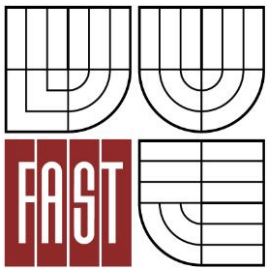
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA STAVEBNÍ
ÚSTAV STAVEBNÍ MECHANIKY

FACULTY OF CIVIL ENGINEERING
INSTITUTE OF STRUCTURAL MECHANICS

MODELOVÁNÍ VOLNÉHO PRUTU ZATÍŽENÉHO SLEDUJÍCÍM ZATÍŽENÍM

MODELLING OF FREE BEAM LOADED BY FOLLOWER LOAD

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN MAŠEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR FRANTÍK, Ph.D.
SUPERVISOR

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STAVEBNÍ

Studijní program	B3607 Stavební inženýrství
Typ studijního programu	Bakalářský studijní program s prezenční formou studia
Studijní obor	3647R013 Konstrukce a dopravní stavby
Pracoviště	Ústav stavební mechaniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student	Jan Mašek
Název	Modelování volného prutu zatíženého sledujícím zatížením
Vedoucí bakalářské práce	Ing. Petr Frantík, Ph.D.
Datum zadání bakalářské práce	30. 11. 2013
Datum odevzdání bakalářské práce	30. 5. 2014

V Brně dne 30. 11. 2013

.....	.....
prof. Ing. Drahomír Novák, DrSc.	prof. Ing. Rostislav Drochytka, CSc., MBA
Vedoucí ústavu	Děkan Fakulty stavební VUT

Podklady a literatura

Literatura dle vývoje a pokynů vedoucího práce.
Brepta, R., Půst, L., Turek, F.: Mechanické kmitání, Technický průvodce 71, nakladatelství Sobotáles,
Praha, 1994.

Zásady pro vypracování

Nastudování potřebných znalostí dle pokynů vedoucího práce. Zorientování se v problematice. Vytvoření numerických modelů a jejich aplikace.

Předepsané přílohy

.....

Ing. Petr Frantík, Ph.D.
Vedoucí bakalářské práce

Abstrakt

Cílem předkládané práce je vytvoření nelineárního dynamického modelu volného prutu zatíženého sledující silou. Reálným předobrazem modelu je štíhlá pružná raketa zatížená tahem reaktivního motoru. Vzhledem k povaze úlohy musí model umožňovat velké deformace. Dalšími požadavky na aplikovaný model je vystižení materiálového tlumení, tlumení vlivem interakce modelu s okolním prostředím a zavedení vlivu gravitačního pole. Model bude v budoucnu použit pro studium pokritického chování uvažované konstrukce, proto je jednou z priorit nízká časová náročnost simulace.

Vlastní vytvořená formulace numerického modelu bude implementována v programovacím jazyku Java. Pro zobrazení průběhu simulace a sledování stavových proměnných bude vytvořeno odpovídající grafické prostředí.

Správnost odvozeného modelu bude ověřena porovnáním vybraných hodnot s analytickým řešením.

Klíčová slova

Fyzikální diskretizace kontinua, metoda tuhých dílců, Lagrangeovská mechanika, pohybové rovnice, nekonzervativní zatížení, velké deformace.

Abstract

The aim of the presented thesis is to create a non-linear dynamical model of a free rod loaded by a follower force. The model is inspired by a slender flexible missile loaded by an end thrust. Because of the nature of the problem, the model has to be capable of large deflections. Another requirements on the model are to implement material the damping and the damping due to the interaction of the model with a surrounding medium and the influence of gravitational field. In the future, the model will be used for examination of the post-critical behavior of such construction. Therefore low computational demands of the simulation are required.

The derived formulation of the numerical model will be implemented using the Java programming language. For observation of the simulation process and for monitoring of the state variables, an appropriate graphic interface will be created.

The accuracy of the derived model will be verified by the comparison of selected values to the analytical solution.

Keywords

Physical discretization of continuum, rigid finite element method, Lagrangian mechanics, equations of motion, nonconservative load, large deflections.

Bibliografická citace práce

MAŠEK, Jan. Modelování volného prutu zatíženého sledujícím zatížením. Brno, 2014. 64 s., 15 s. příl. Bakalářská práce. Vysoké učení technické v Brně, Fakulta stavební, Ústav stavební mechaniky. Vedoucí práce Ing. Petr Frantík, Ph.D.

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci zpracoval samostatně, pod odborným vedením vedoucího práce Ing. Petra Frantíka, Ph.D., a že jsem uvedl všechny použité informační zdroje.

V Brně dne 30.5.2014

.....
podpis autora
Jan Mašek

Poděkování:

Rád bych poděkoval vedoucímu své bakalářské práce panu Ing. Petru Frantíkovi, Ph.D. za otevřený a konstruktivní odborný a přátelský osobní přístup v průběhu naší spolupráce. Především díky jeho nezištné pomoci jsem si osvojil schopnosti a způsob myšlení nezbytné k vypracování této práce.

Dále bych rád poděkoval vyučujícím a vědeckým pracovníkům Ústavu stavební mechaniky, kteří ve mně svým nadšením a pečlivou prací probudili podobné zalíbení v mechanice.

V neposlední řadě chci poděkovat svým rodičům, kteří mě vždy neúnavně podporovali a poskytnuli mi tak skvělé zázemí po celou dobu studia. Poděkování patří i mému bratrovi, bez jehož pomoci a podpory by pro mne studium bylo mnohem obtížnější.

V Brně dne 30.5.2014

.....
podpis autora
Jan Mašek

Obsah

1 Teoretický úvod **13**

1.1 Klasická mechanika . . . 13
1.1.1 Úvod a historie . . . 13
1.1.2 Pravoúhlá souřadná soustava . . . 14
1.1.3 Polární souřadná soustava . . . 16
1.1.4 Newtonovy pohybové zákony . . . 17
1.1.5 Zákon zachování hybnosti . . . 18
1.1.6 Raketový pohon . . . 19
1.2 Energie . . . 21
1.2.1 Potenciální energie . . . 21
1.2.2 Kinetická energie . . . 22
1.2.3 Zákon zachování energie . . . 22
1.3 Lagrangeovská mechanika . . . 24
1.3.1 Zobecněné souřadnice systému . . . 24
1.3.2 Konfigurační prostor, zobecněné rychlosti systému . . . 24
1.3.3 Lagrangeova funkce . . . 24
1.3.4 Virtuální posunutí, virtuální práce . . . 25
1.3.5 D'Alembertův princip . . . 26
1.3.6 Hamiltonův pricip nejmenší akce . . . 26
1.3.7 Lagrangeovy rovnice . . . 27
1.4 Pohybové rovnice a jejich řešení . . . 28
1.4.1 Pohybové rovnice . . . 28
1.4.2 Eulerova metoda . . . 28
1.4.3 Semi-implicitní Eulerova metoda . . . 29
1.4.4 Runge-Kuttova metoda . . . 30
1.5 Zdroje nelinearit . . . 31
1.5.1 Geometrická nelinearita . . . 31
1.5.2 Materiálová nelinearita . . . 32
1.5.3 Nelinearita způsobená okrajovými podmínkami . . . 32
1.6 Konzervativní a nekonzervativní síly . . . 33
1.6.1 Konzervativní síly . . . 33
1.6.2 Nekonzervativní síly . . . 33
1.7 Metoda tuhých dílců . . . 34

2 Odvození numerického modelu **36**
2.1 Úvod . . . 36
2.1.1 Princip odvození . . . 37
2.2 Energie . . . 37
2.2.1 Kinetická energie modelu . . . 37
2.2.2 Potenciální energie modelu . . . 40
2.3 Pohybové rovnice . . . 41
2.3.1 Maticový tvar . . . 44
2.4 Aplikovaný model volného prutu . . . 47
2.4.1 Sledující síla . . . 47
2.4.2 Materiálové tlumení . . . 48
2.4.3 Tlumení vlivem okolního prostředí . . . 49
2.4.4 Gravitační zrychlení . . . 52
2.4.5 Aplikovaný model . . . 52

3 Výsledky numerických simulací **53**
3.1 Vlastní frekvence prutu . . . 53
3.2 Konzola zatížená vlastní tíhou . . . 55
3.3 Kritická síla . . . 56
3.3.1 Volný prut . . . 56
3.3.2 Konzola . . . 58

4 Implementace modelu **59**
4.1 Implementace metod řešení dynamického systému . . . 59
4.1.1 Eulerova metoda . . . 60
4.1.2 Semi-implicitní Eulerova metoda . . . 60
4.1.3 Runge-Kuttova metoda . . . 61
4.2 Srovnání metod řešení dynamického systému . . . 65
4.3 Grafické prostředí modelu . . . 67

5 Závěr **70**
5.1 Výhledy . . . 71

Literatura **72**

Seznam symbolů **74**

Seznam příloh **76**

Přílohy **77**
A: Implementace modelu v jazyce Java . . . 77
B: Kompaktní disk . . . 90

Seznam obrázků

1.1 Zatřídění klasické mechaniky. 13
1.2 Poloha bodu P v souřadné soustavě. 14
1.3 Poloha bodu P vzhledem k pohybujicí se souřadné soustavě $x'y'z'$. 16
1.4 Poloha bodu P v polární souřadné soustavě. 16
1.5 Raketa zatížená tahem motoru. 19
1.6 Translační a rotační pružina. 22
1.7 Tuhé kyvadlo s jedním stupněm volnosti. 22
1.8 Trajektorie vývoje dynamického systému. 27
1.9 Konzola zatížená osamělou silou. 31
1.10 Nelineární pracovní diagram materiálu. 32
1.11 Příklady některých nelineárních vazeb. 33
1.12 Znázornění konzervativních vlastností gravitační síly. 34
1.13 Pružná štíhlá raketa zatížená tahem motoru. 34
1.14 Idealizace konstrukce metodou tuhých dílců. 34
1.15 Varianty kontaktních kumulátorů přetvoření. 35

2.1 Model volného prutu zatíženého sledující silou. 36
2.2 Model volného prutu. 37
2.3 První dílec modelu. 38
2.4 Působení sledující síly. 47
2.5 Odvození závislosti útlumu na směru pohybu modelu. 50

3.1 Konvergence hodnoty první vlastní frekvence volného prutu. 54
3.2 Závislost výpočetní doby na počtu dílců. 54
3.3 Konzola zatížená vlastní tíhou. 55
3.4 Konvergence hodnoty kritické síly volného prutu. 57
3.5 Nárůst výpočetní doby s počtem stupňů volnosti. 57
3.6 Konvergence hodnoty kritické síly konzoly. 58

4.1 Maximální časový krok v závislosti na počtu dílců a metodě. 65
4.2 Výpočetní doba v závislosti na počtu dílců a metodě. 66
4.3 Základní grafické prostředí modelu. 68
4.4 Zobrazení stavové proměnné v závislosti na čase. 69
4.5 Zobrazení stavových proměnných ve vzájemné závislosti. 69
4.6 Zobrazení proměnných stejného typu. 69

Seznam tabulek

3.1 Srovnání vypočtených výsledků s analytickým řešením vlastních frekvencí. . . 53
3.2 Konvergence hodnoty první vlastní frekvence a odpovídající výpočetní doba. 54
3.3 Porovnání výsledků průhybu a pootočení volního konce konzoly. 55
3.4 Výsledky simulací speciálního modelu volného prutu. 56
3.5 Výsledky simulací provedených v aplikaci FyDiK. 56
3.6 Výsledky simulací speciálního modelu konzoly. 58

4.1 Maximální časový krok v závislosti na počtu dílců a metodě. 65
4.2 Výpočetní doba v závislosti na počtu dílců a metodě. 66

Kapitola 1

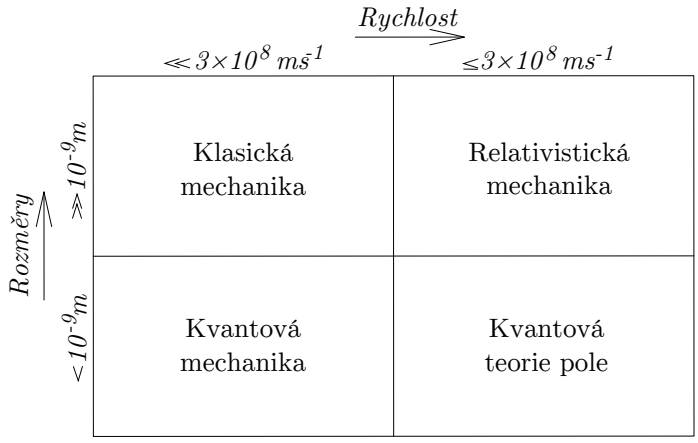
Teoretický úvod

V úvodní části práce je zapotřebí pohovořit o teoriích a pojmech, jenž jsou v dalších částech práce použity. Přestože lze u čtenáře předpokládat znalosti fyziky, matematiky a stavební mechaniky, je na místě mu nabídnout kompaktní teoretický úvod, který navíc předejde případnému nedorozumění.

1.1 Klasická mechanika

1.1.1 Úvod a historie

Mechanika je vědní obor zabývající se studiem pohybu těles v prostoru a čase. Její součást, klasická mechanika, zkoumá pohyb těles a objektů v makroskopickém měřítku [1] při rychlosti výrazně nižší než rychlost světla, viz. obrázek 1.1.



Obrázek 1.1: Zatřídění klasické mechaniky.

S mechanickými jevy se člověk potýká od samých počátků vlastní existence. Například statická rovnováha, princip páky, kladky, nakloněná rovina a další jednoduché stroje, byly předmětem bádání vědců a vynálezců již ve starověkém Řecku. Jedním z nejvýznamnějších byl zcela jistě Archimédes, který navíc definoval pojmy jako těžiště a statický moment.

[1] Rozměry tělesa jsou dosti velké pro pozorování pouhým okem.

Klasická mechanika v dnešní podobě se začala utvářet v 16. století s nástupem tzv. moderní empirické vědy, jenž se opírá především o experiment a měření. Zde je vhodné poznamenat, že patrně mnohem větší nesnází než nevyvinutá technologie byl pro tehdejší vědce mnohdy zatrvrzelý odpor společnosti. Mezi největší osobnosti té doby patří například Galileo Galilei a Blaise Pascal. Pravým otcem klasické mechaniky je potom Isaac Newton, jenž v roce 1687 ve své práci Philosophiae Naturalis Principia Mathematica publikoval zákon všeobecné gravitace a tři pohybové zákony, viz [18].

1.1.2 Pravoúhlá souřadná soustava

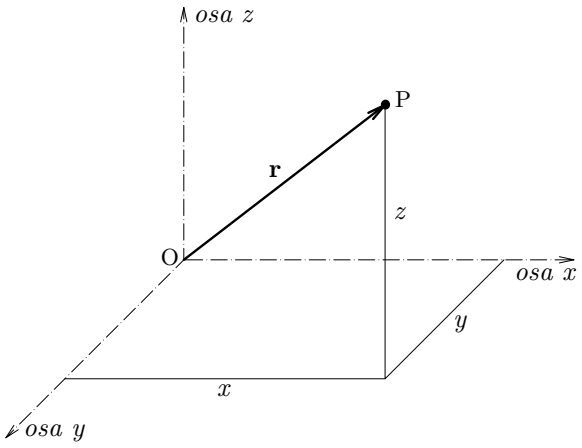
Poloha bodu v pravoúhlé souřadné soustavě

Každému bodu P v trojrozměrném prostoru může být přiřazen vektor $\mathbf{r}$, který určuje jeho polohu vzhledem ke zvolenému počátku O souřadné soustavy xyz, viz obrázek 1.2. Vektor $\mathbf{r}$ lze formulovat jako:

$$\mathbf{r} = x\,\widehat{\mathbf{i}} + y\,\widehat{\mathbf{j}} + z\,\widehat{\mathbf{k}}, \tag{1.1}$$

kde x, y, z jsou reálná čísla, $\widehat{\mathbf{i}}, \widehat{\mathbf{j}}\, a\, \widehat{\mathbf{k}}$ jsou potom tzv. jednotkové vektory, které reprezentují osy pravoúhlého souřadného systému. Pro tyto vektory platí:

$$\widehat{\mathbf{i}} = \begin{bmatrix}1\\0\\0\end{bmatrix},\ \widehat{\mathbf{j}} = \begin{bmatrix}0\\1\\0\end{bmatrix},\ \widehat{\mathbf{k}} = \begin{bmatrix}0\\0\\1\end{bmatrix}. \tag{1.2}$$



Obrázek 1.2: Poloha bodu P v souřadné soustavě.

