proměnné `self.parameter` a představuje konstantní hodnotu stavu i výstupu zdroje od času `self.t0` až do doby ukončení simulace. Význam zbylých dvou parametrů je stejný jako u ostatních zdrojů (viz sekce 3.4).



Obr. 3.6: Graf závislosti stavu `self.x` zdroje třídy UnitStep na čase `self.t`

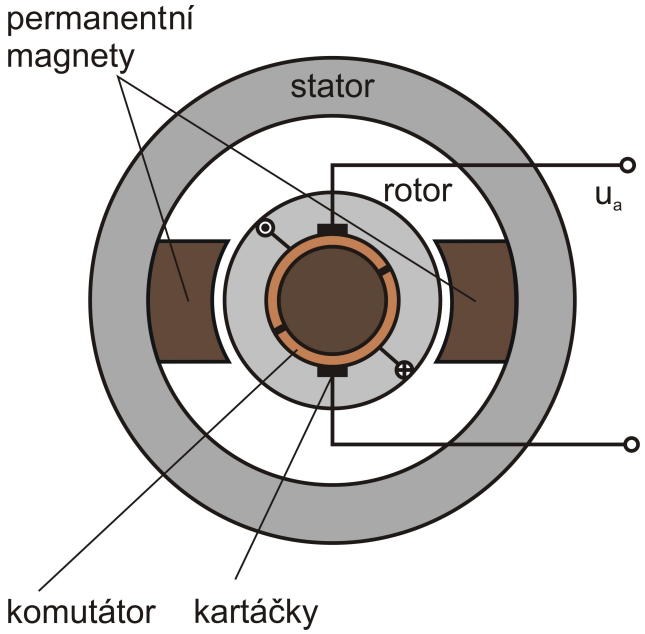
V této třídě je pouze přepsána metoda `source_function()` tak, aby pro hodnoty parametru `t` menší než `self.t0` vracela hodnotu `0.0`, jinak hodnotu `self.parameter`.

Kapitola 4

Činnost balíčku DynSyPy

Tato kapitola je věnována demonstraci funkčnosti balíčku DynSyPy. Pro tento účel byly zvoleny dva testovací příklady. Prvním testovacím příkladem je stejnosměrný motor s permanentními magnety, který slouží pro ověření funkčnosti numerického řešení diferenciálních rovnic. Druhým příkladem je sériový RLC obvod, na němž je demonstrována metoda pro analýzu ve frekvenční oblasti (metoda `frequency_analysis()`).

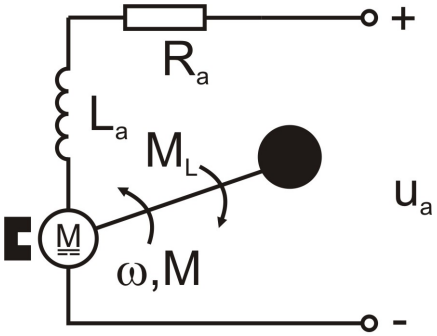
4.1 Testovací příklad – stejnosměrný motor s permanentními magnety



Obr. 4.1: Stejnosměrný motor s permanentními magnety

Stejnosměrný motor s permanentními magnety si lze zjednodušeně představit jako LTI (lineární t-invariantní) dynamický systém se dvěma vstupy. Tyto vstupy představují

napájecí napětí kotvy $u_a[\,\mathrm{V}\,]$ a zátěžný moment $M_L[\,\mathrm{Nm}\,]$ působící na hřídel motoru. Stavovými veličinami tohoto systému jsou proud tekoucí obvodem rotoru $i_a[\,\mathrm{A}\,]$ a úhlová frekvence rotoru $\omega[\,\mathrm{rad}\cdot\mathrm{s}^{-1}\,]$.



Obr. 4.2: Náhradní schéma stejnosměrného motoru

Tyto stavové veličiny je možné popsat pomocí následujících rovnic:

$$\frac{\mathrm{d}i_a}{\mathrm{d}t} = -\frac{R_a}{L_a}i_a - \frac{k_a}{L_a}\omega + \frac{1}{L_a}u_a\ , \tag{4.1}$$

$$\frac{\mathrm{d}\omega}{\mathrm{d}t} = \frac{k_a}{J}i_a - \frac{B_m}{J}\omega - \frac{1}{J}M_L\ . \tag{4.2}$$

Soustavu diferenciálních rovnic (4.1) a (4.2) lze přepsat do maticového tvaru, což je tvar stavové rovnice systému (1.5). Z této rovnice je tedy možné stanovit systémovou matici $\boldsymbol{A}$ a vstupní matici $\boldsymbol{B}$, jak je zde naznačeno:

$$\begin{bmatrix} \dfrac{\mathrm{d}i_a}{\mathrm{d}t} \\ \\ \dfrac{\mathrm{d}\omega}{\mathrm{d}t} \end{bmatrix} = \overbrace{\begin{bmatrix} -\dfrac{R_a}{L_a} & -\dfrac{k_a}{L_a} \\ \\ \dfrac{k_a}{J} & -\dfrac{B_m}{J} \end{bmatrix}}^{\boldsymbol{A}} \begin{bmatrix} i_a \\ \omega \end{bmatrix} + \overbrace{\begin{bmatrix} \dfrac{1}{L_a} & 0 \\ \\ 0 & -\dfrac{1}{J} \end{bmatrix}}^{\boldsymbol{B}} \begin{bmatrix} u_a \\ M_L \end{bmatrix}\ . \tag{4.3}$$

U tohoto systému jsou v rámci testovacího příkladu sledovány stavové veličiny. Vektor výstupu $\boldsymbol{y}$ je tedy v tomto případě shodný s vektorem stavu $\boldsymbol{x}$. Výstupní rovnice (1.6) bude mít pro tento systém následující tvar:

$$\begin{bmatrix} i_a \\ \omega \end{bmatrix} = \overbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}^{\boldsymbol{C}} \begin{bmatrix} i_a \\ \omega \end{bmatrix} + \overbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}}^{\boldsymbol{D}} \begin{bmatrix} u_a \\ M_L \end{bmatrix}\ . \tag{4.4}$$

Z této rovnice lze stanovit zbývající výstupní matici $\boldsymbol{C}$ a matici přímého působení vstupu na výstup $\boldsymbol{D}$.

Pro simulaci systému byly použity následující hodnoty:

$$
\begin{aligned}
R_a &= 2,7\,\Omega\ , \\
L_a &= 4\,\mathrm{mH}\ , \\
k_a &= 0,105\,\mathrm{V}\cdot\mathrm{s}\cdot\mathrm{rad}^{-1}\ , \\
k_a &= 0,105\,\mathrm{N}\cdot\mathrm{m}\cdot\mathrm{A}^{-1}\ , \\
J &= 10^{-4}\,\mathrm{kg}\cdot\mathrm{m}^2\ , \\
B_m &= 93\cdot 10^{-7}\,\mathrm{N}\cdot\mathrm{m}\cdot\mathrm{s}
\end{aligned}
$$

a byly zvoleny nulové počáteční podmínky. V čase $t_0 = 0\,\mathrm{s}$ tedy platí:

$$
\begin{aligned}
i_a(0) &= 0\,\mathrm{A}\ , \\
\omega(0) &= 0\,\mathrm{rad}\cdot\mathrm{s}^{-1}\ .
\end{aligned}
$$

Výsledky simulace byly zjišťovány pro dvě různá napájecí napětí při stejném zátěžném momentu:

$$
\begin{aligned}
M_L &= 0,1\,\mathrm{Nm}\ , \\
u_{a_1} &= 4\,\mathrm{V}\ , \\
u_{a_2} &= 5\,\mathrm{V}\ .
\end{aligned}
$$

4.1.1 Simulace systému pomocí DynSyPy

Aby bylo možné s balíčkem DynSyPy pracovat, je nutné jej nejprve naimportovat do projektu. Spolu s ním je vhodný také import balíčku NumPy, který umožňuje vytvářet proměnné typu `ndarray`, s nimiž je možné pracovat jako s maticemi. Aby bylo možné výsledky simulace vynést do grafů, je zde využit také balíček matplotlib.

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

from dynsypy import *
```

K vykreslování grafů byla pro tento příklad vytvořena jednoduchá funkce `plotter()`, které jsou v parametru předány hodnoty času a hodnoty výstupů.

```
def plotter(t_array, y_array):

    ax1 = plt.subplot(211)
    plt.plot(t_array, y_array[0, :],
             'g', label='$i_a(t)\ [$A$]$')
    plt.setp(ax1.get_xticklabels(), fontsize=6)
```

```
    ax1.legend(loc='best')
    plt.grid()

    ax2 = plt.subplot(212, sharex=ax1)
    plt.plot(t_array, y_array[1, :],
             'b', label='$\omega(t)\ [$rad s$^{-1}]$')
    plt.setp(ax2.get_xticklabels(), fontsize=6)
    ax2.legend(loc='best')
    plt.grid()

    plt.xlabel('$t\ [$s$]$')
    plt.show()
```

Z důvodu názornosti byly zadané hodnoty uloženy do proměnných. Čas ukončení simulace byl zvolen `t_end = 0.15`. Z počátečních hodnot stavových veličin byl vytvořen vektor stavu `x0`.