Rychlost bodu v pravoúhlé souřadné soustavě

Rychlost pohybu bodu P v prostoru lze chápat jako velikost okamžité změny vektoru $\mathbf{r}$ popisujícího jeho polohu. Vektor rychlosti $\mathbf{v}$ lze potom zapsat jako:

$$\mathbf{v} = \frac{\mathrm{d}\mathbf{r}}{\mathrm{d}t} = \lim_{t\to 0} \frac{\Delta\mathbf{r}}{\Delta t}, \tag{1.3}$$

kde $\Delta\mathbf{r}$ je změna vektoru $\mathbf{r}$ za časový okamžik Δt, tedy:

$$\Delta\mathbf{r} = \mathbf{r}(t + \Delta t) - \mathbf{r}(t). \tag{1.4}$$

Před dosazením do výrazu (1.3) připomeňme, že obecně platí:

$$\frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{a} + \mathbf{b}) = \frac{\mathrm{d}\mathbf{a}}{\mathrm{d}t} + \frac{\mathrm{d}\mathbf{b}}{\mathrm{d}t}. \tag{1.5}$$

Potom je již možné vyjádřit vektor rychlosti $\mathbf{v}$:

$$\mathbf{v} = \frac{\mathrm{d}\mathbf{r}}{\mathrm{d}t} = \frac{\mathrm{d}x}{\mathrm{d}t}\,\widehat{\mathbf{i}} + \frac{\mathrm{d}y}{\mathrm{d}t}\,\widehat{\mathbf{j}} + \frac{\mathrm{d}z}{\mathrm{d}t}\,\widehat{\mathbf{k}}, \tag{1.6}$$

čemuž odpovídá:

$$\mathbf{v} = v_x\,\widehat{\mathbf{i}} + v_z\,\widehat{\mathbf{j}} + v_y\,\widehat{\mathbf{k}}. \tag{1.7}$$

Pro další potřeby ještě poznamenejme, že tedy platí:

$$v_x = \frac{\mathrm{d}x}{\mathrm{d}t},\ \ v_y = \frac{\mathrm{d}y}{\mathrm{d}t},\ \ v_z = \frac{\mathrm{d}z}{\mathrm{d}t}. \tag{1.8}$$

Pohyblivá souřadná soustava

Při pozorování chování pohybujícího se tělesa může být výhodné jeho polohu nebo polohu jeho části vztahovat k vhodně umístěné pohyblivé souřadné soustavě.

Uvažujme pohybující se pravoúhlou souřadnou soustavu $x'y'z'$ a soustavu xyz, která je v klidu. Polohu bodu P vzhledem k počátku O soustavy xyz určuje vektor $\mathbf{r}$, polohu počátku O' soustavy $x'y'z'$ potom vektor $\mathbf{s}$, viz obrázek 1.3. Polohu bodu P vůči pohybující se soustavě $x'y'z'$ lze popsat vektorem $\mathbf{r}'$:

$$\mathbf{r}' = \mathbf{r} - \mathbf{R}(t). \tag{1.9}$$

Vektor rychlosti pohybu $\mathbf{v}'$ bodu P, tedy změnu jeho polohy v čase, lze vyjádřit snadno:

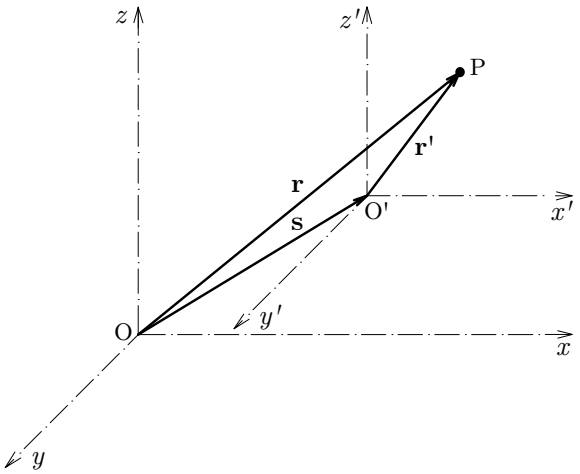
$$\mathbf{v}' = \frac{\mathrm{d}\mathbf{r}'}{\mathrm{d}t} = \frac{\mathrm{d}\mathbf{r}}{\mathrm{d}t} - \frac{\mathrm{d}\mathbf{R}(t)}{\mathrm{d}t}. \tag{1.10}$$

Zkráceně tedy:

$$\mathbf{v}' = \mathbf{v} - \mathbf{V}, \tag{1.11}$$

kde $\mathbf{V}$ je vektor rychlosti pohybu počátku O' soustavy $x'y'z'$ vzhledem k soustavě xyz.

Při řešení úlohy nejen v mechanice je vhodné zvolit počátek souřadného systému tak, aby bylo pozorování chování modelu co možná nejsnazší. Jak bude ukázáno později, v předkládaném modelu je použita stacionární souřadná soustava pro sledování polohy modelu v prostoru a zároveň pohyblivá souřadná soustava, jejíž výhodné použití bude vysvětleno později.

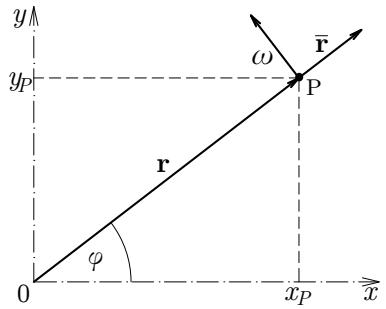


Obrázek 1.3: Poloha bodu P vzhledem k pohybující se souřadné soustavě $x'y'z'$.

1.1.3 Polární souřadná soustava

Jak bylo ukázáno, v pravoúhlé souřadné soustavě lze snadno popsat polohu i rychlost bodu. U některých úloh je ale výhodnější zvolit polární souřadnou soustavu, jenž může být přehlednější a řešení může značně zjednodušit.

Namísto pravoúhlých souřadnic je poloha bodu P v polární souřadné soustavě určena úhlem φ a průvodičem $\mathbf{r}$, viz obrázek 1.4.



Obrázek 1.4: Poloha bodu P v polární souřadné soustavě.

Vektor ω je vektorem úhlové rychlosti kolmo na průvodič $\mathbf{r}$. Vektor $\bar{\mathbf{r}}$ potom znázorňuje změnu délky průvodiče $\mathbf{r}$ v čase při konstantním úhlu φ. Lze tedy psát:

$$\begin{aligned} |\omega| &= \frac{\mathrm{d}\varphi}{\mathrm{d}t}, \\ |\bar{\mathbf{r}}| &= \frac{\mathrm{d}\,|\mathbf{r}|}{\mathrm{d}t}. \end{aligned} \tag{1.12}$$

Rychlost pohybu bodu P v rovině lze popsat vektorem $\mathbf{v}$, který vznikne složením vektorů ω a $\bar{\mathbf{r}}$, tedy:

$$\mathbf{v} = \omega + \bar{\mathbf{r}}. \tag{1.13}$$

Pro převod mezi pravoúhlými a polárními souřadnicemi bodu P lze snadno odvodit vztahy:

$$\begin{aligned} x_P &= |\mathbf{r}| \cos\varphi, \\ y_P &= |\mathbf{r}| \sin\varphi, \end{aligned} \tag{1.14}$$

přičemž:

$$\begin{aligned} |\mathbf{r}| &= \sqrt{x_P^2 + y_P^2}, \\ \varphi &= \arctan\left(\frac{x_P}{y_P}\right). \end{aligned} \tag{1.15}$$

1.1.4 Newtonovy pohybové zákony

Jak již bylo zmíněno, Isaac Newton v roce 1687 vyslovil tři pohybové zákony, jenž jsou základem klasické mechaniky a dynamiky. Zákony popisují vztahy mezi pohybem tělesa a silami na těleso působícími.

První Newtonův zákon

První Newtonův zákon, nazývaný také jako Zákon setrvačnosti, je možné v překladu zapsat:

"Jestliže na těleso nepůsobí žádné vnější síly nebo je výslednice vnějších sil nulová, potom těleso setrvává v klidu nebo v rovnoměrném přímočarém pohybu."

Lze si všimnout, že zákon zmiňuje pouze vnější síly, které působí na těleso. Je tedy zřejmé, že vnitřní síly mezi částmi tělesa nemají na pohyb tělesa jako celku, přesněji na pohyb jeho těžiště, vliv. Matematicky lze Zákon setrvačnosti formulovat například takto:

$$\sum \mathbf{F}_{ext} = 0 \Leftrightarrow \frac{\mathrm{d}v}{\mathrm{d}t} = 0. \tag{1.16}$$

Druhý Newtonův zákon

Druhý Newtonův zákon, známý také jako Zákon síly, říká:

"Působí-li na těleso síla, pak se těleso pohybuje se zrychlením, které je přímo úměrné působící síle a nepřímo úměrné hmotnosti tělesa."

Tvrzení zákona lze matematicky zapsat jako:

$$\mathbf{a} = \frac{\mathbf{F}_{ext}}{m}, \tag{1.17}$$

kde $\mathbf{a}$ je vektor zrychlení tělesa a m je hmotnost tělesa. Je zřejmé, že výraz (1.17) můžeme přepsat do známé podoby:

$$\mathbf{F}_{ext} = m\,\mathbf{a}. \tag{1.18}$$

Rovnici (1.18) lze ještě dále upravit:

$$\mathbf{F}_{ext} = m\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = \frac{\mathrm{d}(m\mathbf{v})}{\mathrm{d}t} = \frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t}, \tag{1.19}$$

tedy můžeme říci, že síla $\mathbf{F}$ je rovna časové změně hybnosti $\mathbf{p}$.

Třetí Newtonův zákon

Prvé dva Newtonovy zákony se zabývaly odezvou tělesa na působení vnějších sil. Třetí Newtonův zákon, nazývaný také jako Zákon akce a reakce, hovoří o vzájemném působení dvou těles:

"Proti každé akci vždy působí stejná reakce; jinak: vzájemná působení dvou těles jsou vždy stejně velká a míří na opačné strany."

Třetí pohybový zákon vyjádřený matematicky potom vypadá následovně:

$$\mathbf{F}_{A,B} = -\mathbf{F}_{B,A}, \tag{1.20}$$

kde $\mathbf{F}_{A,B}$ je síla vyvíjená tělesem A na těleso B a $\mathbf{F}_{B,A}$ je síla, kterou těleso B zpětně působí na těleso A. Tedy:

$$\mathbf{F}_{A,B} + \mathbf{F}_{B,A} = 0\,; \quad \sum F_{ext} = 0. \tag{1.21}$$

1.1.5 Zákon zachování hybnosti

Uvažujme uzavřenou soustavu[2] obsahující dvě na sebe navzájem silově působící hmotná tělesa A a B. Podle Třetího Newtonova zákona platí, že působící síly mají stejnou velikost a opačný směr. Rovnici (1.20) lze dále upravit:

$$\begin{aligned} m_A\mathbf{a}_A &= -m_B\mathbf{a}_B, \\ m_A\frac{\mathrm{d}\mathbf{v}_A}{\mathrm{d}t} &= -m_B\frac{\mathrm{d}\mathbf{v}_B}{\mathrm{d}t}, \\ \frac{\mathrm{d}\mathbf{p}_A}{\mathrm{d}t} &= -\frac{\mathrm{d}\mathbf{p}_B}{\mathrm{d}t}, \\ \frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{p}_A + \mathbf{p}_B) &= 0, \end{aligned} \tag{1.22}$$

z čehož je zřejmé, že *je-li silová výslednice soustavy nulová, celková hybnost soustavy zůstává konstantní*, viz [13].

Pro soustavu obsahující n hmotných bodů analogicky platí:

$$\mathbf{P} = \sum_{i=1}^{n}\mathbf{p}_i = \sum_{i=1}^{n} m_i\mathbf{v}_i, \tag{1.23}$$

$$\dot{\mathbf{P}} = \sum_{i=1}^{n}\dot{\mathbf{p}}_i = \sum_{i=1}^{n} m_i\frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} = \sum_{i=1}^{n} m_i\mathbf{a}_i = \sum \mathbf{F}_i, \tag{1.24}$$

$$\sum_{i=1}^{n}\mathbf{F}_i = 0 \Leftrightarrow \dot{\mathbf{P}} = 0 \Leftrightarrow \mathbf{P} = konst, \tag{1.25}$$

kde $\mathbf{p}_i$ jsou dílčí hybosti hmotných bodů soustavy a $\mathbf{P}$ je celková hybnost soustavy.

[2]Uzavřená soustava těles nebo hmotných bodů je taková soustava, kde na sebe obsažené entity mohou působit silově, popř. si vyměňovat energii, ale nemohou s okolím vyměňovat hmotu.

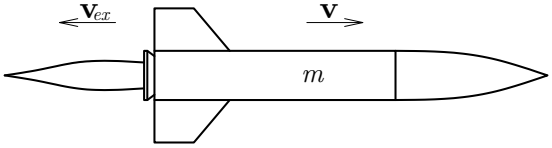
1.1.6 Raketový pohon

Předlohou předkládaného dynamického modelu je štíhlá raketa zatížená tahem raketového motoru, proto je na místě zmínit i princip fungování raketového pohonu. Raketový pohon se řadí mezi základní typy reaktivního pohonu. Dalšími typy reaktivního pohonu jsou:

- proudový pohon,
- náporový pohon,
- pulzační pohon,
- iontový pohon.

Princip reaktivního pohonu vychází z Třetího Newtonova zákona a Zákona zachování hybnosti, viz [19]. Reaktivní motory vyvíjí tah díky tryskání média z hnací trysky motoru. V případě raketového motoru se jedná o spaliny vznikající hořením paliva.

Uvažujme raketu o hmotnosti m pohybující se rychlostí $\mathbf{v}$, viz obrázek 1.5. Rychlost unikajících spalin označme jako $\mathbf{v}_{ex}$.



Obrázek 1.5: Raketa zatížená tahem motoru.

Při hoření paliva a tryskání spalin raketa neustále ztrácí hmotnost. Hybnost rakety $\mathbf{p}$ v čase t lze zapsat jako:

$$\mathbf{p}(t) = m(t)\mathbf{v}(t). \tag{1.26}$$

V čase $(t + \mathrm{d}t)$ se hmotnost rakety sníží na $(m(t) - dm)$. Hybnost rakety se změní na:

$$\mathbf{p}(t + \mathrm{d}t) = (m(t) - \mathrm{d}m)\,(\mathbf{v}(t) + \mathrm{d}\mathbf{v})\,. \tag{1.27}$$

Ztracené spaliny mají hmotnost dm a rychlost $(\mathbf{v}(t) - \mathbf{v}_{ex})$. Celková hybnost rakety a ztracených spalin $\mathbf{P}$ v čase $(t + \mathrm{d}t)$ je potom:

$$\mathbf{P}(t + \mathrm{d}t) = (m(t) - \mathrm{d}m)\,(\mathbf{v}(t) + \mathrm{d}\mathbf{v}) + \mathrm{d}m(\mathbf{v}(t) - \mathbf{v}_{ex}). \tag{1.28}$$

Změna celkové hybnosti $\mathrm{d}\mathbf{P}$ má tedy tvar:

$$\mathrm{d}\mathbf{P} = \mathbf{P}(t + \mathrm{d}t) - \mathbf{P}(t) = m\,\mathrm{d}\mathbf{v} + \mathrm{d}m\,\mathbf{v}_{ex}. \tag{1.29}$$

Pokud na soustavu nepůsobí žádné vnější síly, její hybnost zůstává konstantní a její změna v čase $\mathrm{d}\mathbf{P}$ je nulová. Lze tedy tvrdit, že:

$$m\,\mathrm{d}\mathbf{v} = -\mathrm{d}m\,\mathbf{v}_{ex}. \tag{1.30}$$

Pokud vydělíme obě strany rovnice (1.30) členem $\mathrm{d}t$, získáme výraz:

$$m\,\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -\frac{\mathrm{d}m}{\mathrm{d}t}\,\mathbf{v}_{ex}. \tag{1.31}$$

Tedy:

$$m\,\mathbf{a} = -\dot{m}\,\mathbf{v}_{ex}, \tag{1.32}$$

kde $(-\dot{m})$ je rychlost, s jakou raketa ztrácí hmotnost hořícího paliva. Podle Druhého Newtonova zákona, viz rovnice (1.18), lze levou stranu výrazu (1.32) označit za hnací sílu, tedy tah raketového motoru $\mathbf{F}_{thrust}$:

$$\mathbf{F}_{thrust} = -\dot{m}\,\mathbf{v}_{ex}. \tag{1.33}$$

Separace proměnných v rovnici (1.30) dále vede na výraz:

$$\mathrm{d}\mathbf{v} = -\frac{\mathrm{d}m}{m}\,\mathbf{v}_{ex}. \tag{1.34}$$

Pokud je rychlost tryskání spalin $\mathbf{v}_{ex}$ konstantní, integrací obdržíme výsledek:

$$\mathbf{v} - \mathbf{v}_0 = \mathbf{v}_{ex}\,\ln\left(\frac{m_0}{m}\right), \tag{1.35}$$

kde $\mathbf{v}_0$ je počáteční rychlost rakety a m_0 je počáteční celková hmotnost rakety.

Bližší pohled na výraz (1.35) ukazuje rychlostní omezení raketového pohonu. Poměr m_0/m je maximální ve chvíli, kdy je veškeré palivo spáleno a m obsahuje pouze hmotnost rakety a nákladu. I v krajním případě, kdy by počáteční hmotnost rakety byla tvořena z 90% hmotností paliva, je nejvyšší poměr m_0/m roven 10, tedy maximální získaná rychlost $(\mathbf{v} - \mathbf{v}_0)$ bude přibližně $2.3\,\mathbf{v}_{ex}$.

Z uvedeného vyplývá, že je vhodné maximalizovat rychlost tryskání spalin $\mathbf{v}_{ex}$ a dále s výhodou navrhnout raketu o více stupních, jež mohou být v průběhu letu odhozeny, čímž se sníží celková hmotnost rakety ve vztahu ke stupňům následujícím.

1.2 Energie

Energie je jednou ze základních fyzikálních veličin. Energie je skalární veličinou, je tedy určena pouze velikostí. V přírodě lze energii popsat v mnoha podobách, viz [19]. Uveďme ty hlavní:

- kinetická energie,
- potenciální energie,
  - gravitační potenciální energie,
  - potenciální energie vzniklá pružnou deformací,
  - tlaková potenciální energie,
- elektrická energie,
- magnetická energie,
- vnitřní energie,
  - tepelná energie,
  - jaderná energie,
  - chemická energie.

1.2.1 Potenciální energie

Potenciální energie E_p (v zahraniční literatuře označovaná jako V) je energie, jenž se může kumulovat v systému, viz [19] a [20]. Příkladem může být gravitační potenciální energie, kdy je potenciální energie úměrná hmotnosti tělesa m, gravitačnímu zrychlení g a výšce těžiště tělesa nad srovnávací rovinou h, tedy:

$$E_p = m\,g\,h. \tag{1.36}$$

Další možností kumulace energie je pružná deformace tělesa. Při protažení nebo stlačení translační pružiny tuhosti k a počáteční délky l o délku Δl, viz obrázek 1.6, působí na pružinu síla F_{ts}:

$$F_{ts} = k\,\Delta l. \tag{1.37}$$

Pro potenciální energii translační pružiny $E_{p,ts}$ platí:

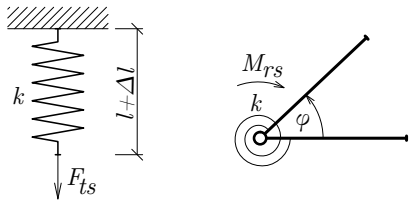
$$E_{p,ts} = \int_0^{\Delta l} F_{ts}\,dl = \frac{1}{2}k\,\Delta l^2. \tag{1.38}$$

Podobně v případě rotační pružiny, viz obrázek 1.6, spojující dva tuhé dílce při deformaci o úhel φ působí moment M_{rs}:

$$M_{rs} = k\,\varphi. \tag{1.39}$$

Výraz pro potenciální energii $E_{p,rs}$ nashromážděnou v rotační pružině má potom tvar:

$$E_{p,rs} = \int_0^{\varphi} M_{rs}\,d\varphi = \frac{1}{2}k\,\varphi^2. \tag{1.40}$$



Obrázek 1.6: Translační a rotační pružina.

1.2.2 Kinetická energie

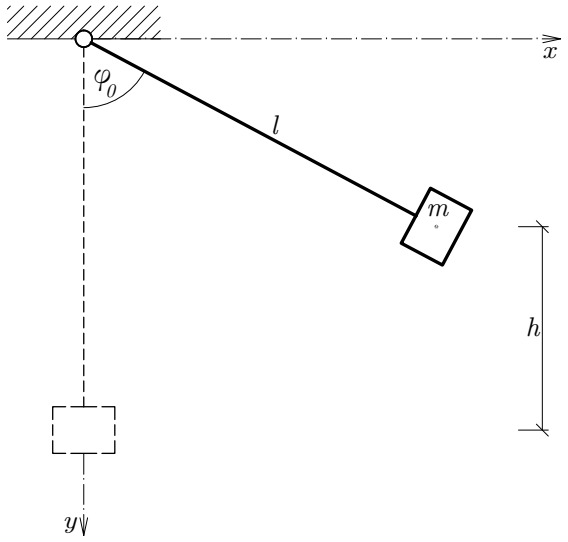
Kinetická energie je možná nejběžnější forma energie. Pro hmotný bod o hmotnosti m a rychlosti v lze výraz pro jeho kinetickou energii E_k (v zahraniční literatuře označovanou také T) formulovat jako:

$$E_k = \frac{1}{2} m\, v^2. \tag{1.41}$$

1.2.3 Zákon zachování energie

Zákon zachování energie říká, že celková energie izolované soustavy[3] zůstává konstantní. Energie tedy nemůže být nijak vyrobena ani zničena, pouze se může přeměňovat z jedné formy do jiné.

Uvažujme kyvadlo o jednom stupni volnosti, viz obrázek 1.7, s nehmotným tuhým závěsem délky l, na jehož konci je soustředěná hmota m. Na hmotu m působí gravitační zrychlení o velikosti g.



Obrázek 1.7: Tuhé kyvadlo s jedním stupněm volnosti.

[3]Izolovaná soustava těles nebo hmotných bodů je taková soustava, kde na sebe obsažené entity mohou působit silově, ale nemohou s okolím vyměňovat energii nebo hmotu.

Kyvadlo je v čase $t_0 = 0$ vychýleno od svislého rovnovážného stavu o úhel φ_0. Jak bylo uvedeno, jeho potenciální energie $E_p(t_0)$ má velikost:

$$E_p(t_0) = m\,g\,h = m\,g\,l\,\sin\varphi(t_0) \tag{1.42}$$

Rychlost závěsu kyvadla v lze vyjádřit pomocí jeho úhlové rychlosti jako:

$$v = l\,\omega, \tag{1.43}$$

kde ω je úhlová rychlost kyvadla, tedy:

$$\omega = \frac{\mathrm{d}\varphi}{\mathrm{d}t}. \tag{1.44}$$

Pro kinetickou energii kyvadla E_k platí vztah (1.41). Po úpravě získáváme výraz:

$$E_k = \frac{1}{2} m\,l^2\omega^2. \tag{1.45}$$

V čase t_0 má kyvadlo nulovou úhlovou rychlost, jeho kinetická energie E_k je proto nulová. Úhlová výchylka kyvadla φ (a tedy výška h) je maximální, což znamená i maximální hodnotu potenciální energie.

Celková energie kyvadla E je prostým součtem jeho potenciální a kinetické energie:

$$E = E_p + E_k = konst. \tag{1.46}$$

Pro energii kyvadla v čase t_0 tedy platí:

$$E = E_p(t_0) + E_k(t_0) = m\,g\,l\,\sin\varphi(t_0) + 0. \tag{1.47}$$

Po uvolnění kyvadla začne výchylka φ i výška kyvadla h klesat, naopak rychlost kyvadla bude růst. V libovolném okamžiku t_1 lze vystihnout energii kyvadla výrazem (1.46), platí tedy:

$$E = E_p(t_1) + E_k(t_1) = m\,g\,l\,\sin\varphi(t_1) + \frac{1}{2} m\,l^2\omega(t_1)^2. \tag{1.48}$$

V okamžiku t_2, kdy je kyvadlo v nejnižší poloze, je výška kyvadla h nad srovnávací rovinou nulová a úhlová rychlost kyvadla ω je maximální. Veškerá energie kyvadla je tedy ve formě kinetické energie. Lze proto psát:

$$E = E_p(t_2) + E_k(t_2) = 0 + \frac{1}{2} m\,l^2\omega(t_2)^2. \tag{1.49}$$

Protože uvažovaný mechanický systém je konzervativní a nedochází tedy ke ztrátám energie, popsaný cyklus pohybu kyvadla a přeměňování energie z potenciální na kinetickou se neustále opakuje.

1.3 Lagrangeovská mechanika

V průběhu let 1772 až 1788 Joseph-Louis Lagrange publikoval svoji formulaci klasické Newtonovské mechaniky. Lagrangeova formulace je ekvivalentním popisem stavu systému v porovnání s Newtonovským přístupem. Na rozdíl od Newtona, využívajícího k popisu pohybu vektory, Lagrange používá skalární veličiny - kinetickou a potenciální energii systému. Lagrangeův přístup rovněž používá efektivnějšího popisu stavu systému, totiž konfigurační prostor zobecněných souřadnic.

Lagrangeova formulace je, jak bude později ukázáno, velmi elegantním přístupem popisu dynamiky systému, viz [7] Pomocí jednoduché počáteční rovnice, obsahující pouze skalární veličiny, lze relativně snadno sestrojit pohybové rovnice popisující chování systému.

1.3.1 Zobecněné souřadnice systému

Jak bylo zmíněno, Lagrangeovský popis stavu systému nepoužívá k popisu stavu systému nutně kartézskou souřadnou soustavu. K popsání stavu systému lze zvolit libovolné parametry, které v každém okamžiku jednoznačně popisují stav sledovaného systému. Použití vhodně zvolených zobecněných souřadnic bývá výhodnější také z důvodu značné rozmanitosti úloh v mechanice.

Pokud má sledovaný systém n stupňu volnosti[4], pro jednoznačný popis stavu systému je zřejmě zapotřebí alespoň n zobecněných souřadnic. Poznamenejme, že použití více zobecněných souřadnic již dále nepřináší žádné výhody.

1.3.2 Konfigurační prostor, zobecněné rychlosti systému

Zobecněné souřadnice systému tvoří tzv. konfigurační prostor všech přípustných poloh neboli konfigurací systému. Takto popsán, konfigurační prostor neposkytuje úplnou informaci o fyzikálním stavu systému, viz [7]. V dynamické úloze je zapotřebí znát navíc rychlosti hmotných bodů, tedy zobecněné rychlosti systému. Označíme-li zobecněné souřadnice systému jako $q_0, q_1, \ldots, q_{n-1}$, pro jejich zobecněné rychlosti $\dot{q}_0, \dot{q}_1, \ldots, \dot{q}_{n-1}$ platí:

$$\dot{q}_i = \frac{\mathrm{d}q_i}{\mathrm{d}t}. \tag{1.50}$$

1.3.3 Lagrangeova funkce

Uvažujme hmotný bod M o hmotnosti m pohybující se prostorem. Kinetická energie hmotného bodu E_k má zřejmě tvar, viz 1.41. Kinetická energie je tedy funkcí rychlosti hmotného bodu:

$$E_k = \frac{1}{2} m\, \mathbf{v}^2 = \frac{1}{2} m\, \dot{\mathbf{r}}^2 = \frac{1}{2} m \left(\dot{x}^2 + \dot{y}^2 + \dot{z}^2\right) = E_k\left(\dot{x}, \dot{y}, \dot{z}\right). \tag{1.51}$$

Potenciální energie hmotného bodu E_p nabývá tvaru, viz 1.36. Závisí tedy na poloze hmotného bodu v prostoru:

$$E_p = m\, g\, h(\mathbf{r}) = E_p(x, y, z). \tag{1.52}$$

[4]Stupeň volnosti je směr posunu nebo pootočení, jimiž se hmotný bod nebo těleso může pohybovat. V prostoru obecně $n = 3N - v$, kde N je počet hmotných bodů a v je počet vazeb.

Lagrangeova funkce (nebo zkráceně Lagrangián) $\mathcal{L}$ je prostým rozdílem dvou skalárů - kinetické a potenciální energie systému. Tedy:

$$\mathcal{L} = E_k - E_p, \tag{1.53}$$

v zahraniční literatuře pak:

$$\mathcal{L} = T - V. \tag{1.54}$$

Lagrangeova funkce $\mathcal{L}$ je funkcí rychlosti a polohy hmotného bodu, tedy všech proměnných popisujících dynamický stav systému:

$$\mathcal{L} = \mathcal{L}\left(t, x, y, z, \dot{x}, \dot{y}, \dot{z}\right). \tag{1.55}$$

Pro obecný systém popsaný pomocí n zobecněných souřadnic a rychlostí:

$$\mathcal{L} = \mathcal{L}\left(t, q_0, \dot{q}_0, q_1, \dot{q}_1, \ldots, q_{n-1}, \dot{q}_{n-1}\right) = \mathcal{L}\left(t, \mathbf{q}, \dot{\mathbf{q}}\right). \tag{1.56}$$

1.3.4 Virtuální posunutí, virtuální práce

Virtuální posunutí hmotného bodu $\delta\mathbf{r}$ je libovolné myšlené posunutí, které vyhovuje okrajovým podmínkám úlohy, viz [11] a [12]. Může jím být virtuální posun $\delta\mathbf{s}$ nebo virtuální pootočení $\delta\varphi$.

Virtuální práce δWje práce skutečných sil nebo momentů na virtuálním přemístění, tedy například:

$$\begin{aligned} \delta W &= \mathbf{F}\,\delta\mathbf{s}, \\ \delta W &= \mathbf{M}\,\delta\varphi. \end{aligned} \tag{1.57}$$

Virtuální práce je skalární veličinou, neboť vzniká skalárním součinem vektorů.

Uvažujme hmotný bod, na který působí výslednice vnějších sil $\mathbf{F}$. Vektor síly F lze rozložit ve směru souřadnicových os x, y, z na dílčí vektory $\mathbf{F}_x, \mathbf{F}_y, \mathbf{F}_z$. Podobně virtuální posun $\delta\mathbf{s}$ se skládá ze složek $\delta\mathbf{s}_x, \delta\mathbf{s}_y, \delta\mathbf{s}_z$. Virtuální práce δW výslednice vnějších sil $\mathbf{F}$ na virtuálním posunutí $\delta\mathbf{s}$ je potom dána vztahem:

$$\delta W = \mathbf{F}_x\,\delta\mathbf{s}_x + \mathbf{F}_y\,\delta\mathbf{s}_y + \mathbf{F}_z\,\delta\mathbf{s}_z. \tag{1.58}$$

Pro statickou rovnovážnou soustavu sil dále platí Lagrangeův princip virtuálních prací:

"Virtuální práce δW rovnovážné soustavy sil $F_i (i = 0, 1, \ldots, n)$ působící na hmotný bod nebo tuhé těleso je při libovolném virtuálním posunutí $\delta\boldsymbol{r}$ rovna nule."

Matematicky zapsáno:

$$\delta W = \sum_{i=0}^{n} F_i\,\delta\mathbf{r}_i = 0. \tag{1.59}$$

1.3.5 D'Alembertův princip

Sestavme pohybovou rovnici pro hmotný bod M o hmotnosti m, na který působí výslednice vnějších sil $\mathbf{F}$ a výslednice reakčních sil $\mathbf{R}$:

$$\mathbf{F} + \mathbf{R} = m\,\mathbf{a} = m\,\ddot{\mathbf{r}}, \tag{1.60}$$

kde $\mathbf{r}$ je polohový vektor hmotného bodu M. Pravou stranu rovnice (1.60) lze označit jako setrvačnou sílu $\mathbf{Z}$, pro kterou platí:

$$\mathbf{Z} = -m\,\mathbf{a} = -\dot{\mathbf{p}}. \tag{1.61}$$

Po dosazení tedy lze psát:

$$\mathbf{F} + \mathbf{R} + \mathbf{Z} = 0, \tag{1.62}$$

což je matematickým zápisem D'Alembertova principu, jenž má původ v Newtonových pohybových zákonech:

"Dynamický systém je v rovnováze pod vlivem sil vnějších, reakčních a setrvačných."

Dále užijme principu virtuálních prací. Pokud budeme uvažovat ideální vazby, pro které platí:

$$\delta W_R = \mathbf{R}\,\delta\mathbf{s} = 0, \tag{1.63}$$

získáváme výraz:

$$\delta W = \mathbf{F}\,\delta\mathbf{s} + \mathbf{Z}\,\delta\mathbf{s} = 0, \tag{1.64}$$

tedy:

$$\delta W = \mathbf{F}\,\delta\mathbf{s} - (m\,\ddot{\mathbf{r}})\,\delta\mathbf{s} = \mathbf{F}\,\delta\mathbf{s} - \dot{\mathbf{p}}\delta\mathbf{s} = 0, \tag{1.65}$$

což je formulace D'Alembertova prinicpu ve formě virtuálních prací:

"Rozdíl virtuálních prací, konaných vnějšími silami působícími na hmotný bod na virtuálním posunutí $\delta\boldsymbol{s}$, a virtuálních prací, konaných silami setrvačnými na témže posunutí, je roven nule. "

1.3.6 Hamiltonův pricip nejmenší akce

Definujme nejprve akci systému $\mathcal{S}$ jako integrál Lagrangeovy funkce $\mathcal{L}$, viz (1.56), mezi dvěma časovými okamžiky t_a a t_b:

$$\mathcal{S}(t_a, t_b) = \int_{t_a}^{t_b} \mathcal{L}\,(t, \mathbf{q}, \dot{\mathbf{q}})\;dt \tag{1.66}$$

Princip nejmenší akce se zakládá na tvrzení, že ze všech přípustných trajektorií systému je uskutečněna trajektorie nejvýhodnější, viz [7]. Například z hlediska energetické náročnosti, nutného času[5] a podobně. U mechanického systému předpokládáme, že ze všech možných trajektoriií proběhne ta, pro kterou integrál (1.66) nabývá minimální hodnoty.