```
t0 = 0
t_end = 0.15

i_a0 = 0.0
omega0 = 0.0

x0 = np.array([[i_a0],
               [omega0]])

u_a = 4
M_L = 0.1

R_a = 2.7
L_a = 4e-3
k_a = 0.105
J = 1e-4
B_m = 93e-7
```

Pomocí vytvořených proměnných je nyní možné podle rovnic (4.3) a (4.4) sestavit matice $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ a $\boldsymbol{D}$.

```
A = np.array([[-R_a / L_a, -k_a / L_a],
              [k_a / J, -B_m / J]])

B = np.array([[1 / L_a, 0],
              [0, -1 / J]])

C = np.array([[1, 0],
              [0, 1]])
```

```
D = np.array([[0, 0],
              [0, 0]])
```

Nyní je již možné vytvořit jednotlivé systémy a instanci třídy Pool, která realizuje simulaci.

Objektu třídy Pool je nastaven krok na hodnotu $0,01\,\mathrm{s}$, čas ukončení simulace na hodnotu `t_end` a čas počátku simulace na hodnotu `t0`. Jelikož nepovinný parametr `adaptive_step` není vyplněn, bude použita přednastavená hodnota `True`. Bude tedy použit integrátor s adaptivním krokem.

Jelikož má tento systém dvě různé vstupní veličiny, je potřeba vytvořit dva objekty. Hodnoty obou těchto veličin jsou konstantní, je tedy možné je zrealizovat jako objekty třídy UnitStep.

```
pool = Pool(0.01, t_end, t0)

voltage = UnitStep(u_a)

load_torque = UnitStep(M_L)
```

Stejnosměrný motor je zde chápán jako LTI systém. Pro jeho simulaci je tedy potřeba vytvořit instanci třídy LinearSystem. Nepovinné parametry `x0`, `number_of_inputs` a `number_of_outputs` je nutné vyplnit, neboť tento systém má více než jednu stavovou veličinu, více než jeden vstup a také více než jeden výstup.

Dále je nutné k systému „připojit“ zdroje pomocí metody `DC_motor.connect()`. Připojování zdrojů je nutné provádět podle pořadí veličin ve vektoru vstupu (viz rovnice (4.3), popř. (4.4)).

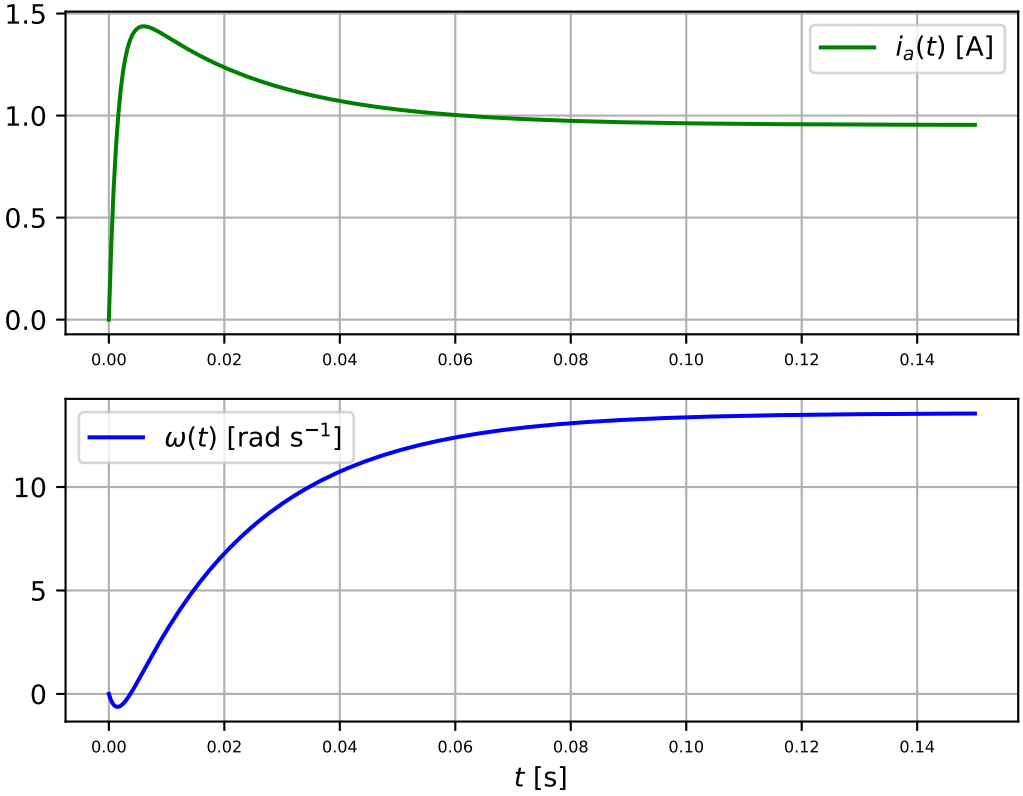
Nakonec musí být systém pomocí metody `pool.add()` předán objektu třídy Pool, aby mohla proběhnout simulace.

```
DC_motor = LinearSystem(A, B, C, D, t0=t0, x0=x0,
                        number_of_inputs=2, number_of_outputs=2)
DC_motor.connect(voltage.output, 0)
DC_motor.connect(load_torque.output, 1)

pool.add(DC_motor)
```

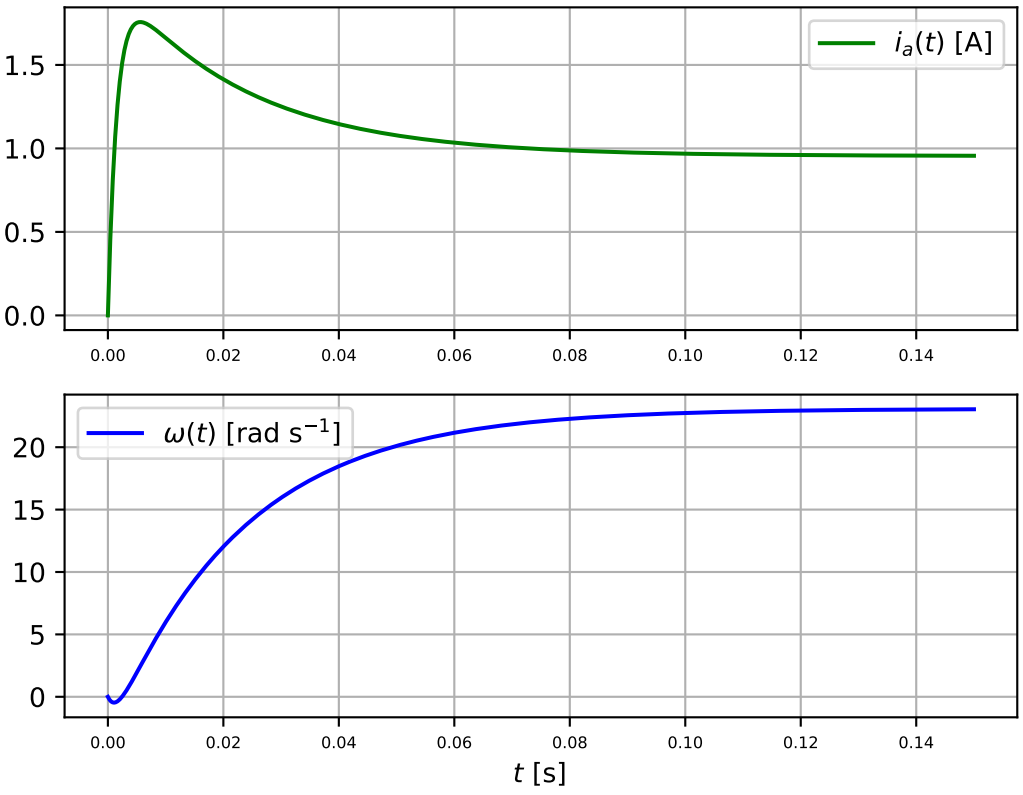
Nyní již pro realizaci simulace stačí zavolat metodu `pool.simulate()` a pro vykreslení průběhů funkci `plotter()`.

```
pool.simulate()