Uvedený integrál se nazývá funkcionál[6]. Úkolem je tedy najít minimální hodnotu funkcionálu, totiž extrém ve stacionárním bodě.

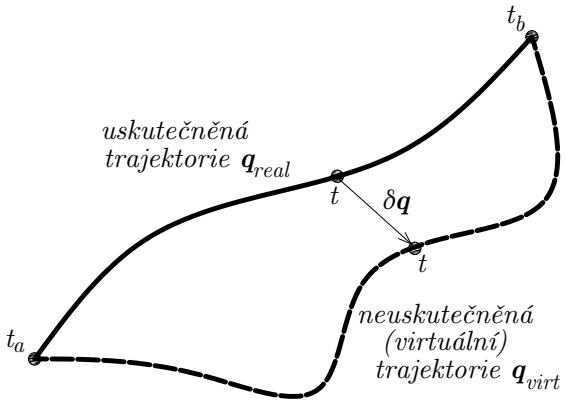
[5] Příkladem je tzv. Fermatův princip v optice: Světlo se v prostoru šíří z jednoho bodu do druhého po takové dráze, aby doba potřebná k překonání této dráhy byla minimální.

[6] Funkcionál je zobrazení, které přiřazuje funkci číslo - energii, čas a podobně.

1.3.7 Lagrangeovy rovnice

Hamiltonův princip nejmenší akce dále použijeme k odvození Lagrangeových rovnic. Jak bylo uvedeno, vývoj stavu systému bude probíhat po trajektorii s minimální akcí. Zaveďme virtuální posunutí $\delta\mathbf{q}(t)$ jako infinitezimální rozdíl myšlené a uskutečněné trajektorie systému v libovolném čase t, viz obrázek 1.8:

$$\delta\mathbf{q}(t) = \mathbf{q}_{virt}(t) - \mathbf{q}_{real}(t). \tag{1.67}$$



Obrázek 1.8: Trajektorie vývoje dynamického systému.

Dále uveďme, že virtuální i uskutečněná trajektorie začíná a končí v totožném bodě konfiguračního prostoru:

$$\delta\mathbf{q}(t_a) = \delta\mathbf{q}(t_b) = 0, \tag{1.68}$$

a nakonec záměnnost operací derivace podle času a variace δ:

$$\frac{\mathrm{d}}{\mathrm{d}t}\delta\mathbf{q} = \delta\dot{\mathbf{q}}. \tag{1.69}$$

Dále budeme hledat minimum integrálu akce $\mathcal{S}$. Stanovme podmínky extremálnosti:

$$\begin{aligned}
&\delta \int_{t_a}^{t_b} \mathcal{L}\left(t, \mathbf{q}, \dot{\mathbf{q}}\right)\, dt = 0, \\
&\int_{t_a}^{t_b} \delta\mathcal{L}\left(t, \mathbf{q}, \dot{\mathbf{q}}\right)\, dt = 0, \\
&\int_{t_a}^{t_b} \left(\frac{\partial\mathcal{L}}{\partial q_k}\delta q_k + \frac{\partial\mathcal{L}}{\partial\dot{q}_k}\delta\dot{q}_k\right) dt = 0.
\end{aligned} \tag{1.70}$$

Nyní proveďme integraci per partes druhého členu výsledku (1.70):

$$\int_{t_a}^{t_b} \left\{\frac{\partial\mathcal{L}}{\partial q_k}\delta q_k - \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial\mathcal{L}}{\partial\dot{q}_k}\right)\delta q_k\right\} dt + \left[\frac{\partial\mathcal{L}}{\partial\dot{q}_k}\delta q_k\right]_{t_a}^{t_b} = 0. \tag{1.71}$$

S přihlédnutím k tvrzení (1.68) lze poslední člen výrazu (1.71) označit za nulový. Můžeme potom psát:

$$\int_{t_a}^{t_b} \left\{ \frac{\partial \mathcal{L}}{\partial q_k} - \frac{\mathrm{d}}{\mathrm{d}t} \left(\frac{\partial \mathcal{L}}{\partial \dot{q}_k} \right) \right\} \delta q_k dt = 0. \tag{1.72}$$

Rovnost (1.72) musí platit pro každé časy t_a a t_b a pro každé virtuální posunutí δq_k. Zároveň virtuální posunutí δq_k jsou na sobě nezávislá, protože počet zobecněných souřadnic je roven počtu stupňů volnosti. Pro každé k potom musí platit:

$$\frac{\partial \mathcal{L}}{\partial q_k} - \frac{\mathrm{d}}{\mathrm{d}t} \left(\frac{\partial \mathcal{L}}{\partial \dot{q}_k} \right) = 0, \quad k = 0, 1, \ldots, n-1\,, \tag{1.73}$$

kde n je počet zobecněných souřadnic systému.

Rovnice (1.73) se nazývají Lagrangeovy rovnice, viz [20]. Představují nutné podmínky extremálnosti (minimality) funkcionálu akce $\mathcal{S}$. Matematicky se jedná o obyčejné diferenciální rovnice druhého řádu pro uskutečněnou trajektorii q_k, viz [14] a [15].

1.4 Pohybové rovnice a jejich řešení

1.4.1 Pohybové rovnice

Pohybová rovnice systému je matematický předpis popisující možné trajektorie vývoje systému v závislosti na čase, viz [14] a [2]. Pro řešení pohybových rovnic je potřeba znát počáteční podmínky, zde počáteční hodnoty stavových proměnných systému.

Pro řešení disipativních systémů[7] běžně postačují tzv. explicitní numerické metody, viz [16]. Konzervativní[8] systémy zpravidla vyžadují robustnější numerické metody kvůli náročnosti na numerickou stabilitu řešení.

1.4.2 Eulerova metoda

Eulerova metoda je nejjednodušší explicitní metodou pro řešení obyčejných diferenciálních rovnic prvního řádu. Eulerova metoda je jednokroková[9] a jednobodová[10] numerická metoda.

Uvažujme diferenciální rovnici ve tvaru:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = F(t, x(t)). \tag{1.74}$$

Předpis Eulerovy metody je potom:

$$x(t+h) = x(t) + h\,F(t, x(t)), \tag{1.75}$$

kde h je krok numerické metody a $F(t, x(t))$ je směrnice aproximační přímky v bodě $x(t)$. Směrnici $F(t, x(t))$ lze obdržet jako numerickou první derivaci. Přibližně potom platí:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}. \tag{1.76}$$

[7] U disipativního mechanického systému se celková energie systému nezachovává. Disipace energie může být způsobena například tlumením systému.

[8] V konzervativních mechanických systémech zůstává celková energie systému konstantní.

[9] Pro řešení k-krokové numerické metody je zapotřebí znát k předchozích stavů - kroků.

[10] Při řešení pomocí l-bodové numerické metody je potřeba spočítat l hodnot funkce $F(t, x(t))$.

Po úpravě lze tedy psát:

$$f(x+h) \approx f(x) + h\, f'(x). \tag{1.77}$$

Chyba Eulerovy metody je řádu $\mathcal{O}(h^2)$. Pro řešení dynamického systému s n stavovými proměnnými nabývá předpis Eulerovy metody tvaru:

$$x_i(t+h) = x_i(t) + h\, F_i(t, x_1(t), x_2(t), \ldots, x_n(t)). \tag{1.78}$$

1.4.3 Semi-implicitní Eulerova metoda

Mnohem stabilnějšího řešení při použití Eulerovy metody lze dosáhnout jednodouchou úpravou vycházející z vlastností dynamických systémů. Dynamické systémy popisující chování mechanických konstrukcí mají původ v diferenciálních rovnicích druhého řádu a pomocí jednoduchých substitucí se upravují na diferenciální rovnice prvního řádu.

Například diferenciální rovnici ve tvaru:

$$m\frac{\mathrm{d}^2x}{\mathrm{d}t^2} + c\frac{\mathrm{d}x}{\mathrm{d}t} + k\,x = 0, \tag{1.79}$$

lze substitucí

$$v = \frac{\mathrm{d}x}{\mathrm{d}t}, \tag{1.80}$$

upravit na soustavu

$$\begin{aligned} \frac{\mathrm{d}v}{t} &= -\frac{c}{m}v - \frac{k}{m}x, \\ \frac{\mathrm{d}x}{\mathrm{d}t} &= v. \end{aligned} \tag{1.81}$$

Při řešení Eulerovou metodou, viz (1.75), dostáváme diferenční vztahy:

$$\begin{aligned} v(t+h) &= v(t) - h\,(\frac{c}{m}v(t) + \frac{k}{m}x(t)), \\ x(t+h) &= x(t) + h\,v(t). \end{aligned} \tag{1.82}$$

Pokud nejdříve vyřešíme hodnotu stavové proměnné v v čase $t+h$, lze předpis (1.82) upravit na:

$$\begin{aligned} v(t+h) &= v(t) - h\,(\frac{c}{m}v(t) + \frac{k}{m}x(t)), \\ x(t+h) &= x(t) + h\,v(t+h). \end{aligned} \tag{1.83}$$

Popsaná úprava Eulerovy metody tedy spočívá v použití nově vypočtené rychlosti v pro výpočet nového posunutí x. Stejné stability výpočtu dosáhneme opačnou úpravou, totiž užitím nově vypočteného posunutí pro výpočet nové rychlosti.

1.4.4 Runge-Kuttova metoda

Runge-Kuttova metoda je jednokroková a čtyřbodová numerická metoda. Pro výpočet je tedy zapotřebí znát jeden předchozí stav a hodnoty funkce $F(t, x(t))$ ve čtyřech bodech. Průběh funkce je aproximován polynomem čtvrtého stupně. Předpis Runge-Kuttovy metody je následující:

$$\begin{aligned}
k_1 &= F\left(t,\ x(t)\right), \\
k_2 &= F\left(t + \frac{h}{2}, x(t) + k_1\,\frac{h}{2}\right), \\
k_3 &= F\left(t + \frac{h}{2}, x(t) + k_2\,\frac{h}{2}\right), \\
k_4 &= F\left(t + h, x(t) + \ k_3\,h\right), \\
x(t+h) &= x(t) + \frac{h}{6}\left(k_1 + 2k_2 + \ 2k_3 + k_4\right).
\end{aligned} \tag{1.84}$$

Chyba Runge-Kuttovy metody je řádu $\mathcal{O}(h^5)$. Pro řešení soustav rovnic dynamického systému s n stavovými proměnnými lze předpis zobecnit:

$$\begin{aligned}
k_{1i} &= F_i\left(t, x_1(t),\ x_2(t), \ldots, x_n(t)\right) \\
k_{2i} &= F_i\left(t + \frac{h}{2}, x_1(t) + k_{11}\,\frac{h}{2}, x_2(t) + k_{12}\,\frac{h}{2}, \ldots, x_n(t) + k_{1n}\,\frac{h}{2}\right) \\
k_{2i} &= F_i\left(t + \frac{h}{2}, x_1(t) + k_{21}\,\frac{h}{2}, x_2(t) + k_{22}\,\frac{h}{2}, \ldots, x_n(t) + k_{2n}\,\frac{h}{2}\right) \\
k_{4i} &= F_i\left(t + h, x_1(t) + k_{31}\,h, x_2(t) + k_{32}\,h, \ldots,\ x_n(t) + k_{3n}\,h\right) \\
x_i(t+h) &= x(t) + \frac{h}{6}\left(k_{1i} + \ 2k_{2i} + 2k_{3i} + k_{4i}\right), \quad i = 1, 2, \ldots, n.
\end{aligned} \tag{1.85}$$

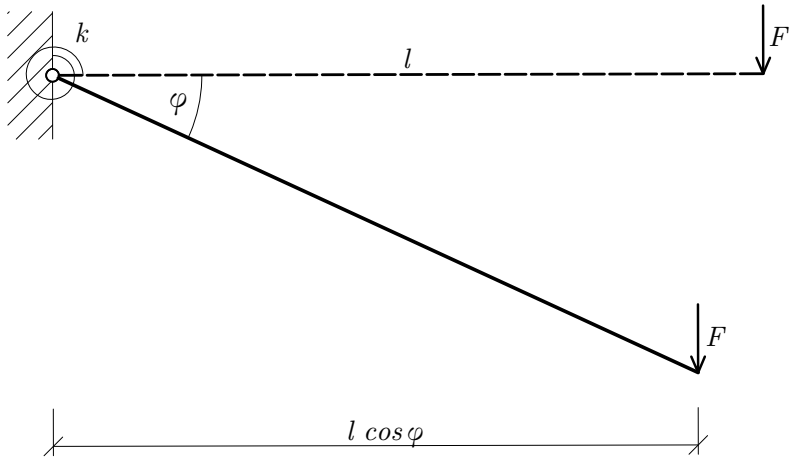
1.5 Zdroje nelinearit

Fyzikální nelinearitou rozumíme existenci nelineární závislosti mezi charakteristikami vnějšího zatížení, napjatosti a deformace konstrukce. U reálných konstrukcí se vždy vyskytuje nelineární chování, nicméně u mnohých úloh je za určitých podmínek linearizace řešení přípustná a vede ke snazšímu vyřešení problému se zanedbatelnou odchylkou oproti skutečnému řešení.

1.5.1 Geometrická nelinearita

Geometrická nelinearita řešení je způsobena závislostí tvaru (geometrie) konstrukce na zatížení nebo okrajových podmínkách. Vznik geometrické nelinearity ukažme na následujícím příkladu.

Uvažujme prut délky l o jednom stupni volnosti, kterému v pootočení brání pružina tuhosti k, viz obrázek 1.9. Na volném konci prutu působí osamělá síla F.



Obrázek 1.9: Konzola zatížená osamělou silou.

Momentová podmínka k bodu vetknutí má tvar:

$$F\,l\,\cos\varphi = k\,\varphi. \tag{1.86}$$

Při malém pootočení nosníku lze uvažovat:

$$\cos\varphi \approx 1, \tag{1.87}$$

což vede k lineární závislosti mezi působící silou F a pootočením φ. Pro malé deformace lze tento předpoklad použít, aniž bychom se dopustili závažné chyby. S rostoucí deformací se ale lineární řešení od přesného významně odchyluje a není již přípustné.

Pro přesnější aproximaci lze nelineární funkci rozložit do řady, z níž se dále použijí první dva nebo více členů. Pro zmíněný případ platí:

$$\cos\varphi = 1 - \frac{\varphi^2}{2!} + \frac{\varphi^4}{4!} - \frac{\varphi^6}{6!} + \dots \tag{1.88}$$

Po dosazení do momentové podmínky obdržíme:

$$F\,l\left(1 - \frac{\varphi^2}{2!}\right) = k\,\varphi. \tag{1.89}$$

Tato aproximace se jeví jako vyhovující pro větší rozsah úhlu φ. Při dalším nárustu nelinearity se pochopitelně i toto řešení začíná lišit od přesných hodnot.

U úloh, kde vznikají natolik významná přetvoření, že analytické řešení je složité nebo nemožné, nezbývá než řešit úlohu numericky. Tedy rozdělit řešení na podkroky a v každém uvažovat odpovídající geometrii konstrukce.

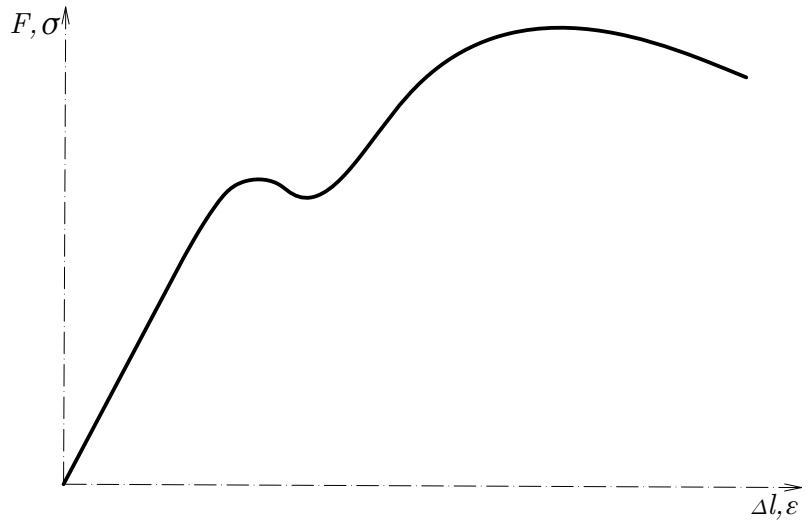
1.5.2 Materiálová nelinearita

Materiálová nelinearita vzniká v případě, pokud materiál modelu není lineárně pružný s platností Hookeova zákona:

$$\sigma = E\,\varepsilon. \tag{1.90}$$

Závislost mezi napjatostí a deformací tedy není lineární, viz obrázek 1.10, a vlastnosti materiálu jsou funkcí jeho deformace:

$$\sigma = E(\varepsilon)\,\varepsilon. \tag{1.91}$$



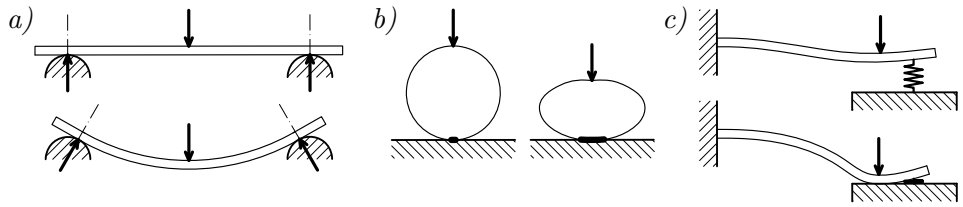
Obrázek 1.10: Nelineární pracovní diagram materiálu.

Podobně jako v předchozím případě je třeba úlohu řešit numericky a v každém kroku použít materiálové charakteristiky odpovídající deformaci modelu.

1.5.3 Nelinearita způsobená okrajovými podmínkami

Při řešení mechanických úloh jsou skutečná uložení a zatížení často nahrazována idealizovanými vazbami a silami. Tato zjednodušení ovšem nesmí být na úkor výstižnosti modelu. Ve skutečnosti se uložení a zatížení děje jako vzájemný kontakt mezi tělesy. Tato skutečnost má za následek proměnlivé charakteristiky okrajových podmínek.

Jako příklad uveďme změnu polohy teoretických podpor a směru působení jejich reakcí, viz obrázek 1.11a, proměnlivost styčných ploch na kontaktu těles (1.11b) nebo proměnnou tuhost podpory (1.11c). Běžné je také například nadzdvižení konstrukce z podpory nebo naopak kontakt s dalším objektem v důsledku její deformace.



Obrázek 1.11: Příklady některých nelineárních vazeb.

1.6 Konzervativní a nekonzervativní síly

1.6.1 Konzervativní síly

Konzervativní silou je každá síla, která je závislá pouze na své poloze. Je tedy možné v každém okamžiku určit její potenciál. Jako příklad konzervativní síly lze uvést gravitační sílu, viz sekce 1.2. Pro konzervativní síly platí:

- Při pohybu tělesa po uzavřené křivce, viz obrázek 1.12a, je práce konaná konzervativní silou rovna nule:

$$W = \oint_C \mathbf{F}\, d\mathbf{r} = 0. \tag{1.92}$$

- Při pohybu tělesa z jednoho bodu do druhého není celková práce vykonaná konzervativní silou závislá na trajektorii, viz obrázek 1.12b.

- Konzervativní síla je opačná ke gradientu svého potenciálu:

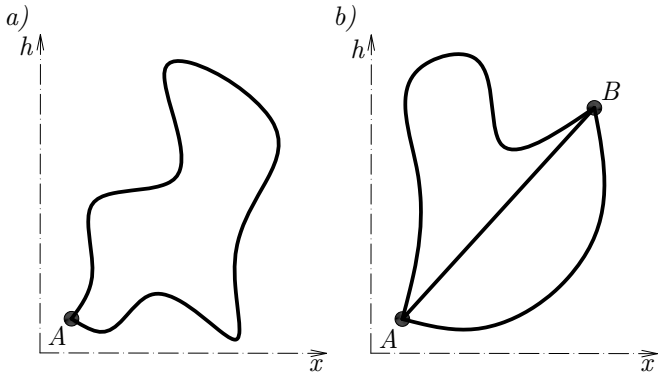
$$\mathbf{F} = -\nabla\, \Phi. \tag{1.93}$$

1.6.2 Nekonzervativní síly

Pro nekonzervativní síly neplatí uvedené vlastnosti sil konzervativních. Pro nekonzervativní síly tedy nelze určit potenciál, neboť různé trajektorie vedou k jiné vykonané práci. Stejně tak jimi vykonaná práce na uzavřené křivce není rovna nule. Zde jako příklad lze uvést třecí nebo tlumící síly.

Nekonzervativní síly nejsou závislé pouze na poloze svého působiště, ale také na okamžité konfiguraci systému. Nekonzervativní síly, které mění své působení v závislosti na geometrii konstrukce, se nazývají *sledující síly*, viz [10].

Reálným příkladem sledujících sil je tryskání média z konzolově uložené trubky nebo právě předmět této práce - tah reaktivního motoru působící tečně vůči konci rakety, viz obrázek 1.13.

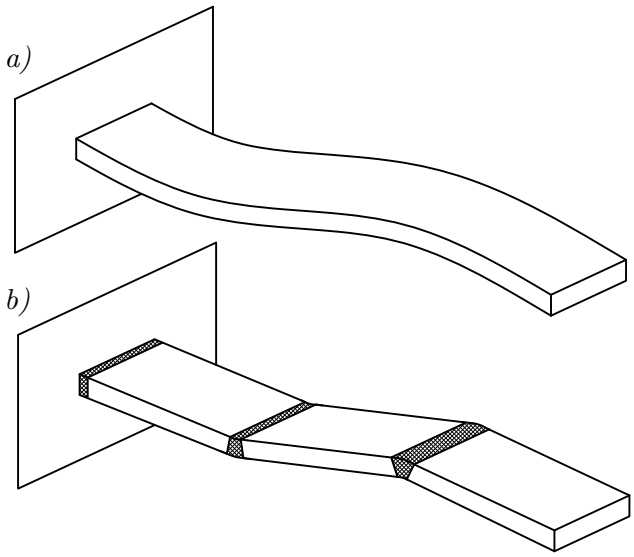


Obrázek 1.12: Znázornění konzervativních vlastností gravitační síly.

Obrázek 1.13: Pružná štíhlá raketa zatížená tahem motoru.

1.7 Metoda tuhých dílců

Metoda tuhých dílců, nazývaná také metoda simplexních prvků, viz [8] a [2], užívá speciálního způsobu diskretizace kontinua[11]. Kontinuum je rozděleno na tuhé dílce vzájemně spojené prvky schopné deformace, tzv. kumulátory přetvoření, viz obrázek 1.14b. Narozdíl od reálné konstrukce a metody konečných prvků nezůstává model hladce spojitý, viz obrázek 1.14a. Výhodou metody tuhých dílců je bezproblémové vystihnutí geometrické nelinearity. V případě dynamické formulace navíc obdržíme přímo soustavu pohybových rovnic v kanonickém tvaru.



Obrázek 1.14: Idealizace konstrukce metodou tuhých dílců.

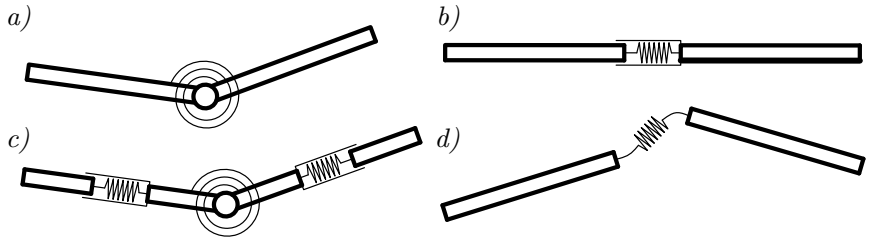
[11] Jako fyzikální diskretizace je myšleno vytvoření modelu konstrukce, který je svou podstatou diskrétní, lze jej ale chápat jako reálnou konstrukci, viz [4].

Každý tuhý dílec má vlastní poměrnou hmotnost a vlastní těžiště, je tedy nositelem kinetické energie modelu. Deformace - potenciální energie modelu - se akumuluje na kontaktech tuhých dílců v kumulátorech přetvoření. Tyto jsou běžně uvažovány jako dokonale pružné.

Pružné spojení tuhých dílců by mělo být voleno s ohledem na povahu úlohy tak, aby dostatečně vystihovalo podstatu modelu, ale nenavyšovalo zbytečně složitost odvození a nebo výpočetní náročnost řešení.

Vybrané možné varianty jsou ukázány na obrázku 1.15:

- spojení tuhých dílců rotační pružinou (1.15a),
- spojení translační pružinou (1.15b),
- spojení kombinací rotačních a translačních pružin (1.15c),
- spojení obecnou pružinou umožňující navíc zahrnutí vlivu smyku (1.15d).



Obrázek 1.15: Varianty kontaktních kumulátorů přetvoření.

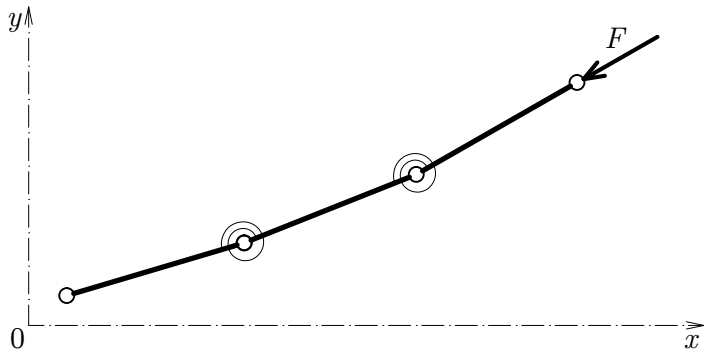
Kapitola 2

Odvození numerického modelu

2.1 Úvod

Záměrem práce je vytvoření nelineárního dynamického modelu volného prutu schopného velkých deformací. Pro tento úkol použijeme zvláštní případ metody tuhých dílců, viz [8].

Výhodou tohoto modelu je, jak později uvidíme, jeho jednoduchost a nenáročnost na výpočetní dobu. Ta je u numerických modelů přímo úměrná nejvyšší frekvenci, kterou model kmitá. U štíhlého ocelového prutu jsou řádově nejvyšší frekvence dosahovány při normálovém kmitání (kmitání pružných kumulátorů normálového přetvoření - translačních pružin). Na chování takové konstrukce má však normálové kmitání na rozdíl od kmitání příčného zanedbatelný vliv. Volíme tedy model normálově dokonale tuhý, čímž se vyhneme vzniku vyšších frekvencí. Jedinými kumulátory přetvoření v modelu tak zůstávají rotační pružiny spojující tuhé dílce. Volný prut uvažujeme konstantního průřezu s hmotností M,



Obrázek 2.1: Model volného prutu zatíženého sledující silou.

délkou L a ohybovou tuhostí EI. Výpočtový model rozděluje prut na n tuhých dílců vzájemně spojených klouby. V místech kloubů nahrazují ohybovou tuhost lineární rotační pružiny, neboť díky dostatečné štíhlosti prutu uvažujeme materiál jako lineárně pružný. Lze ukázat, že pro prvky modelu platí:

$$m = \frac{M}{n}, \quad l = \frac{L}{n}, \quad k = \frac{EI}{l}, \tag{2.1}$$

kde m je hmotnost tuhého dílce, l je délka tuhého dílce a k je tuhost rotační pružiny.

2.1.1 Princip odvození

Při odvozování se budeme řídit zásadami klasické mechaniky, viz [19]. Nejdříve sestavíme Lagrangeovu funkci L, viz sekce 1.3.7.

$$L = E_k - E_p, \tag{2.2}$$

kde E_k je kinetická energie modelu a E_p je potenciální energie modelu. Pohybové rovnice modelu potom obdržíme užitím Eulerovy-Lagrangeovy rovnice, viz (1.73):

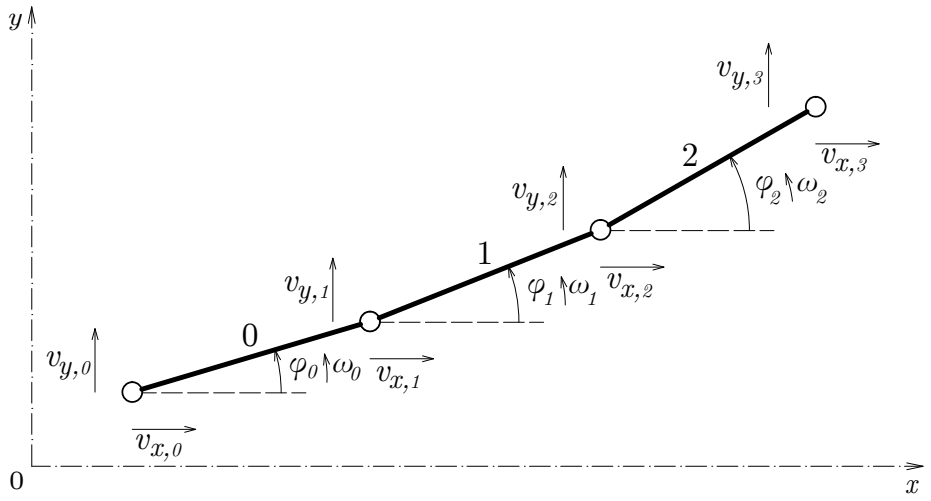
$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{q}_i}\right) = \frac{\partial L}{\partial q_i}, \tag{2.3}$$

kde t je čas, q_i je zobecněná souřadnice systému a $\dot{q}_i$ je zobecněná rychlost systému.

2.2 Energie

2.2.1 Kinetická energie modelu

Uvažujme rovinný model volného prutu složený ze tří tuhých dílců, viz obrázek 2.2. Dílce jsou stejně dlouhé a rotační pružiny mají stejnou tuhost. Vztahy, které zde získáme, nám dále poslouží pro zobecnění celého problému.

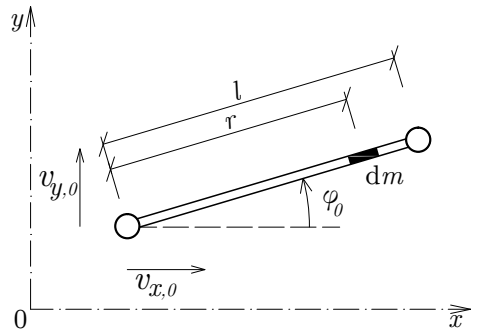


Obrázek 2.2: Model volného prutu.

Jako první formulujme výraz pro kinetickou energii prvního dílce E_{k1}. Pro vytnutý diferenciální element o délce dr platí:

$$dE_{k0} = \frac{1}{2} dm\, v_1^2 = \frac{1}{2} \rho\, dr\, v_1^2, \tag{2.4}$$

kde dm je hmotnost diferenciálního elementu, ρ je hustota elementu a v_1 je absolutní rychlost elementu.



Obrázek 2.3: První dílec modelu.