plotter(DC_motor.archive_t, DC_motor.archive_y)
```

4.1.2 Výsledky simulací



Obr. 4.3: Vykreslené průběhy pro hodnoty vstupu $u_a = 4\,\mathrm{V}$ a $M_L = 0,1\,\mathrm{Nm}$



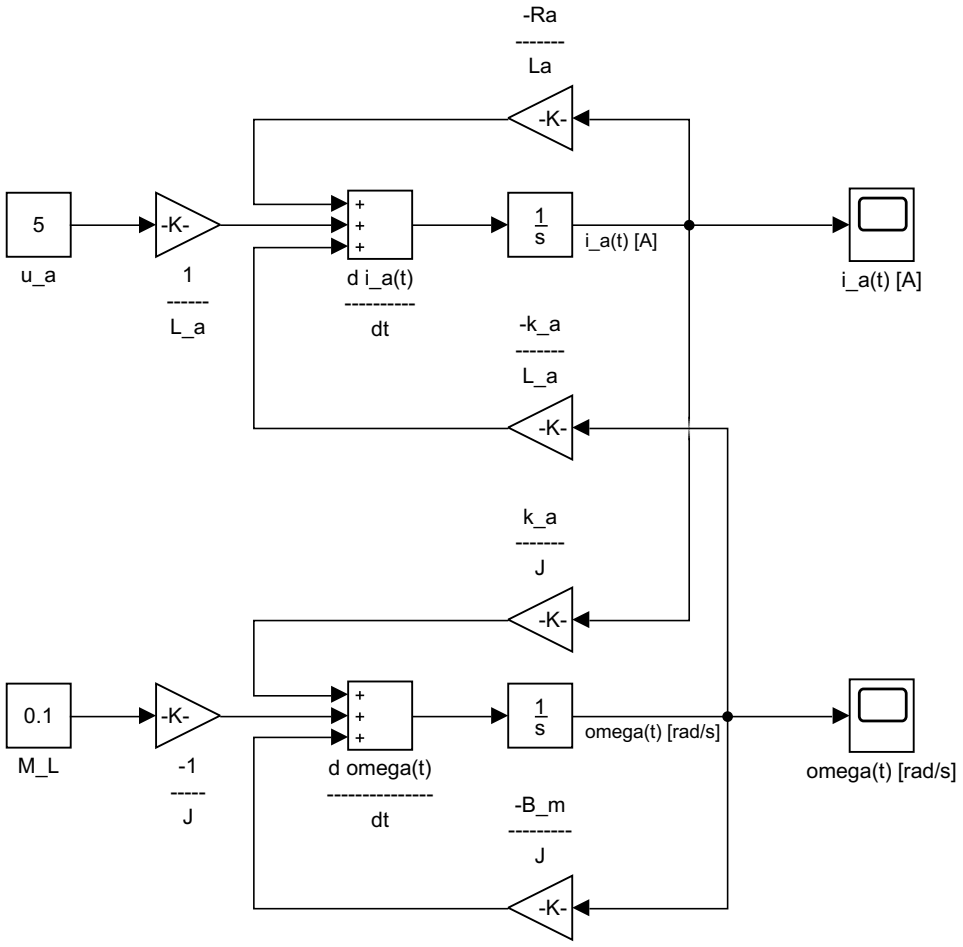
Obr. 4.4: Vykreslené průběhy pro hodnoty vstupu $u_a = 5\,\mathrm{V}$ a $M_L = 0,1\,\mathrm{Nm}$

Na obrázcích 4.3 a 4.4 jsou grafické výstupy ze simulace systému vykreslené pomocí funkce `plotter()`.

Lze si všimnout, že úhlová frekvence $\omega(t)$ rotoru je zpočátku záporná (motor se točí opačně). Je to způsobeno tím, že na hřídel motoru v tomto případě působí konstantní zátěžný moment M_L. Tento moment má zpočátku větší velikost než elektromagnetický moment motoru M. S rostoucím proudem v obvodu kotvy roste také velikost elektromagnetického momentu M, tudíž po chvíli dojde k překonání zátěžného momentu M_L a motor se roztočí „správným“ směrem.

Grafy také ukazují, že při vyšším napájecím napětí u_a se motor točí rychleji (úhlová frekvence $\omega(t)$ je vyšší) $\rightarrow$ otáčky stejnosměrného motoru s permanentními magnety lze měnit pomocí napájecího napětí obvodu kotvy. Zároveň je při vyšším napájecím napětí zátěžný moment M_L překonán rychleji.

Pro ověření správnosti výsledků byl systém simulován také pomocí nástroje MATLAB Simulink. Realizace systému pomocí tohoto nástroje je ukázána na obr. 4.5.



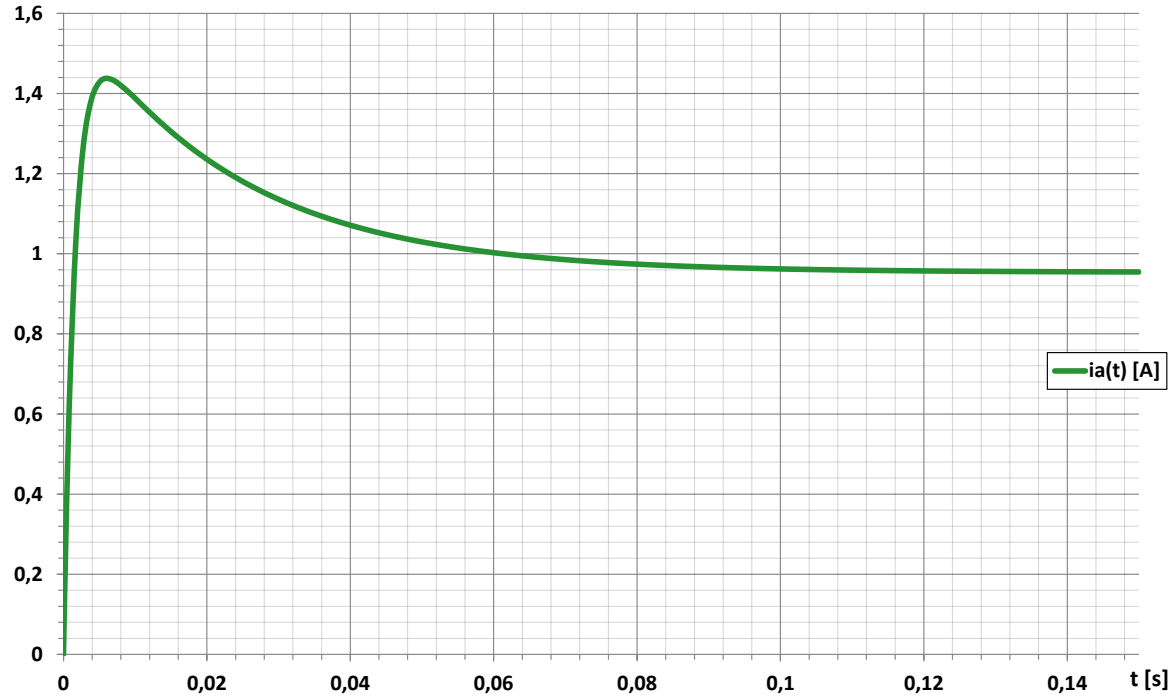
Obr. 4.5: Realizace testovacího příkladu 4.1 pomocí MATLAB Simulink

Jelikož grafické výstupy z tohoto nástroje by bylo nutné zmenšit, čímž by došlo k výraznému snížení kvality a čitelnosti, byl k vykreslení průběhů použit jiný nástroj. Pomocí MATLAB Simulink byla získána data, která byla následně vykreslena pomocí

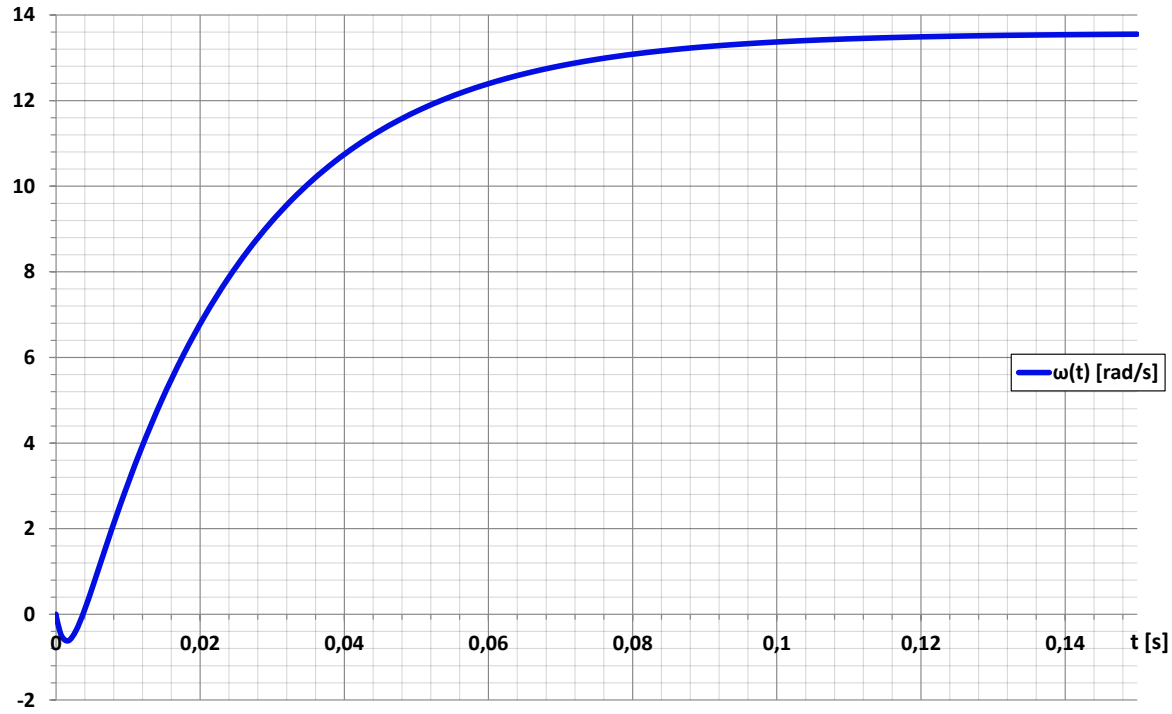
nástroje Microsoft Excel.

Na obr. 4.6 a 4.7 jsou vykreslená data pro motor napájený ze zdroje napětí $u_a = 4\,\mathrm{V}$ a na obr. 4.8 a 4.9 pro motor napájený ze zdroje napětí $u_a = 5\,\mathrm{V}$.

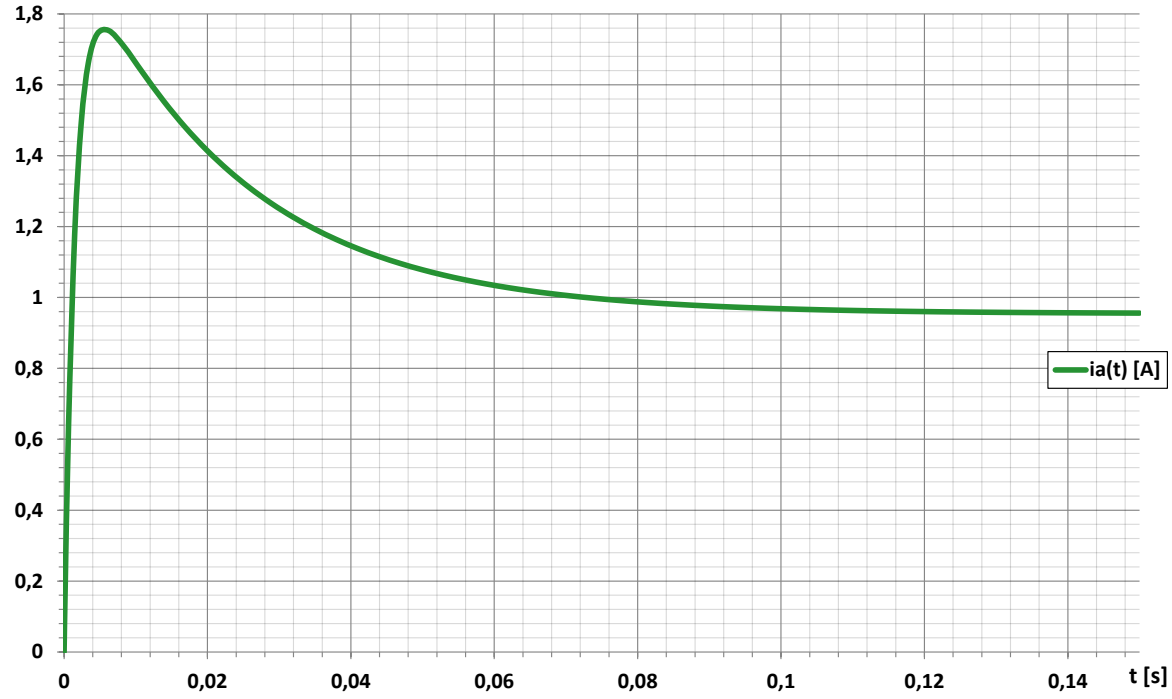
Grafické výstupy přímo z MATLAB Simulink jsou umístěny v příloze A.1.



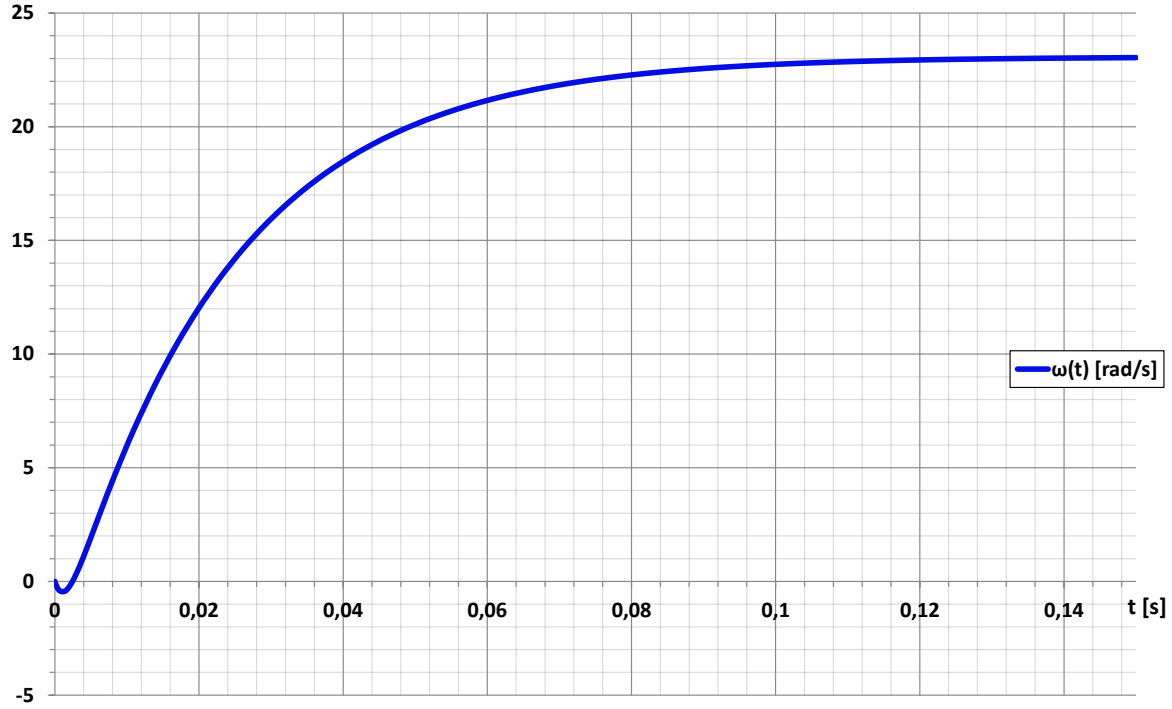
Obr. 4.6: Průběh proudu $i_a(t)$ pro hodnoty vstupu $u_a = 4\,\mathrm{V}$ a $M_L = 0,1\,\mathrm{Nm}$



Obr. 4.7: Průběh úhlové frekvence $\omega(t)$ pro hodnoty vstupu $u_a = 4\,\mathrm{V}$ a $M_L = 0,1\,\mathrm{Nm}$



Obr. 4.8: Průběh proudu $i_a(t)$ pro hodnoty vstupu $u_a = 5\,\mathrm{V}$ a $M_L = 0,1\,\mathrm{Nm}$

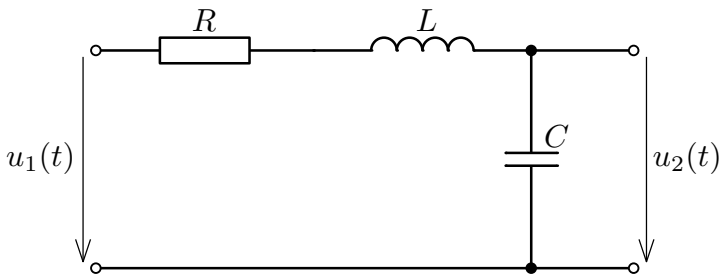


Obr. 4.9: Průběh úhlové frekvence $\omega(t)$ pro hodnoty vstupu $u_a = 5\,\mathrm{V}$ a $M_L = 0,1\,\mathrm{Nm}$

4.2 Testovací příklad – Sériový RLC obvod

Pro demonstraci funkčnosti metody `frequency_analysis()` je zde uveden sériový RLC obvod, který byl již zmíněn v teoretickém úvodu (viz sekce 1.3.1). Jeho schéma ovšem

bylo upraveno do podoby vhodnější pro frekvenční analýzu testovaného obvodu. Vstupem systému je tedy harmonický napěťový signál a jeho výstupem napětí na kapacitoru.



Obr. 4.10: Schéma zapojení RLC obvodu pro frekvenční analýzu

Rovnice pro sériový RLC obvod již byly výše vypočítány (viz rovnice (1.11) a (1.12)), matice $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ a $\boldsymbol{D}$ zde budou stejné.

Pro frekvenční analýzu byly použity následující hodnoty součástek:

$$R = 5\,\Omega\ ,$$
$$L = 0,1\,\mathrm{H}\ ,$$
$$C = 0,001\,\mathrm{F}\ ,$$

4.2.1 Realizace frekvenční analýzy systému pomocí DynSyPy

Frekvenční charakteristiky jsou obvykle vykreslovány do grafů s logaritmickou osou x. Byla proto vytvořena nová funkce pro vykreslování `plotter_log_axis()`.