Absolutní rychlost elementu získáme složením dvou nezávislých pohybů - translačního pohybu prvního bodu o rychlosti v_0 a rotačního pohybu prvního dílce o úhlové rychlosti ω_0. Dílčí složky translační rychlosti v_{x1} a v_{y1} můžeme tedy zapsat ve tvaru:

$$\begin{aligned} v_{x1} &= v_{x0} - \omega_0\, r \sin(\varphi_0), \\ v_{y1} &= v_{y0} + \omega_0\, r \cos(\varphi_0). \end{aligned} \tag{2.5}$$

Pro složky translační rychlosti v_{x0} a v_{y0} platí:

$$v_{x0} = \frac{d}{dt} x_0, \;\; v_{y0} = \frac{d}{dt} y_0, \tag{2.6}$$

a pro úhlové rychlosti ω_i obecně:

$$\omega_i = \frac{d}{dt} \varphi_i, \;\; i = 0, 1 \ldots n-1. \tag{2.7}$$

S užitím výrazu:

$$v_1 = \sqrt{v_{x1}^2 + v_{y1}^2}, \tag{2.8}$$

můžeme po úpravě a dosazení do rovnice (2.4) psát:

$$dE_{k0} = \frac{1}{2} \rho\, dr\, v_1^2 = \frac{1}{2} \rho\, dr \, \left[(v_{x0} - \omega_0\, r \sin(\varphi_0))^2 + (v_{y0} + \omega_0\, r \cos(\varphi_0))^2 \right]. \tag{2.9}$$

Celkovou kinetickou energii prvního dílce E_{k1} získáme integrací po celé délce:

$$\begin{aligned} E_{k0} =& \frac{1}{2} \rho \int_0^l \left[(v_{x0} - \omega_0\, r \sin(\varphi_0))^2 + (v_{y0} + \omega_0\, r \cos(\varphi_0))^2 \right] dr \\ =& \frac{1}{2} m \left[v_{x0}^2 + v_{y0}^2 + \frac{1}{3} \omega_0^2\, l^2 - v_{x0}\, \omega_0\, l\, \sin(\varphi_0) + v_{y0}\, \omega_0\, l\, \cos(\varphi_0) \right]. \end{aligned} \tag{2.10}$$

V případě druhého dílce postupujeme obdobně. Celkovou rychlost elementu na druhém dílci obdržíme složením translační rychlosti počátečního bodu v_0, obvodové rychlosti koncového bodu prvního dílce a rotačního pohybu diferenciálního elementu na druhém dílci o úhlové rychlosti ω_1. Pro složky v_{x2} a v_{y2} získáváme vztahy:

$$\begin{aligned} v_{x2} &= v_{x0} - \omega_0\, l \sin(\varphi_0) - \omega_1\, r \sin(\varphi_1), \\ v_{y2} &= v_{y0} + \omega_0\, l \cos(\varphi_0) + \omega_1\, r \cos(\varphi_1). \end{aligned} \tag{2.11}$$

Výraz pro kinetickou energii diferenciálního elementu na druhém dílci má tedy následující podobu:

$$\begin{aligned} dE_{k1} = \frac{1}{2}\rho\, dr\, v_2^2 = \frac{1}{2}\rho\, dr\, \big[&(v_{x0} - \omega_0\, l \sin(\varphi_0) - \omega_1\, r \sin(\varphi_1))^2 + \\ &+ (v_{y0} + \omega_0\, l \cos(\varphi_0) + \omega_1\, r \cos(\varphi_1))^2\big], \end{aligned} \tag{2.12}$$

Pro získání kinetické energie druhého dílce E_{k1} opět zintegrujme výraz (2.12) přes celou délku dílce:

$$\begin{aligned} E_{k1} = \frac{1}{2}\rho \int_0^l v_2^2 dr = \frac{1}{2} m \Big[& v_{x0}^2 + v_{y0}^2 + \omega_0^2 l^2 + \frac{1}{3}\,\omega_1 2 l^2 - 2\, v_{x0}\, \omega_0\, l \sin(\varphi_0) + \\ & + 2\, v_{y0}\, \omega_0\, l \cos(\varphi_0) - v_{x0}\, \omega_1\, l\, \sin(\varphi_1) + v_{y0}\, \omega_1\, l\, \cos(\varphi_1) + \\ & + \omega_0\, \omega_1\, l^2 \cos(\varphi_1 - \varphi_0) \Big]. \end{aligned} \tag{2.13}$$

Oproti výrazu (2.10) budou nadále přibývat součiny úhlových rychlostí, které v dalším odvození pohybových rovnic způsobí nárůst složitosti výpočtu.

S užitím stejných předpokladů jako v předchozím případě můžeme dále odvodit výraz pro kinetickou energii třetího dílce E_{k2}. Dílčí složky celkové rychlosti diferenciálního elementu na třetím dílci vzniknou obdobně:

$$\begin{aligned} v_{x3} &= v_{x0} - \omega_0\, l \sin(\varphi_0) - \omega_1\, l \sin(\varphi_1) - \omega_2\, r \sin(\varphi_2), \\ v_{y3} &= v_{y0} + \omega_0\, l \cos(\varphi_0) + \omega_1\, l \cos(\varphi_1) + \omega_2\, r \cos(\varphi_2). \end{aligned} \tag{2.14}$$

Po integraci získáváme kinetickou energii třetího dílce E_{k2} ve tvaru:

$$\begin{aligned} E_{k2} = \frac{1}{2}\rho \int_0^l v_3^2 dr = \frac{1}{2} m \Big[& v_{x0}^2 + v_{y0}^2 + \omega_0^2 l^2 + \omega_1^2 l^2 + \frac{1}{3}\,\omega_2^2 l^2 - 2\, v_{x0}\, \omega_0\, l \sin(\varphi_0) + \\ & + 2\, v_{y0}\, \omega_0\, l \cos(\varphi_0) - 2\, v_{x0}\, \omega_1\, l \sin(\varphi_1) + 2\, v_{y0}\, \omega_1\, l \cos(\varphi_1) - \\ & - v_{x0}\, \omega_2\, l\, \sin(\varphi_2) + v_{y0}\, \omega_2\, l\, \cos(\varphi_2) + 2\, \omega_0\, \omega_1\, l^2 \cos(\varphi_1 - \varphi_0) + \\ & + \omega_0\, \omega_2\, l^2 \cos(\varphi_2 - \varphi_0) + \omega_1\, \omega_2\, l^2 \cos(\varphi_2 - \varphi_1) \Big]. \end{aligned} \tag{2.15}$$

Nyní formulujme obecný tvar kinetické energie s-tého tuhého dílce $E_{k,s}$. Všimneme-li si rozvoje výrazů (2.10), (2.13) a (2.15), můžeme psát:

$$\begin{aligned} E_{k,s} = \frac{1}{2}m \Bigg\{ & v_{x0}^2 + v_{y0}^2 + l^2 \sum_{i=0}^{s-1} \omega_i^2 + \frac{1}{3}\omega_s^2 l^2 - 2\, v_{x0}\, l \sum_{i=0}^{s-1} \omega_i \, \sin(\varphi_i) + \\ & + 2\, v_{y0}\, l \sum_{i=0}^{s-1} \omega_i \, \cos(\varphi_i) + \omega_s\, l \left[-v_{x0} \sin(\varphi_s) + v_{y0} \cos(\varphi_s) \right] + \\ & + 2\, l^2 \sum_{k=0}^{s-2} \left[\sum_{i=1}^{k-1} \omega_k\, \omega_i \, \cos(\varphi_i - \varphi_k) + \sum_{i=k+1}^{s-1} \omega_k\, \omega_i \, \cos(\varphi_i - \varphi_k) \right] + \\ & + l^2 \sum_{k=0}^{s-1} \omega_k\, \omega_s \, \cos(\varphi_s - \varphi_k) \Bigg\}. \end{aligned} \tag{2.16}$$

Výraz pro celkovou kinetickou energii modelu E_k má tedy podobu:

$$\begin{aligned} E_k = \frac{1}{2}m \Bigg\{ & n \left(v_{x0}^2 + v_{y0}^2 \right) + l^2 \sum_{i=0}^{n-1} \left[\omega_i^2 \left(\frac{1}{3} + (n - i - 1) \right) \right] - \\ & - l\, v_{x0} \sum_{i=0}^{n-1} \left[\omega_i \sin(\varphi_i) \left(1 + 2(n - i - 1) \right) \right] + \\ & + l\, v_{y0} \sum_{i=0}^{n-1} \left[\omega_i \cos(\varphi_i) \left(1 + 2(n - i - 1) \right) \right] + \\ & + \sum_{j=0}^{n-1} \Bigg[2\, l^2 \sum_{k=0}^{j-2} \left(\sum_{i=1}^{k-1} \omega_k\, \omega_i \, \cos(\varphi_i - \varphi_k) + \sum_{i=k+1}^{j-1} \omega_k\, \omega_i \, \cos(\varphi_i - \varphi_k) \right) + \\ & + l^2 \sum_{k=0}^{n-2} \omega_k\, \omega_{n-1} \, \cos(\varphi_{n-1} - \varphi_k) \Bigg] \Bigg\}. \end{aligned} \tag{2.17}$$

2.2.2 Potenciální energie modelu

Získání výrazu pro potenciální energii E_p bude snazší, neboť v uvažovaném modelu se potenciální energie akumuluje pouze v rotačních pružinách mezi tuhými dílci. Pružině spojující s-tý a $s{+}1$ dílec přidělíme index $s, s{+}1$. Napjatost pružiny můžeme zapsat jako:

$$M_{s,s+1} = k\, \Delta\varphi = k \left(\varphi_{s+1} - \varphi_s \right), \tag{2.18}$$

pro její potenciální energii $E_{p,s}$ tedy platí:

$$E_{p,s} = \frac{1}{2} M_{s,s+1}\, \Delta\varphi = \frac{1}{2} k \, \left(\varphi_{s+1} - \varphi_s \right)^2. \tag{2.19}$$

Celkovou potenciální energii E_p získáme sumací všech dílčích výrazů:

$$E_p = \sum E_{p,i} = \frac{1}{2} k \sum_{s=0}^{n-2} \left(\varphi_{s+1} - \varphi_s \right)^2. \tag{2.20}$$

2.3 Pohybové rovnice

Jak již bylo zmíněno, pro odvození pohybových rovnic použijeme základní vztahy (2.2) a (2.3). Zároveň budeme mít na paměti substituce (2.6) a (2.7). Odvození tedy provedeme podle rovnic:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial v_{x0}}\right) = \frac{\partial L}{\partial x_0}, \tag{2.21}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial v_{y0}}\right) = \frac{\partial L}{\partial y_0}, \tag{2.22}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \omega_s}\right) = \frac{\partial L}{\partial \varphi_s},\ s = 0, 1, 2, \ldots, n-1. \tag{2.23}$$

Nejprve provedeme derivace kinetické energie E_k podle stavových proměnných v_{x0}, v_{y0} a ω_s. Podle výrazu (2.17) můžeme psát:

$$\frac{\partial E_k}{\partial v_{x0}} = \frac{1}{2}m\left\{2\,n\,v_{x0} - l\sum_{i=0}^{n-1}\left[\omega_i \sin(\varphi_i)\,(1+2(n-i-1))\right]\right\}, \tag{2.24}$$

$$\frac{\partial E_k}{\partial v_{y0}} = \frac{1}{2}m\left\{2\,n\,v_{y0} + l\sum_{i=0}^{n-1}\left[\omega_i \cos(\varphi_i)\,(1+2(n-i-1))\right]\right\}, \tag{2.25}$$

$$\begin{aligned}\frac{\partial E_k}{\partial \omega_s} = \frac{1}{2}m\Bigg\{&2\,l^2\,\omega_s\left(\frac{1}{3}+(n-s-1)\right) - l\,v_{x0}\,\sin(\varphi_s)\,(1+2(n-s-1)) + \\ &+ l\,v_{y0}\,\cos(\varphi_s)\,(1+2(n-s-1)) + 2\,l^2\sum_{i=0}^{s-1}\omega_i\,\cos(\varphi_s-\varphi_i) + \\ &+ l^2\omega_{n-1}\,\cos(\varphi_s-\varphi_{n-1})\ \Bigg\},\end{aligned} \tag{2.26}$$

Výsledky, které jsme obdrželi, dále zderivujeme podle času:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial E_k}{\partial v_{x0}}\right) = \frac{1}{2}m\Bigg\{&2\,n\,\dot{v}_{x0} - l\sum_{i=0}^{n-1}\left[\dot{\omega}_i \sin(\varphi_i)\,(1+2(n-i-1))\right] - \\ &- l\sum_{i=0}^{n-1}\left[\omega_i^2\cos(\varphi_i)\,(1+2(n-i-1))\right]\Bigg\},\end{aligned} \tag{2.27}$$

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial E_k}{\partial v_{y0}}\right) = \frac{1}{2}m\Bigg\{&2\,n\,\dot{v}_{y0} + l\sum_{i=0}^{n-1}\left[\dot{\omega}_i \cos(\varphi_i)\,(1+2(n-i-1))\right] - \\ &- l\sum_{i=0}^{n-1}\left[\omega_i^2\sin(\varphi_i)\,(1+2(n-i-1))\right]\Bigg\},\end{aligned} \tag{2.28}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial E_k}{\partial \omega_s}\right) = \frac{1}{2}m\left\{2\,l^2\,\dot{\omega}_s\left(\frac{1}{3}+(n-s-1)\right) - l\,\dot{v}_{x0}\,\sin(\varphi_s)\,(1+2(n-s-1)) - \right.$$
$$- l\,v_{x0}\,\omega_s\cos(\varphi_s)\,(1+2(n-s-1)) + l\,\dot{v}_{y0}\,\cos(\varphi_s)\,(1+2(n-s-1)) -$$
$$- l\,v_{y0}\,\omega_s\sin(\varphi_s)\,(1+2(n-s-1)) + 2\,l^2\sum_{i=0}^{s-1}\dot{\omega}_i\,\cos(\varphi_s-\varphi_i) -$$
$$- 2\,l^2\sum_{i=0}^{s-1}\left[\omega_i\omega_s\,\sin(\varphi_s-\varphi_i)+\omega_i^2\sin(\varphi_s-\varphi_i)\right] + l^2\dot{\omega}_{n-1}\,\cos(\varphi_s-\varphi_i) -$$
$$\left. - l^2\omega_s\omega_{n-1}\,\sin(\varphi_{n-1}-\varphi_s) + l^2\omega_{n-1}^2\,\sin(\varphi_{n-1}-\varphi_s)\ \right\}. \tag{2.29}$$

Nyní si připravíme derivace kinetické energie E_k podle proměnných x_0, y_0 a φ_s. Z podmínek rovnováhy lze odvodit, že:

$$\frac{\partial E_k}{\partial x_0} = 0, \quad \frac{\partial E_k}{\partial y_0} = 0. \tag{2.30}$$

Derivace podle úhlů pootočení dílců φ_s má potom tvar:

$$\frac{\partial E_k}{\partial \varphi_s} = \frac{1}{2}m\left\{\ - l\,v_{x0}\,\omega_s\cos(\varphi_s)\,(1+2(n-s-1)) - l\,v_{y0}\,\omega_s\sin(\varphi_s)\,(1+2(n-s-1)) - \right.$$
$$- 2\,l^2\sum_{i=0}^{s-1}\omega_i\,\omega_s\,\sin(\varphi_s-\varphi_i) + 2\,l^2\sum_{i=s+1}^{n-2}\omega_s\,\omega_i\,\sin(\varphi_i-\varphi_s) +$$
$$\left. + l^2\omega_s\,\omega_{n-1}\,\sin(\varphi_{n-1}-\varphi_s)\right\}, \tag{2.31}$$

$$\frac{\partial E_k}{\partial \varphi_{n-1}} = \frac{1}{2}m\left\{\ - l\,v_{x0}\,\omega_{n-1}\cos(\varphi_{n-1}) - l\,v_{y0}\,\omega_{n-1}\sin(\varphi_{n-1}) - \right.$$
$$\left. - l^2\sum_{i=0}^{n-2}\omega_i\,\omega_{n-1}\,\sin(\varphi_{n-1}-\varphi_i)\right\}. \tag{2.32}$$

Posledním krokem je získat derivace potenciální energie E_k podle úhlů pootočení dílců φ_s. Vyjdeme z výrazu (2.20) a získáme výsledek:

$$\frac{\partial E_p}{\partial \varphi_0} = k\,(\varphi_0 - \varphi_1), \tag{2.33}$$

$$\frac{\partial E_p}{\partial \varphi_s} = k\,(-\varphi_{s-1} + 2\varphi_s - \varphi_{s+1}), \tag{2.34}$$

$$\frac{\partial E_p}{\partial \varphi_{n-1}} = k\,(\varphi_{n-1} - \varphi_{n-2}), \tag{2.35}$$

$$\frac{\partial Ep}{\partial x_0} = 0, \tag{2.36}$$

$$\frac{\partial Ep}{\partial x_0} = 0. \tag{2.37}$$

V tuto chvíli je již vše potřebné připraveno a podle výrazů (2.21), (2.22) a (2.23) můžeme vytvořit soustavu rovnic:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial v_{x0}}\right) = \frac{\partial Ek}{\partial x_0} - \frac{\partial Ep}{\partial x_0}, \tag{2.38}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial v_{y0}}\right) = \frac{\partial Ek}{\partial y_0} - \frac{\partial Ep}{\partial y_0}, \tag{2.39}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \omega_s}\right) = \frac{\partial Ek}{\partial \varphi_s} - \frac{\partial Ep}{\partial \varphi_s},\ s = 0, 1, 2, \ldots, n-1. \tag{2.40}$$

2.3.1 Maticový tvar

Pro přehlednost zápisu převedeme soustavu rovnic do maticového tvaru:

$$\begin{aligned} \mathbf{M}\frac{\mathrm{d}}{\mathrm{d}t}(v+\omega) &= \mathbf{Q}\omega^2 - \mathbf{K}\varphi, \\ \frac{\mathrm{d}}{\mathrm{d}t}\varphi &= \omega, \\ \frac{\mathrm{d}}{\mathrm{d}t}s &= v. \end{aligned} \tag{2.41}$$

kde v, ω a φ jsou vektory stavových proměnných systému - translačních rychlostí, úhlových rychlostí a úhlů jednotlivých dílců.