```
    def plotter_log_axis(freq_array, val_array):

        ax1 = plt.subplot(211)
        plt.semilogx(freq_array, val_array[0, :],
                     color="green", label='$A(f)\ [$dB$]$')
        plt.setp(ax1.get_xticklabels(), fontsize=6)
        ax1.legend(loc='best')
        plt.grid()

        ax2 = plt.subplot(212, sharex=ax1)
        plt.semilogx(freq_array, val_array[1, :],
                     color="blue", label='${\\varphi}(f)\ [$°$]$')
        plt.setp(ax2.get_xticklabels(), fontsize=6)
        ax2.legend(loc='best')
        plt.grid()

        plt.xlabel('$f\ [$Hz$]$')

        plt.show()
```

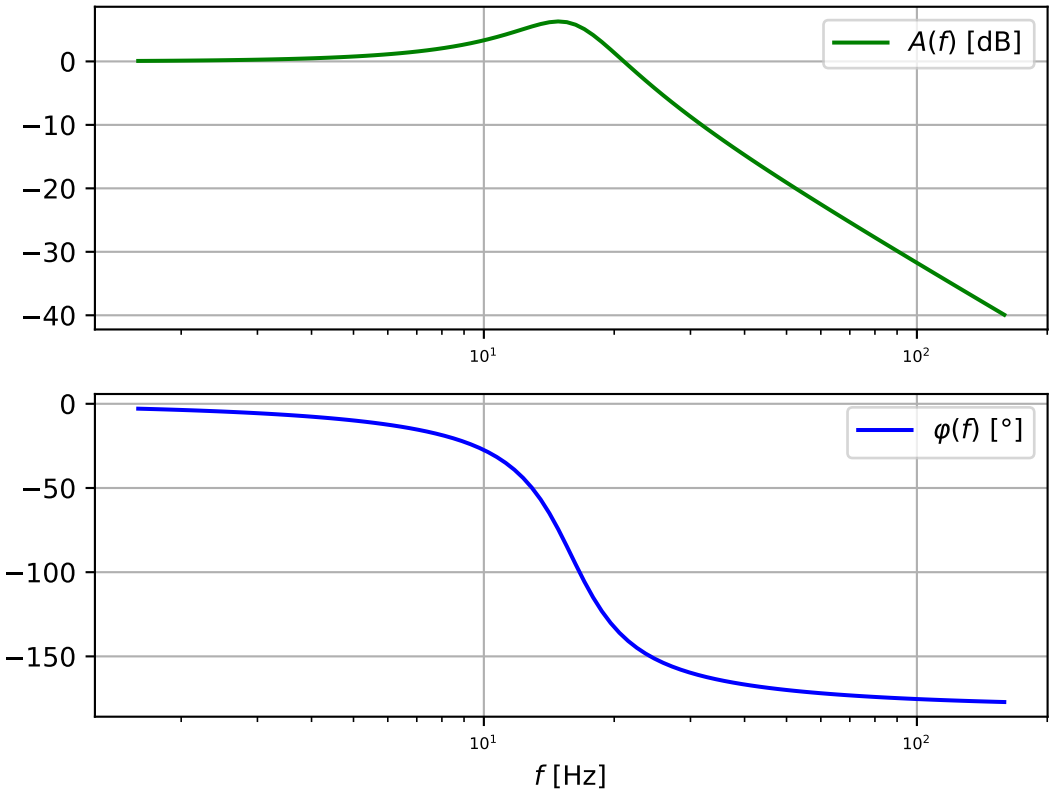
Z důvodu názornosti byly hodnoty součástek uloženy do proměnných. Z nich byly poté sestaveny matice $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ a $\boldsymbol{D}$. Počáteční podmínky stavových veličin by nebylo nutné vyplňovat, pokud by v systému figurovala pouze jedna stavová veličina. Jelikož jsou zde dvě, je nutné sestavit také počáteční vektor stavu, aby bylo možné vytvořit objekt třídy LinearSystem.

```
iL0 = 0.0
uC0 = 0.0

x0 = np.array([[iL0],
               [uC0]])

R = 5
L = 0.1
C = 0.001

A1 = np.array([[-R / L, -1 / L],
               [1 / C, 0]])

B1 = np.array([[1 / L],
               [0]])

C1 = np.array([[0.0, 1.0]])

D1 = np.array([[0]])
```

Pomocí sestavených matic $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ a $\boldsymbol{D}$ je již možné vytvořit objekt třídy LinearSystem a provést frekvenční analýzu systému.

```
transient_2 = LinearSystem(A1, B1, C1, D1, x0=x0)

transient_2.frequency_analysis()

plotter_log_axis(transient_2.archive_frequency,
                 transient_2.archive_bode)
```

4.2.2 Výsledky frekvenční analýzy



Obr. 4.11: Amplitudová a fázová frekvenční charakteristika pomocí balíčku DynSyPy

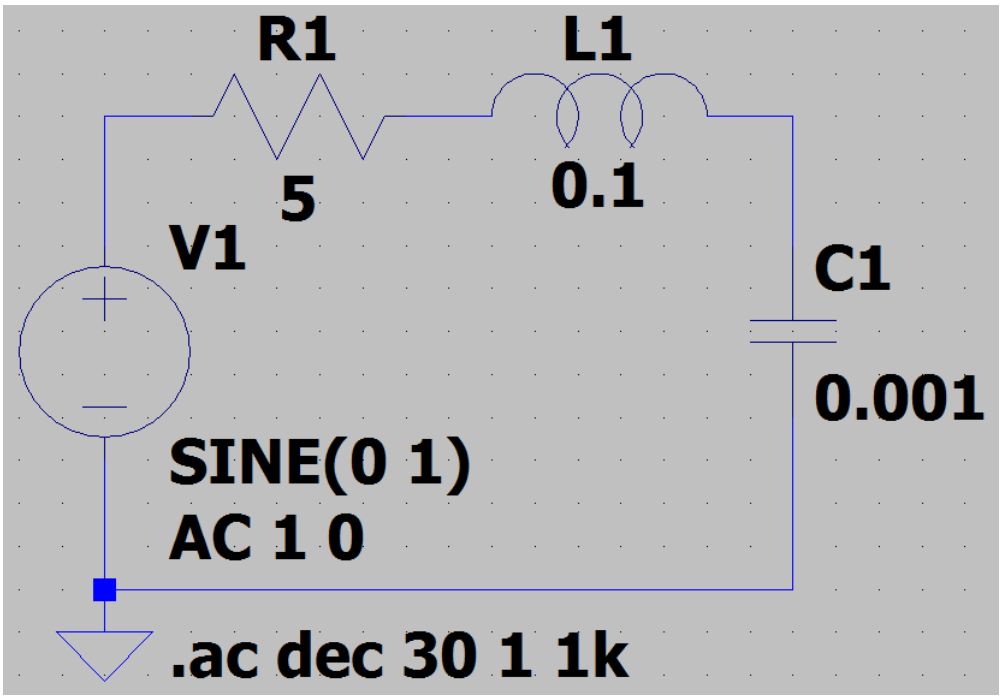
Na obr. 4.11 jsou vykreslená data získaná pomocí metody `frequency_analysis()`. Na amplitudové frekvenční charakteristice je vidět, že v oblasti kolem rezonanční frekvence dochází k rezonančnímu převýšení. To je způsobeno tím, že velikost odporu R v obvodu je menší než velikost tzv. kritického odporu R_k, což je odpor, při kterém by v tomto obvodu nastal přechodný děj na mezi aperiodicity.

Pro ověření správnosti výsledků byl tento systém simulován také pomocí nástroje LTspice. Realizace systému pomocí nástroje LTspice je ukázána na obr. 4.12.

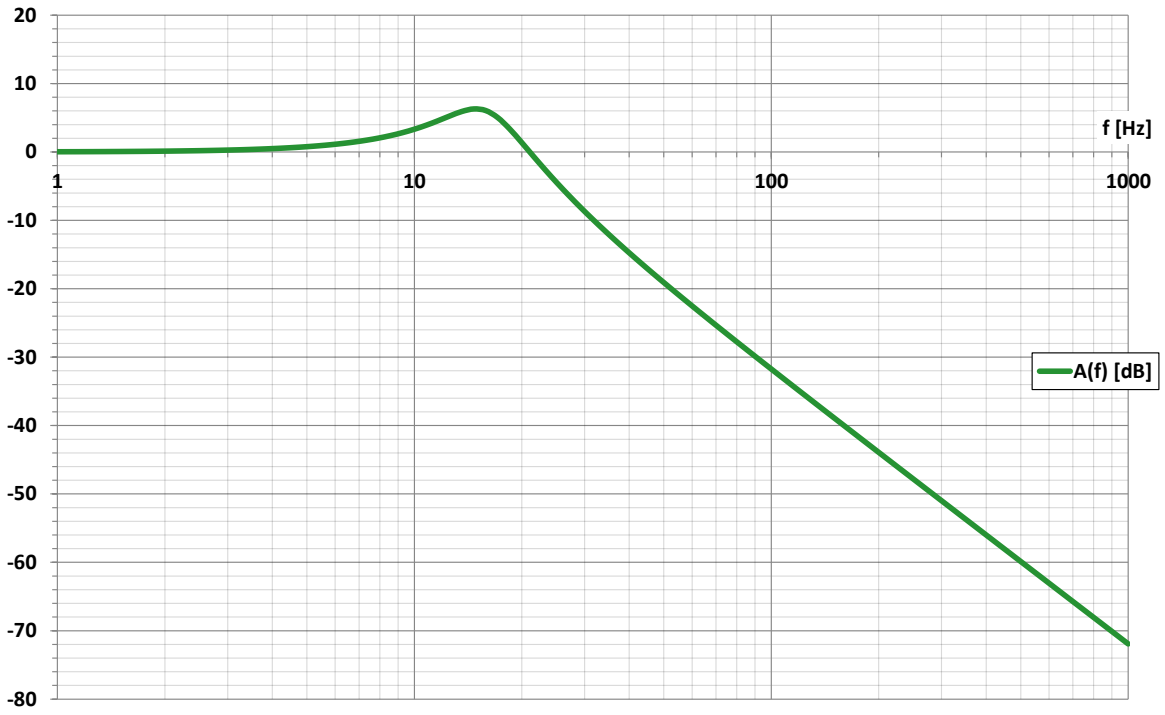
Ze stejných důvodu jako u příkladu se stejnosměrným motorem byl nástroj LTspice využit pouze k získání dat, která byla následně vykreslena pomocí nástroje Microsoft Excel.

Na obr. 4.13 je vynesena amplitudová frekvenční charakteristika a na obr. 4.14 je vynesena fázová frekvenční charakteristika.

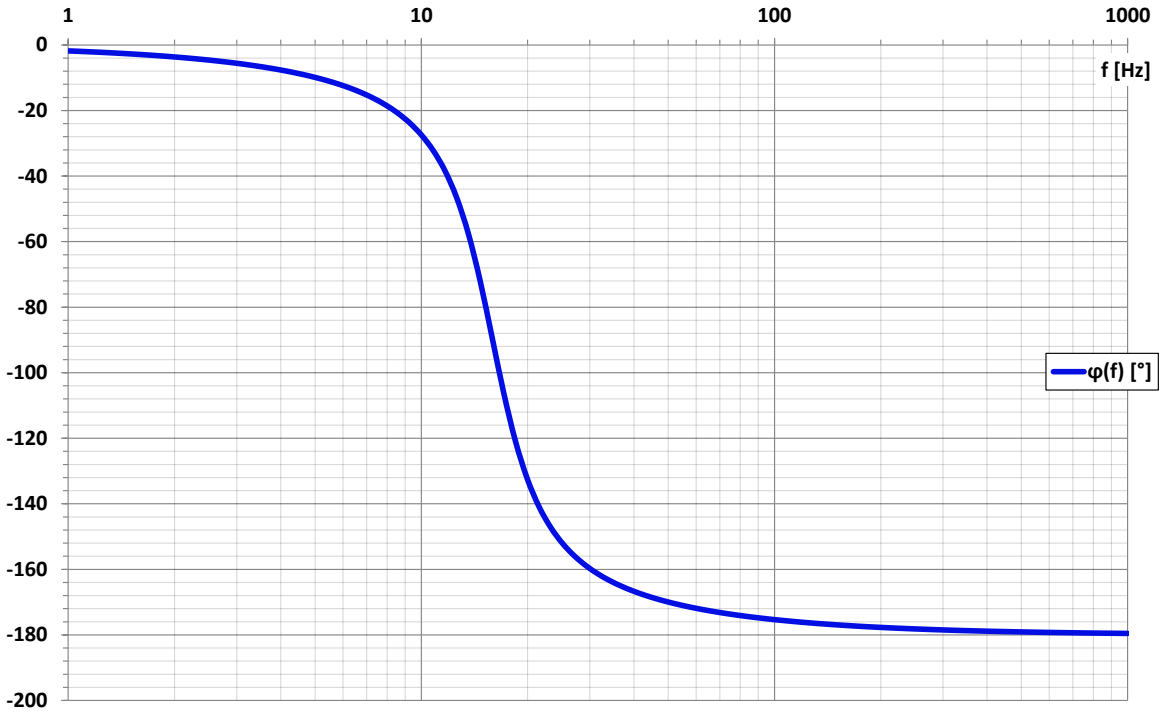
Grafické výstupy přímo z LTspice jsou umístěny v příloze A.2.



Obr. 4.12: Realizace testovacího příkladu 4.2 pomocí LTspice



Obr. 4.13: Amplitudová frekvenční charakteristika z hodnot získaných pomocí nástroje LTspice



Obr. 4.14: Fázová frekvenční charakteristika z hodnot získaných pomocí nástroje LTspice

Závěr

Cílem bakalářské práce bylo vytvořit balíček v jazyce Python, který by byl schopen vytvářet LTI systémy, k nim připojit zdroje a provádět jejich analýzu v časové i frekvenční oblasti. Požadovaný cíl byl splněn vytvořením balíčku DynSyPy, který umožňuje zmíněné systémy modelovat. K LTI systémům je možné připojit základní zdroje, také je možné tyto systémy vzájemně propojovat.

Jelikož jsou LTI systémy popsány pomocí ODE, bylo nutné k jejich řešení do třídy System implementovat metody pro numerickou integraci. V třídě System jsou dostupné integrační metody s pevným, ale také s adaptivním krokem. Pro umožnění využití metody s adaptivním krokem bylo potřeba vytvořit prostředí, které zajistí, že simulované systémy budou v průběhu simulace synchronizovány. Toho je docíleno použitím třídy Pool a jejích metod. Aby bylo možné v rámci balíčku DynSyPy provádět základní analýzu ve frekvenční oblasti, byla ve třídě LinearSystem vytvořena metoda `frequency_analysis()` využívající balíček SciPy, která na základě systémových matic vypočítá hodnoty přenosu a fáze v závislosti na frekvenci.