Matice $\mathbf{M}$ je symetrická a reprezentuje hmotné momenty setrvačnosti dílců. Můžeme si všimnout, že při malých hodnotách vzájemného pootočení dílců (malých deformacích prutu) zůstane přibližně konstantní, k čemuž by došlo v případě linearizace řešení (momenty setrvačnosti dílců nejsou závislé na jejich pootočení), viz [2]. Rovněž je zde zřejmé rozdělení matice, kdy se první dva řádky vztahují k translačním rychlostem a zbylý obsah matice se týká rychlostí úhlových.

Matice $\mathbf{Q}$ obsahuje transformační momenty setrvačnosti. Není již čtvercová, avšak můžeme vidět, že spodní čtvercová část je antisymetrická s nulami na diagonále. Na rozdíl od matice $\mathbf{M}$ se zde hodnoty při malých deformacích prutu blíží nule. Význam matice $\mathbf{Q}$ tedy narůstá s deformací prutu a uplatní se pro nelineární řešení.

Maticový tvar pohybových rovnic uzavírá pásová matice ohybových tuhostí rotačních pružin $\mathbf{K}$. Pro dodržení formálně správného tvaru matice nesmíme zapomenout doplnit dva prázdné řádky (derivace podle v_{x0} *a* v_{y0}). Matice tuhosti má potom tvar:

$$\mathbf{K} = k \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \\ 1 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 1 \end{bmatrix}. \tag{2.42}$$

$$
\mathbf{M} = \frac{1}{2}m \begin{bmatrix}
2n & 0 & -l\sin\varphi_0 & -l\sin\varphi_1 & \cdots & -l\sin\varphi_i & -l\sin\varphi_{n-1} \\
 & 2n & (1+2(n-1)) & (1+2(n-2)) & \cdots & (1+2(n-i-1)) & \\
 & & l\cos\varphi_0 & l\cos\varphi_1 & \cdots & l\cos\varphi_i & l\cos\varphi_{n-1} \\
 & & (1+2(n-1)) & (1+2(n-2)) & \cdots & (1+2(n-i-1)) & \\
 & & 2l^2(\frac{1}{3}+(n-1)) & 2l^2\cos(\varphi_1-\varphi_0) & \cdots & 2l^2\cos(\varphi_i-\varphi_0) & l^2\cos(\varphi_{n-1}-\varphi_0) \\
 & & & 2l^2(\frac{1}{3}+(n-2)) & \cdots & 2l^2\cos(\varphi_i-\varphi_1) & l^2\cos(\varphi_{n-1}-\varphi_1) \\
 & & & & \ddots & \vdots & \vdots \\
 & & & \textit{symetrie} & & 2l^2(\frac{1}{3}+(n-i-1)) & l^2\cos(\varphi_{n-1}-\varphi_i) \\
 & & & & & & \frac{2}{3}l^2
\end{bmatrix}, \tag{2.43}
$$

$$\mathbf{Q} = \frac{1}{2}m \begin{bmatrix} l\cos\varphi_0 \\ (1+2(n-1)) & l\cos\varphi_1 \\ & (1+2(n-2)) & l\cos\varphi_2 \\ l\sin\varphi_0 & & (1+2(n-3)) & \cdots & l\cos\varphi_i \\ (1+2(n-1)) & l\sin\varphi_1 & & & (1+2(n-i-1)) \\ & (1+2(n-2)) & l\sin\varphi_2 & \cdots & l\sin\varphi_i \\ & & (1+2(n-3)) & & (1+2(n-i-1)) \\ 0 & l^2\sin(\varphi_1-\varphi_0) & & \cdots & l^2\sin(\varphi_i-\varphi_0) \\ & (1+2(n-2)) & l^2\sin(\varphi_2-\varphi_0) & & (1+2(n-i-1)) \\ & & (1+2(n-3)) & & \\ l^2\sin(\varphi_0-\varphi_1) & 0 & l^2\sin(\varphi_2-\varphi_1) & \cdots & l^2\sin(\varphi_i-\varphi_1) \\ (1+2(n-2)) & & (1+2(n-3)) & & (1+2(n-i-1)) \\ l^2\sin(\varphi_0-\varphi_2) & l^2\sin(\varphi_1-\varphi_2) & & \cdots & l^2\sin(\varphi_i-\varphi_2) \\ (1+2(n-3)) & (1+2(n-3)) & 0 & & (1+2(n-i-1)) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l^2\sin(\varphi_0-\varphi_i) & l^2\sin(\varphi_1-\varphi_i) & l^2\sin(\varphi_2-\varphi_i) & \cdots & 0 \\ (1+2(n-i-1)) & (1+2(n-i-1)) & (1+2(n-i-1)) & & \\ l^2\sin(\varphi_0-\varphi_{n-1}) & l^2\sin(\varphi_1-\varphi_{n-1}) & l^2\sin(\varphi_2-\varphi_{n-1}) & \cdots & l^2\sin(\varphi_i-\varphi_{n-1}) \end{bmatrix}$$

$$\begin{matrix} l\cos\varphi_{n-1} \\ \\ l\sin\varphi_{n-1} \\ \\ l^2\sin(\varphi_{n-1}-\varphi_0) \\ \\ l^2\sin(\varphi_{n-1}-\varphi_1) \\ \\ l^2\sin(\varphi_{n-1}-\varphi_2) \\ \\ \vdots \\ \\ l^2\sin(\varphi_{n-1}-\varphi_i) \\ \\ 0 \end{matrix} \quad . \tag{2.44}$$

2.4 Aplikovaný model volného prutu

V předchozí části jsme se zabývali odvozením konzervativního modelu volného prutu. Pro naše potřeby ale model ještě postrádá množství potřebných součástí. V prvé řadě je třeba doplnit do modelu sledující zatížení, jenž je nezbytnou složkou úlohy. Dále je zapotřebí postihnout různé varianty disipace energie, neboť varianta konzervativního modelu není pro simulaci reálného chování výstižná. Poslední nezbytnou součástí je zahrnutí vlivu gravitačního pole.

2.4.1 Sledující síla

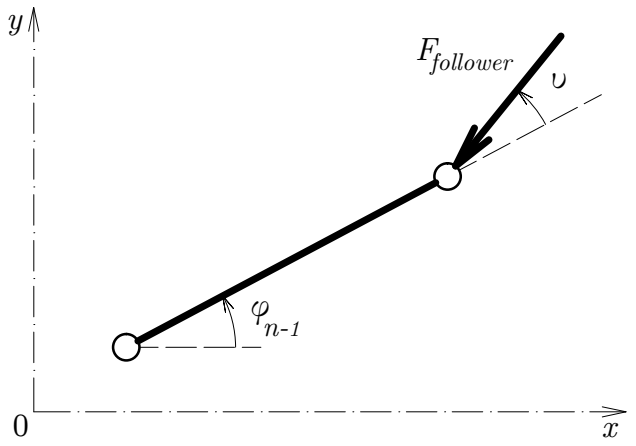
Sledující síla je hlavní součástí modelu, neboť nahrazuje tah reaktivního motoru v reálné úloze. Sledující síla mění svůj směr v závislosti na deformaci konstrukce, viz [10]. V našem případě tak, že působí pod konstantním úhlem vůči konci volného prutu. Přidání účinků sledující síly tedy dává pohybovým rovnicím (2.41) tvar:

$$\mathbf{M}\frac{d}{dt}(v+\omega) = \mathbf{Q}\omega^2 - \mathbf{K}\varphi + \mathbf{F}, \tag{2.45}$$

kde $\mathbf{F}$ je vektor účinků sledující síly.

Sledující sílu označme jako $F_{follower}$ a úhel sevřený s posledním dílcem modelu potom jako υ, viz obrázek 2.4. Dílčí složky sledující síly $F_{follower,x}$ *a* $F_{follower,y}$ potom můžeme vyjádřit jako:

$$\begin{aligned} F_{follower,x} &= F_{follower}\cos(\varphi_{n-1}+\upsilon), \\ F_{follower,y} &= F_{follower}\sin(\varphi_{n-1}+\upsilon). \end{aligned} \tag{2.46}$$



Obrázek 2.4: Působení sledující síly.

Vektor vlivu sledující síly **F** vznikne složením translačního působení sledující síly a momentů sledující síly vůči jednotlivým kloubům:

$$\mathbf{F} = \begin{bmatrix} F_x \\ F_y \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} + \mathrm{F}_{follower,x}\, l \begin{bmatrix} 0 \\ 0 \\ \sin(\varphi_0) \\ \sin(\varphi_1) \\ \vdots \\ \sin(\varphi_{n-2}) \\ \sin(\varphi_{n-1}) \end{bmatrix} - \mathrm{F}_{follower,y}\, l \begin{bmatrix} 0 \\ 0 \\ \cos(\varphi_0) \\ \cos(\varphi_1) \\ \vdots \\ \cos(\varphi_{n-2}) \\ \cos(\varphi_{n-1}) \end{bmatrix}. \tag{2.47}$$

2.4.2 Materiálové tlumení

Je zřejmé, že výstižná formulace materiálového tlumení vyžaduje experimentální ověření při vhodných podmínkách. Předložená definice bude proto později ověřena experimentem.

Při formulaci materiálového tlumení vycházíme z nejjednodušší definice lineárního viskózního tlumení:

$$M = -c\, m\, \omega, \tag{2.48}$$

kde c je koeficient útlumu, m je hmotnost dílce a ω je vzájemná relativní rychlost sousedních dílců. Pro popisovaný model tedy platí:

$$M = -c_{int}\, m\, \frac{\mathrm{d}}{\mathrm{d}t}\Delta\varphi = -c_{int}\, m\, \omega_r, \tag{2.49}$$

kde c_{int} je koeficient materiálového útlumu a ω_r je rychlost rozevírání nebo svírání dvou sousedních dílců. Pro tlumící moment mezi s-tým a $s{+}1$ dílcem tedy zřejmě platí:

$$M = -c_{int}\, m\, (\omega_{s+1} - \omega_s). \tag{2.50}$$

Matici materiálového útlumu označme $\mathbf{D}_i$. Pohybové rovnice (2.41) tedy získají následující podobu:

$$\mathbf{M}\frac{d}{dt}(v + \omega) = \mathbf{Q}\omega^2 - \mathbf{K}\varphi - \mathbf{D}_i\omega, \tag{2.51}$$

kde ω je vektor úhlových rychlostí dílců.

Matice materiálového tlumení $\mathbf{D}_i$ tedy nabývá tvaru:

$$\mathbf{D}_i = c_{int}\, m \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \\ 1 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 1 \end{bmatrix}. \tag{2.52}$$

2.4.3 Tlumení vlivem okolního prostředí

Do chování modelu chceme zahrnout i disipaci energie vlivem okolního prostředí. Vektor tlumení vlivem okolního prostředí označíme $\mathbf{D}_e$. Upravme tedy pohybové rovnice (2.41) do tvaru:

$$\mathbf{M}\frac{d}{dt}(v+\omega) = \mathbf{Q}\omega^2 - \mathbf{K}\varphi - \mathbf{D}_e. \tag{2.53}$$

Nejprve definujme tlumící sílu obecně jako:

$$D = c\, m\, v, \tag{2.54}$$

kde c je koeficient útlumu, m je hmotnost dílce a v je rychlost dílce. Abychom postihli závislost velikosti útlumu na směru pohybu, vztah (2.54) zapíšeme jako:

$$D = c_{ext}\, a_i\, m\, v, \tag{2.55}$$

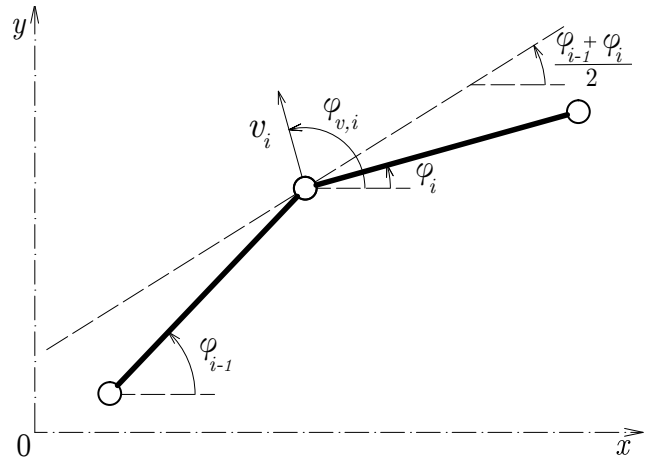
kde c_{ext} je koeficient útlumu vlivem okolního prostředí a a_i je koeficient zohledňující vliv rozdílného natočení dílců vůči směru jejich pohybu, pro který platí:

$$a_i = \left|\sin\left(\varphi_{v,i} - \frac{\varphi_{i-1}+\varphi_i}{2}\right)\right|, \tag{2.56}$$

kde $\varphi_{v,i}$ je úhel vektoru absolutní rychlosti dílce, který získáme podle vztahu:

$$\varphi_{v,i} = \arctan\left(\frac{v_{y,i}}{v_{x,i}}\right). \tag{2.57}$$

Přehledně je odvození znázorněno na obrázku 2.5.:



Obrázek 2.5: Odvození závislosti útlumu na směru pohybu modelu.

Při pohledu na výraz (2.56) si všimněme, že v případě malého rozdílu průměrného úhlu natočení sousedních dílců a směru jejich pohybu bude hodnota koeficientu a_i velmi malá. Naopak s rostoucím rozdílem těchto úhlů koeficient a_i nabývá maximálních hodnot a tlumení je tak nejvýraznější. Díky této formulaci se intenzita tlumení vlivem okolního prostředí bude v každém místě modelu adekvátně lišit.

Do výrazu pro tlumící sílu navíc vložíme člen s_i, kterým zavedeme vliv stabilizace rakety pomocí křidélek. V místech modelu, kde bude požadováno toto přídavné tlumení, je hodnota parametru s_i větší než jedna. Tření o plášť rakety bude zanedbáno. Obecně tedy platí:

$$D = c_{ext}\, s_i\, a_i\, m\, v, \tag{2.58}$$

Pro tlumící síly působící v kloubech modelu nakonec můžeme psát:

$$\begin{aligned}
D_{x,0} &= c_{ext}\, s_0\, a_0\, \frac{m}{2}\, v_{x,0},\\
D_{y,0} &= c_{ext}\, s_0\, a_0\, \frac{m}{2}\, v_{y,0},\\
D_{x,i} &= c_{ext}\, s_i\, a_i\, m\, v_{x,i}, \quad i = 1, 2, \ldots, n-1,\\
D_{y,i} &= c_{ext}\, s_i\, a_i\, m\, v_{y,i}, \quad i = 1, 2, \ldots, n-1,\\
D_{x,n} &= c_{ext}\, s_n\, a_n\, \frac{m}{2}\, v_{x,n},\\
D_{y,n} &= c_{ext}\, s_n\, a_n\, \frac{m}{2}\, v_{y,n}.
\end{aligned} \tag{2.59}$$

Vektor tlumení vlivem okolního prostředí $\mathbf{D}_e$ se skládá z translačních účinků výslednice tlumících sil a momentů tlumících sil k jednotlivým kloubům modelu. Výsledný tvar je tedy následující:

$$\mathbf{D}_e = \begin{bmatrix} \sum_{i=0}^{n} D_{x,i} \\ \sum_{i=0}^{n} D_{y,i} \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} - l \begin{bmatrix} 0 \\ 0 \\ \sin(\varphi_0) \sum_{i=1}^{n} D_{x,i} \\ \sin(\varphi_1) \sum_{i=2}^{n} D_{x,i} \\ \vdots \\ \sin(\varphi_{k-1}) \sum_{i=k}^{n} D_{x,i} \\ \sin(\varphi_{n-1}) D_{x,n} \end{bmatrix} + l \begin{bmatrix} 0 \\ 0 \\ \cos(\varphi_0) \sum_{i=1}^{n} D_{y,i} \\ \cos(\varphi_1) \sum_{i=2}^{n} D_{y,i} \\ \vdots \\ \cos(\varphi_{k-1}) \sum_{i=k}^{n} D_{y,i} \\ \cos(\varphi_{n-1}) D_{y,n} \end{bmatrix}. \tag{2.60}$$

2.4.4 Gravitační zrychlení

Poslední zbývající část aplikovaného modelu je vliv působení gravitačního pole. Přidejme tedy do pohybových rovnic (2.41) vektor gravitačního zrychlení **A**:

$$\mathbf{M}\frac{d}{dt}(v+\omega) = \mathbf{Q}\omega^2 - \mathbf{K}\varphi + \mathbf{A}. \tag{2.61}$$

Velikost tíhové síly F_g působící v těžišti tuhého dílce odvodíme snadno z předpokladu:

$$F_g = m\,g, \tag{2.62}$$

kde m je hmotnost dílce a g je intenzita gravitačního pole. Podobně pro setrvačnou sílu F_a působící v těžišti tuhého dílce platí:

$$F_a = m\,a_g, \tag{2.63}$$

kde m je setrvačná hmotnost dílce a a_g je zrychlení působící na dílec.