Kontrolní simulace testovacích příkladů prováděné pomocí nástrojů MATLAB Simulink a LTspice prokázaly, že balíček DynSyPy provádí spolehlivě a s dostatečnou přesností analýzy jak v časové, tak ve frekvenční oblasti.

V rámci dalšího vývoje balíčku DynSyPy bude vhodné rozšířit možnosti spojování systémů, zejména umožnění zpětnovazebního spojení systémů, čehož lze v budoucnu využít např. pro modelování regulačních smyček. Pro zrychlení simulace vzájemně spojených LTI systémů se stejnou vzorkovací frekvencí bude balíček pravděpodobně v budoucnu rozšířen o metody pro spojování těchto systémů, vycházející z principů popsaných v sekci 1.3.3.

Další vývoj balíčku bude také pravděpodobně vyžadovat rozšíření třídy Source tak, aby byla schopna vytvářet řízené zdroje, a zároveň do balíčku přidat nové zdroje, jako např. zdroj šumu.

Kód balíčku DynSyPy je dostupný na GitHub, viz [5].

Použitá literatura

[1] Geoff Boeing. "Pynamical: Model and visualize discrete nonlinear dynamical systems, chaos, and fractals". In: *Journal of Open Source Education* [online] 1.1 (2018). [cit. 2.6.2021], p. 15. ISSN: 2577-3569. DOI: 10.21105/jose.00015.

[2] Geoff Boeing. "Visual Analysis of Nonlinear Dynamical Systems: Chaos, Fractals, Self-Similarity and the Limits of Prediction". In: *Systems* [online] 4.4 (2016). [cit. 2.6.2021], p. 37. ISSN: 2079-8954. DOI: 10.3390/systems4040037.

[3] Robert Clewley et al. *PyDSTool, a software environment for dynamical systems modeling*. [online]. 2007 [cit. 3.6.2021]. URL: http://pydstool.sourceforge.net.

[4] António Ferreira. *S-timator: dynamical systems modelling in python*. [online]. © 2006 [cit. 3.6.2021]. URL: https://webpages.ciencias.ulisboa.pt/~aeferreira/stimator/.

[5] GITHUB. *kadlec99/DynSyPy*. GitHub [online]. GitHub, © 2021 [cit. 7.6.2021]. URL: https://github.com/kadlec99/DynSyPy.

[6] GITHUB. *simupy/simupy*. GitHub [online]. GitHub, © 2021 [cit. 2.6.2021]. URL: https://github.com/simupy/simupy.

[7] Oliver Grasl. *A Simple Python Library For System Dynamics*. In. *Transentis.com* [online]. 1.10.2018 [cit. 2.6.2021]. URL: https://www.transentis.com/simple-python-library-system-dynamics/en/.

[8] James Houghton and Michael Siegel. "Advanced data analytics for system dynamics models using PySD". In: *Proceedings of the 33rd International Conference of the System Dynamics Society* (2015). [online]. [cit. 3.6.2021]. URL: https://pysd.readthedocs.io/en/master/.

[9] Pavel Karban et al. "FEM based robust design optimization with Agros and Ārtap". In: *Computers & Mathematics with Applications* 81 (2020). [cit. 7.6.2021], pp. 618–633. ISSN: 0898-1221. DOI: https://doi.org/10.1016/j.camwa.2020.02.010.

[10] Benjamin W. L. Margolis. *API Documentation – SimuPy 1.0.0 documentation*. [online]. © 2015 [cit. 2.6.2021]. URL: https://simupy.readthedocs.io/en/latest/api/api.html.

[11] Gordon College Department of Mathematics and Computer Science. *Numerical Solution of Ordinary Differential Equations: Single-step and Multistep methods*. In. *Math-cs.gordon.edu* [online]. 2019 [cit. 17.05.2021]. URL: `http://www.math-cs.gordon.edu/courses/mat342/python/diffeq.py`.

[12] John H. MATHEWS and Kurtis D. FINK. *Numerical Methods Using MATLAB, 4th Edition*. Upper Saddle River: Pearson Prentice Hall, 2004. ISBN: 0-13-065248-2.

[13] David Panek, Tamas Orosz, and Pavel Karban. "Artap: Robust Design Optimization Framework for Engineering Applications". In: *2019 Third International Conference on Intelligent Computing in Data Sciences (ICDS)* [online] (2019). [cit. 7.6.2021]. DOI: `10.1109/icds47004.2019.8942318`.

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

[15] The pandas development team. *pandas documentation*. [online]. © 2008 [cit. 3.6.2021]. URL: `https://pandas.pydata.org/docs/`.

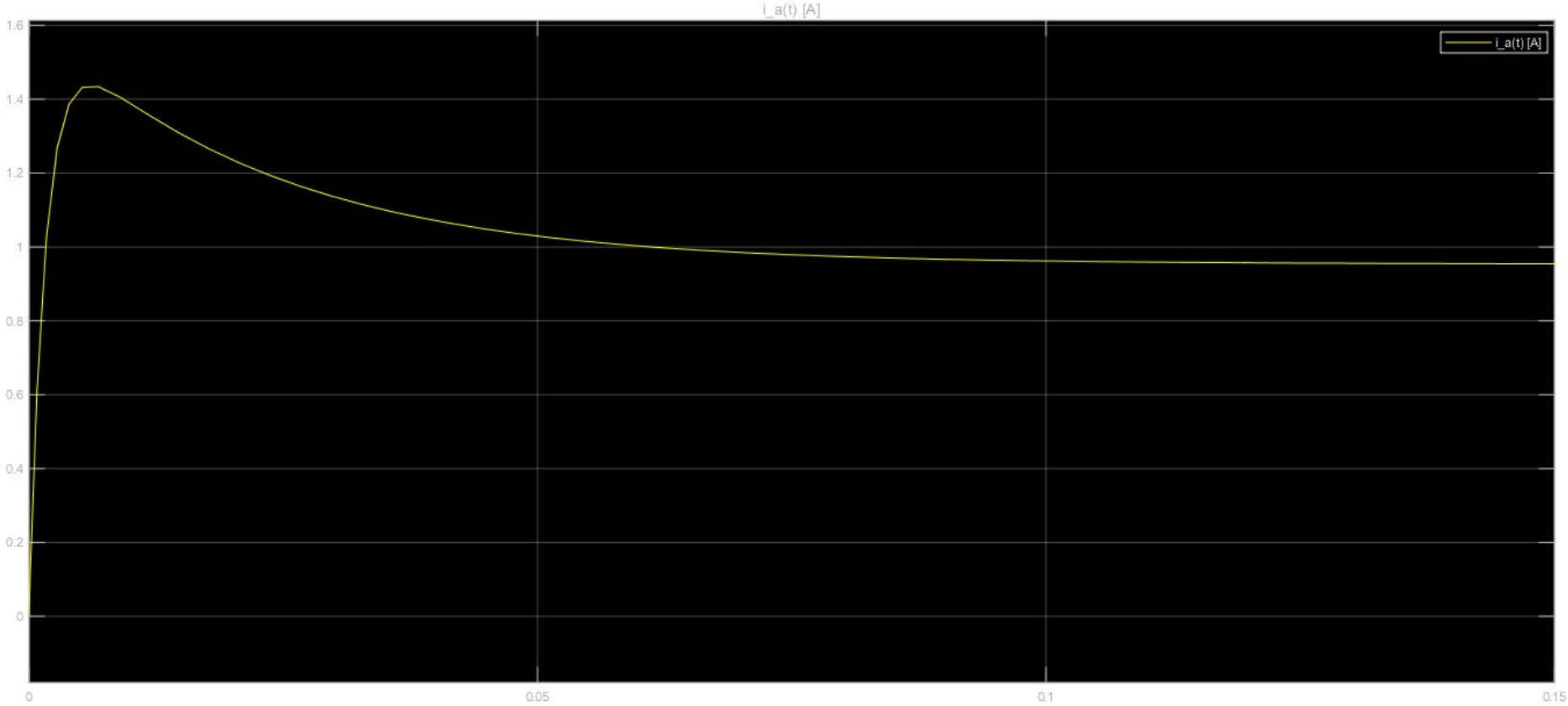
[16] František TŮMA. *Teorie řízení*. Plzeň: Západočeská univerzita, 1997. ISBN: 80-7082-341-0.

[17] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* [online] 17 (2020). [cit. 2.6.2021], pp. 261–272. ISSN: 1548-7105. DOI: `10.1038/s41592-019-0686-2`.

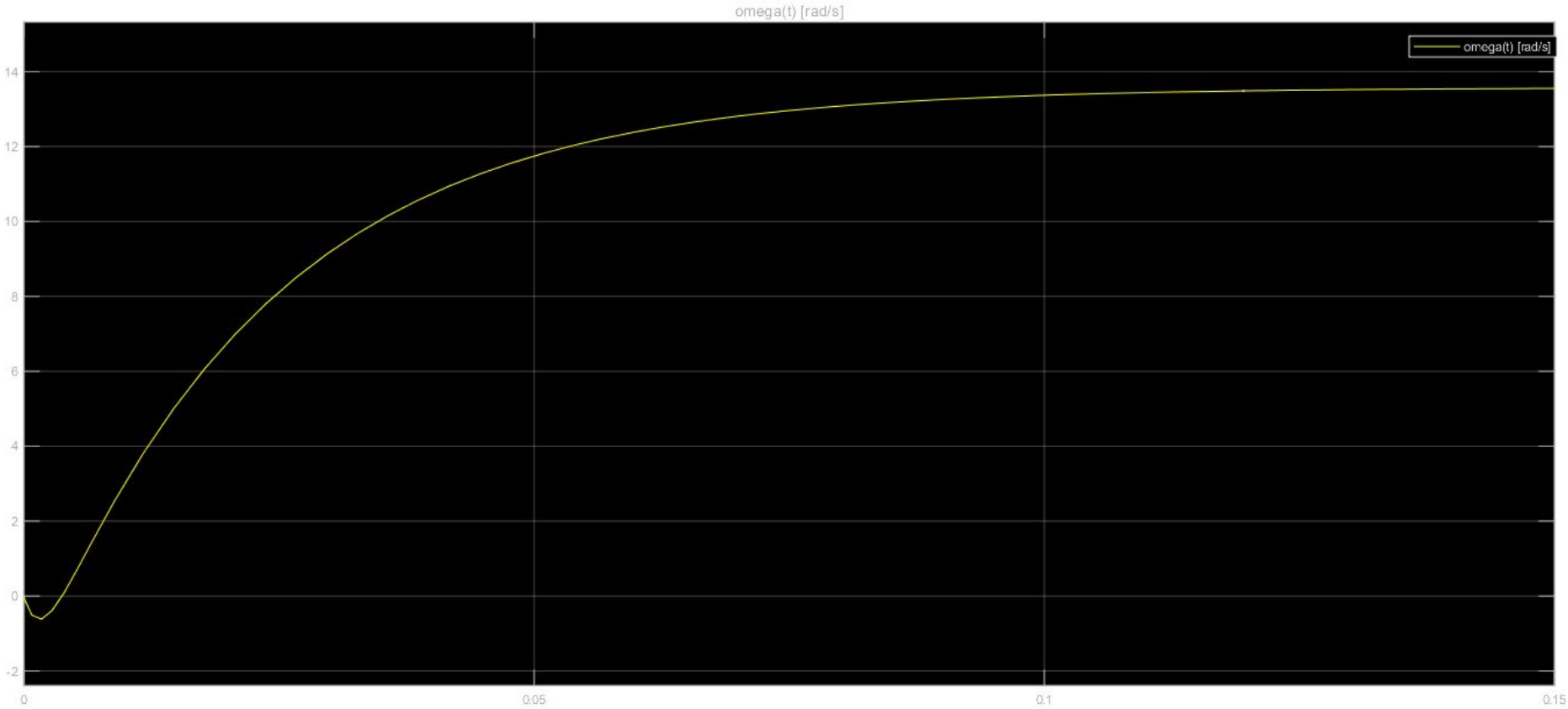
Příloha A

Výstupy ze simulace

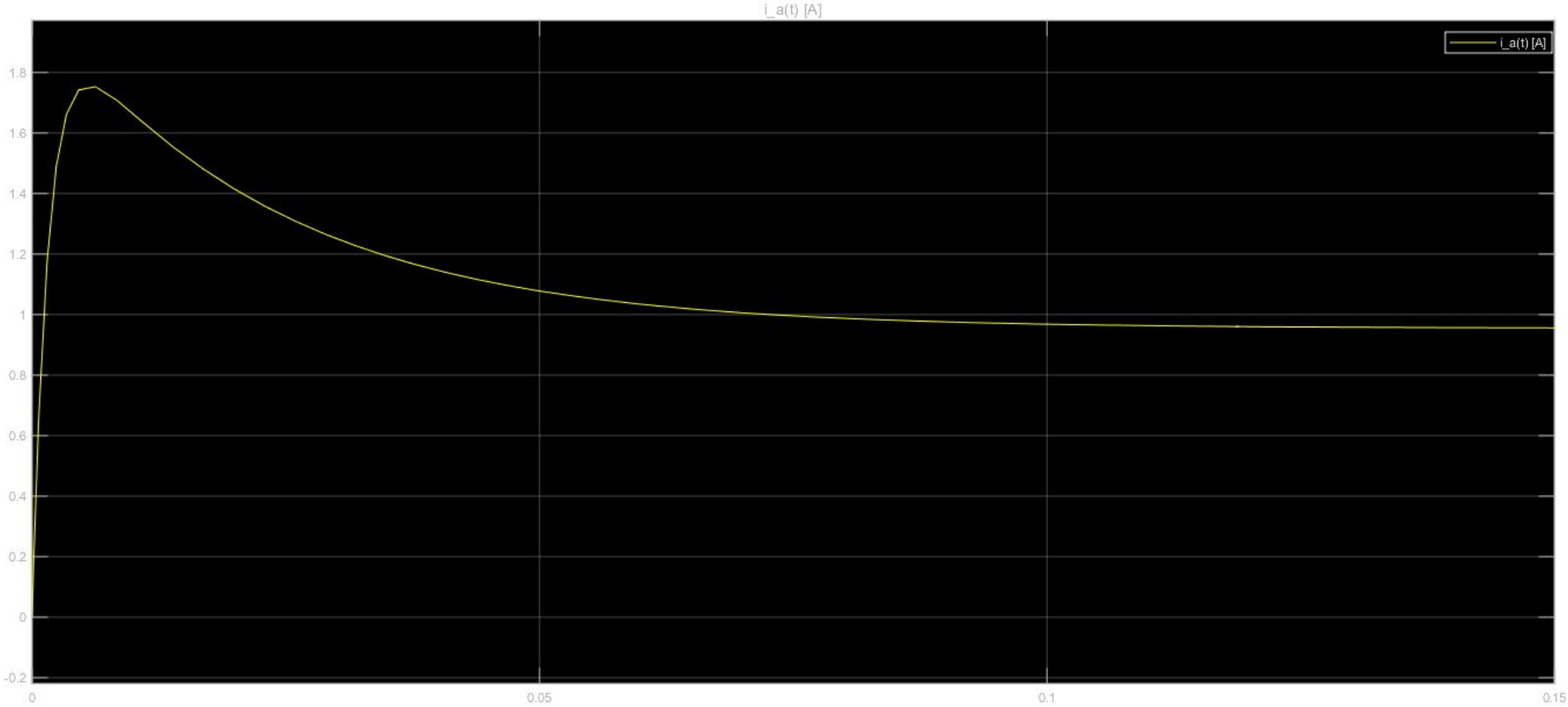
A.1 Grafy průběhů proudu a otáček stejnosměrného motoru s permanentními magnety – průběhy z nástroje MATLAB Simulink



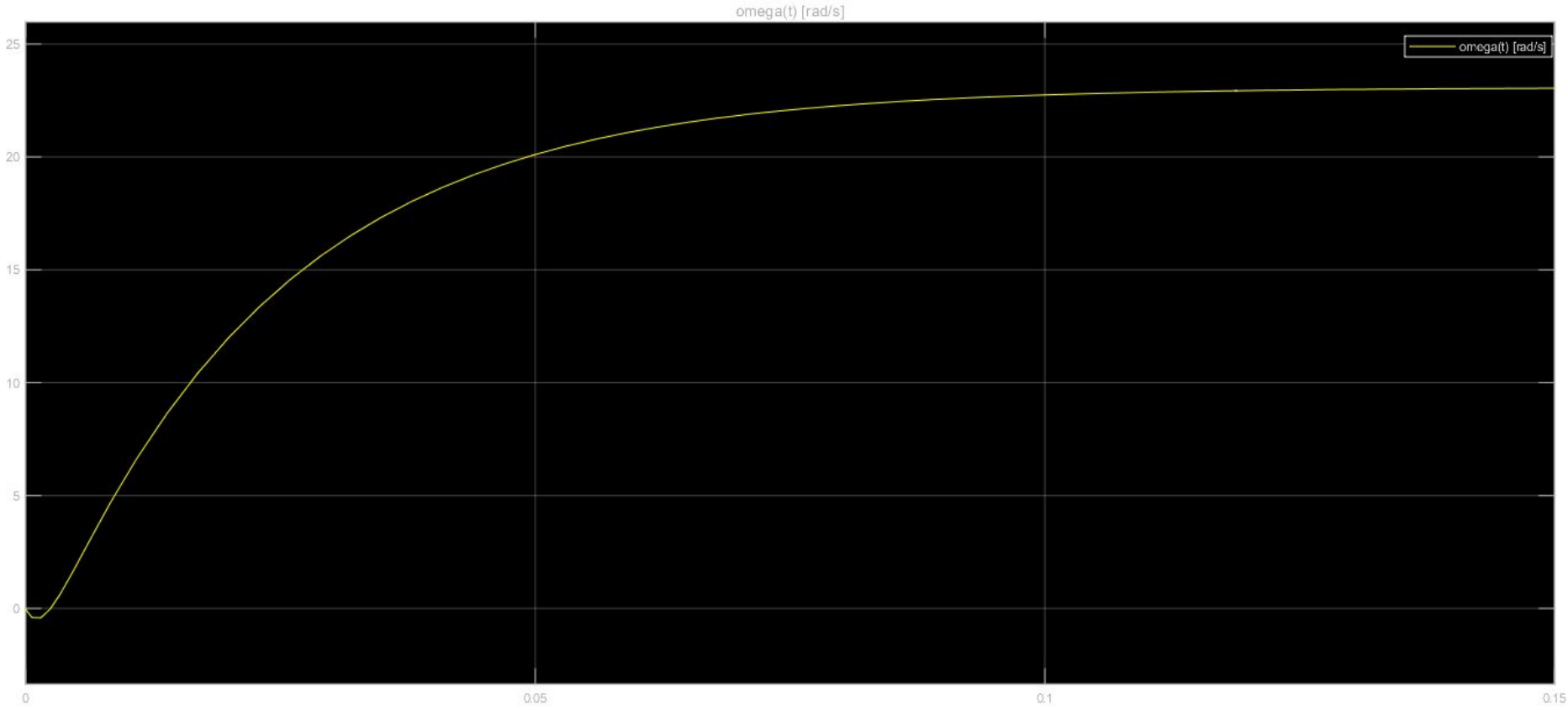
Obr. A.1: Průběh proudu $i_a(t)$ pro hodnoty vstupu $u_a = 4\,\mathrm{V}$ a $M_L = 0,1\,\mathrm{Nm}$



Obr. A.2: Průběh úhlové frekvence $\omega(t)$ pro hodnoty vstupu $u_a = 4\,\mathrm{V}$ a $M_L = 0,1\,\mathrm{Nm}$

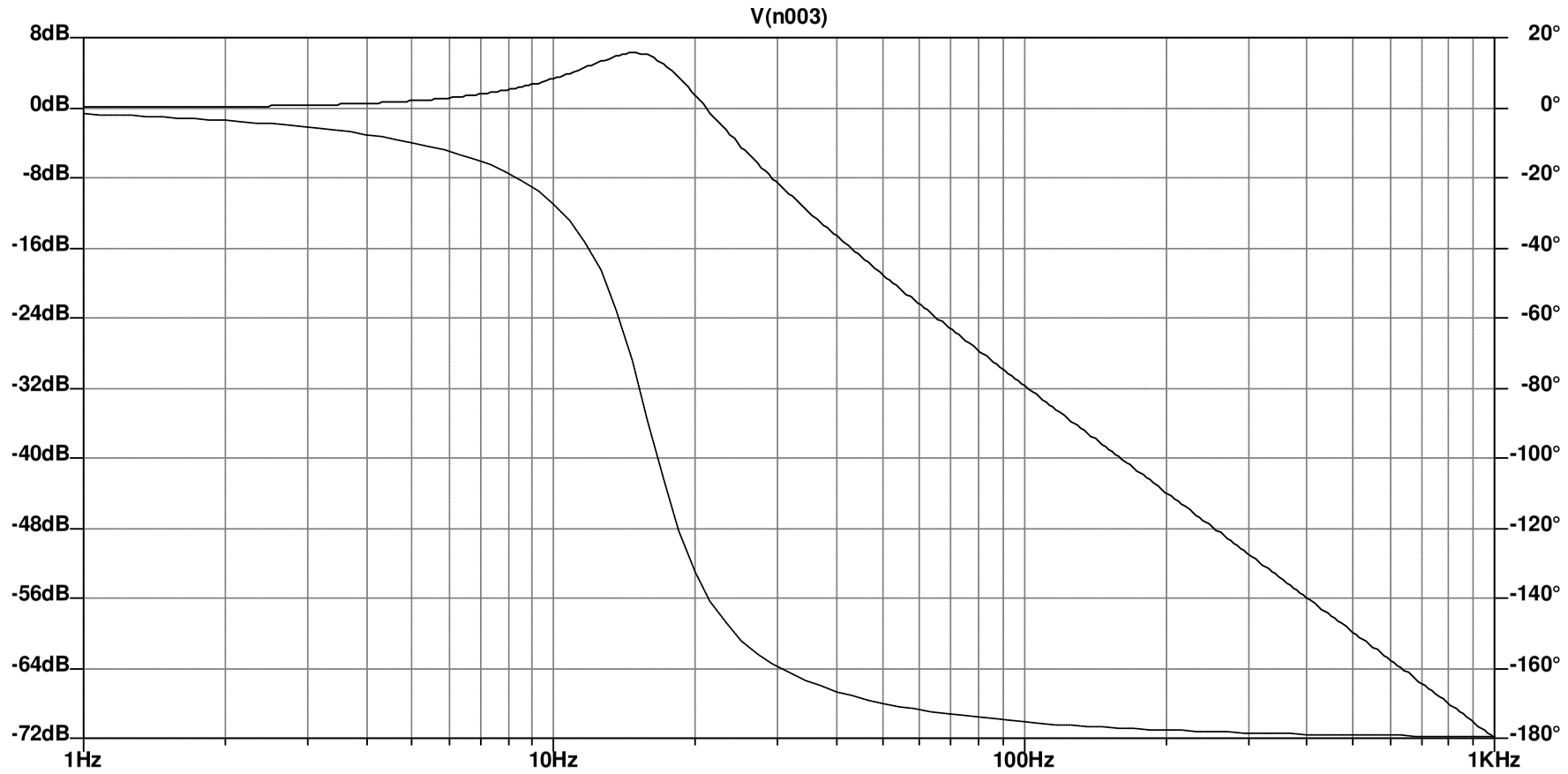


Obr. A.3: Průběh proudu $i_a(t)$ pro hodnoty vstupu $u_a = 5\,\mathrm{V}$ a $M_L = 0,1\,\mathrm{Nm}$



Obr. A.4: Průběh úhlové frekvence $\omega(t)$ pro hodnoty vstupu $u_a = 5\,\mathrm{V}$ a $M_L = 0,1\,\mathrm{Nm}$

A.2 Amplitudová a fázová frekvenční charakteristika sériového RLC obvodu – průběhy z programu LTspice



Obr. A.5: Amplitudová a fázová frekvenční charakteristika sériového RLC obvodu