Vektor gravitačního zrychlení **A** je tedy složen z translačních účinků setrvačných sil a z momentů setrvačných sil ke kloubům modelu:

$$\mathbf{A} = \begin{bmatrix} 0 \\ M\,a_g \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} - \mathrm{a}_g\,m\,l \begin{bmatrix} 0 \\ 0 \\ (n-\frac{1}{2})\,\cos(\varphi_0) \\ (n-\frac{1}{2}-1)\,\cos(\varphi_1) \\ \vdots \\ (n-\frac{1}{2}-i)\,\cos(\varphi_i) \\ \frac{1}{2}\,\cos(\varphi_{n-1}) \end{bmatrix}. \tag{2.64}$$

2.4.5 Aplikovaný model

Výsledná podoba aplikovaného modelu vznikne složením všech zmíněných dílčích složek, viz (2.45), (2.51), (2.53) a (2.61). Konečný tvar aplikovaného modelu je tedy následující:

$$\begin{aligned} &\mathbf{M}\frac{d}{dt}(v+\omega) = \mathbf{Q}\omega^2 - \mathbf{K}\varphi - \mathbf{D}_i - \mathbf{D}_e\omega + \mathbf{F} + \mathbf{A}, \\ &\frac{d}{dt}\varphi = \omega, \\ &\frac{d}{dt}s = v. \end{aligned} \tag{2.65}$$

Kapitola 3

Výsledky numerických simulací

3.1 Vlastní frekvence prutu

Pro zjištění vlastních frekvencí konstrukce je zapotřebí vhodně zvolit budící impuls. V případě volného prutu je obtížné prut rozkmitat se současným zachováním rovnováhy sil. Mohou tedy vzniknout nežádoucí posuny nebo pootočení, které komplikují sledování vibrací.

Volný prut můžeme vybudit symetricky, v takovém případě ovšem obdržíme pouze vlastní frekvence se symetrickými amplitudami. Z toho důvodu byl model upraven také do podoby, kdy je počátek prutu pevně vetknut a konstrukce tedy působí jako konzola.

Simulace byly provedeny pro ocelový prut o délce $L = 2\,\mathrm{m}$, průřezové ploše $A = 0.001\,\mathrm{m}^2$, momentu setrvačnosti $I = 8.33 \times 10^{-9}\,\mathrm{m}^4$, modulu pružnosti $E = 210 \times 10^9\,\mathrm{Pa}$ a hmotnosti $m = 15\,700\,\mathrm{g}$. Prut byl rozdělen na 40 tuhých dílců. Postupy analytického řešení, jakož i kořeny frekvenčních rovnic, byly převzaty, viz [17]. Krok výpočtu byl nastaven na $8 \times 10^{-4}\,\mathrm{s}$, simulovaný čas byl 50 s. Výsledky frekvenční analýzy můžeme vidět v tabulce 3.1.

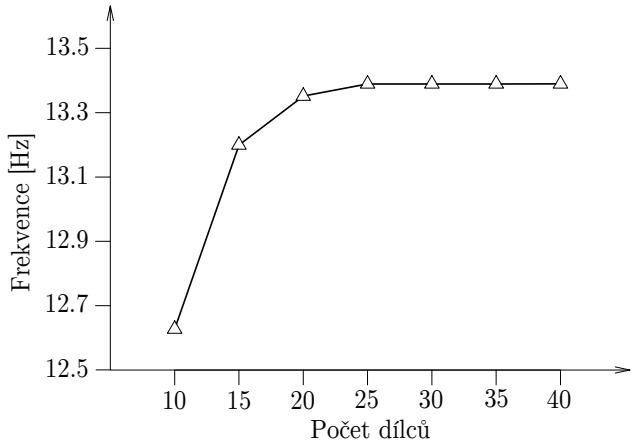
Frekvence	Analytické řešení [Hz]	Simulace [Hz]	Odchylka [%]
f_1, volný prut	13.2876	13.3896	+0.772
f_3, volný prut	71.8056	72.7841	+1.363
f_5, volný prut	177.3140	176.1263	−0.670
f_1, konzola	2.1966	2.1352	−2.685
f_2, konzola	13.7662	13.7785	+0.211
f_3, konzola	38.5457	38.4750	−0.059
f_4, konzola	75.5343	75.6227	+0.241
f_5, konzola	124.8640	125.2371	+0.299

Tabulka 3.1: Srovnání vypočtených výsledků s analytickým řešením vlastních frekvencí.

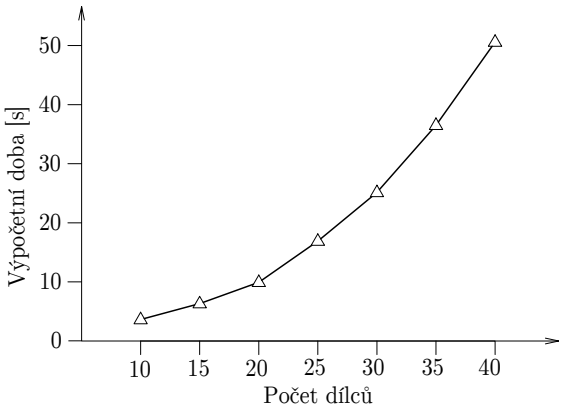
Tabulka 3.2 potom ukazuje konvergenci hodnot k první vlastní frekvenci volného prutu a časovou náročnost simulace. Výsledky jsou znázorněny na obrázku 3.1.

Počet dílců	Frekvence [Hz]	Výpočetní doba [s]
10	12.6271	3.624
15	13.1993	6.325
20	13.3815	9.928
25	13.3893	16.904
30	13.3895	25.137
35	13.3896	36.510
40	13.3896	50.581

Tabulka 3.2: Konvergence hodnoty první vlastní frekvence a odpovídající výpočetní doba.



Obrázek 3.1: Konvergence hodnoty první vlastní frekvence volného prutu.



Obrázek 3.2: Závislost výpočetní doby na počtu dílců.

3.2 Konzola zatížená vlastní tíhou

Dalším parametrem, který byl ověřen, je průhyb a pootočení volného konce konzoly zatížené vlastní tíhou q, viz obrázek 3.3. Toto ověření je zároveň ověřením správnosti účinků gravitačního pole. Výpočet byl proveden pro ocelový prut o délce $L = 5\,m$, ohybové tuhosti $EI = 100Nm^2$ a hmotnosti $m = 50\,g$.

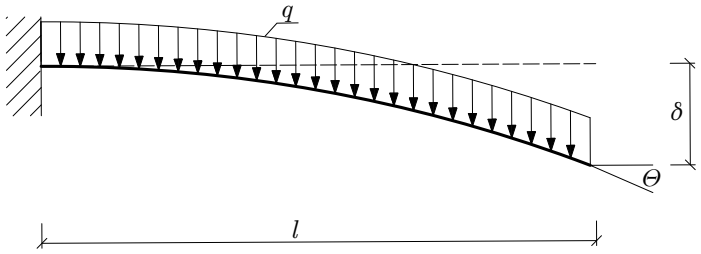
Konzolového uložení prutu je dosaženo fixací polohy prvního kloubu modelu a pootočení prvního dílce. Z toho důvodu je skutečná délka konzoly $L = (5 - 5/n)$, kde n je počet dílců. Simulace probíhala s modelem o 50 dílcích. Vzorce pro analytické řešení byly převzaty, viz [11].

Průhyb volného konce konzoly δ lze analyticky vyjádřit:

$$\delta = \frac{q\,l^4}{8EI} = \frac{\frac{0{,}05\times 9.81}{5-5/50} \times (5 - 5/50)^4}{8 \times 100} = \mathbf{0.0721}\,m. \tag{3.1}$$

Pro pootočení volného konce konzoly Θ platí vztah:

$$\Theta = \frac{q\,l^3}{6EI} = \frac{\frac{0{,}05\times 9.81}{5-5/50} \times (5 - 5/50)^3}{6 \times 100} = \mathbf{0.0196}\,rad \tag{3.2}$$



Obrázek 3.3: Konzola zatížená vlastní tíhou.

Srovnání výsledků z numerického výpočtu s analytickým řešením nabízí tabulka 3.3.

	Analytické řešení	Simulace	Odchylka [%]
Průhyb δ [m]	0.0721	0.0735	+1.904
Pootočení Θ [rad]	0.0196	0.0198	+1.020

Tabulka 3.3: Porovnání výsledků průhybu a pootočení volního konce konzoly.

3.3 Kritická síla

Ztráta stability konstrukce a hodnota kritické síly jsou velmi vhodné nástroje pro ověření správnosti chování modelu. Jev ztráty stability u numerického modelu potvrzuje kvalitativně stejné nelineární chování jako u reálné konstrukce. Protože ke ztrátě stability dochází náhle, je samotná hodnota kritické síly výborným kvantitativním ověřením chování modelu, viz [3].

3.3.1 Volný prut

Simulace pro získání hodnoty kritické síly volného prutu byly prováděny pro ocelový prut o délce $L = 5\,\mathrm{m}$ a ohybové tuhosti $EI = 100\,\mathrm{Nm}^2$. Simulovaný čas byl 200 sekund. Získané hodnoty kritické síly byly dále porovnány s výsledky ze simulací provedených pro stejný model v aplikaci FyDiK, viz [6]. Výsledky z numerických simulací byly dále porovnány s analytickým řešením, viz [9]:

$$F_{cr} \approx 109.54\frac{EI}{L^2} = 109.54\frac{100}{5^2} = \mathbf{438.16}\,N. \tag{3.3}$$

Výsledky numerických simulací speciálního modelu - obdržené hodnoty kritické síly, jejich odchylky oproti analytickému řešení (3.3) a odpovídající výpočetní doba - jsou přehledně zobrazeny v tabulce 3.4.

Počet dílců	Kritická síla [N]	Odchylka [%]	Výpočetní doba [s]
5	326.12	−25.571	1.255
10	411.83	−6.009	2.397
15	426.81	−2.590	4.326
20	432.05	−1.394	7.344
25	434.49	−0.838	12.042
30	435.80	−0.539	18.783
35	436.59	−0.358	27.486
40	437.14	−0.233	38.478
45	437.48	−0.155	51.829
50	**437.91**	**−0.057**	**71.481**

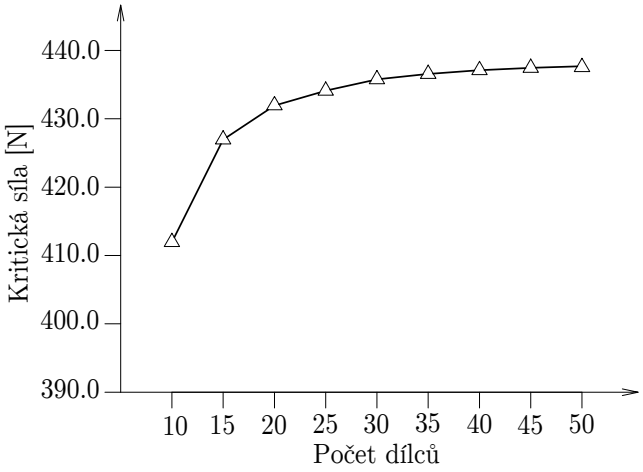
Tabulka 3.4: Výsledky simulací speciálního modelu volného prutu.

Výsledky obdržené při simulacích v aplikaci FyDiK ukazuje podobně strukturovaná tabulka 3.5.

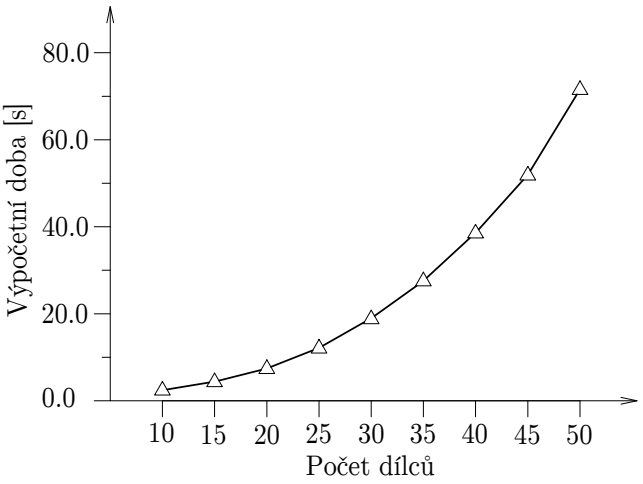
Počet dílců	Kritická síla [N]	Odchylka [%]	Výpočetní doba [s]
10	400.73	−8.543	27.666
20	433.27	−1.116	84.761
30	439.52	+0.310	216.790
40	441.73	+0.815	444.110
50	**442.75**	**+1.048**	**825.040**

Tabulka 3.5: Výsledky simulací provedených v aplikaci FyDiK.

Grafické znázornění konvergence kritické síly nabízí obrázek 3.4, nárůst výpočetní doby v závislosti na počtu dílců modelu potom obrázek 3.5.



Obrázek 3.4: Konvergence hodnoty kritické síly volného prutu.



Obrázek 3.5: Nárůst výpočetní doby s počtem stupňů volnosti.

3.3.2 Konzola

Hodnota kritické síly byla ověřena i na konzolovém prutu. Model zůstává totožný, tedy ocelový prut o délce $L = 5\,\mathrm{m}$ a ohybové tuhosti $EI = 100\,\mathrm{Nm}^2$. Výsledky numerické simulace budou porovnány s analytickým řešením, viz [1] a [3]:

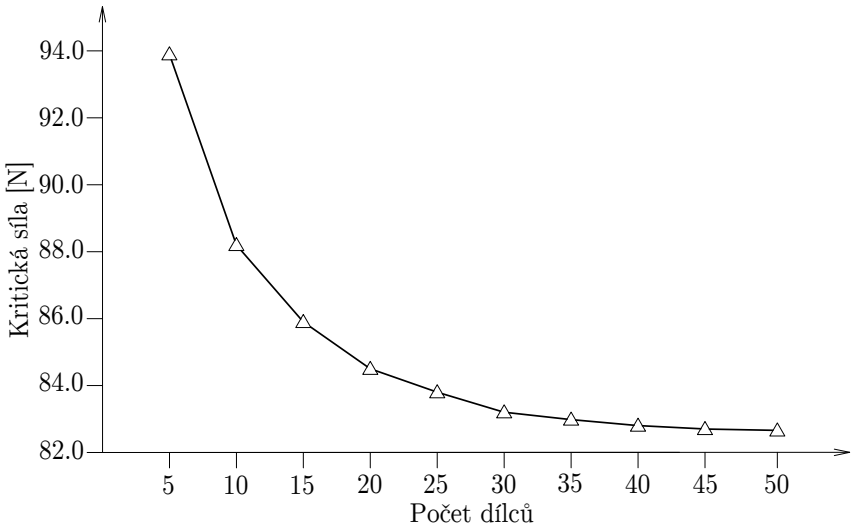
$$F_{cr} \approx 20.05 \frac{EI}{L^2}. \tag{3.4}$$

Jak bylo zmíněno, délka vlastní konzoly je vždy $L = (5 - 5/n)$. Kritická síla tedy navíc závisí na počtu dílců a liší se tak pro každý případ. Výsledky numerických simulací ve srovnání s analytickým řešením můžeme vidět v tabulce 3.6.

Počet dílců	Kritická síla, simulace [N]	Kritická síla, analytické řešení [N]	Odchylka [%]
5	93.95	125.31	−33.382
10	88.23	99.01	−12.221
15	85.85	92.06	−7.241
20	84.51	88.87	−5.152
25	83.75	87.02	−3.908
30	83.21	85.83	−3.144
35	82.98	84.99	−2.419
40	82.83	84.37	−1.845
45	82.70	83.89	−1.435
50	**82.65**	**83.51**	**−1.037**

Tabulka 3.6: Výsledky simulací speciálního modelu konzoly.

Grafické znázornění konvergence výsledků nabízí obrázek 3.6



Obrázek 3.6: Konvergence hodnoty kritické síly konzoly.

Kapitola 4

Implementace modelu

Implementace numerického modelu byla provedena v programovacím jazyku Java s kompilátorem Java Development Kit, viz [21]. Implementace probíhala v prostředí Eclipse IDE for Java Developers, viz [22]. Pro frekvenční analýzu kmitání prutu je použita Rychlá Fourierova transformace (FFT), viz [4].

4.1 Implementace metod řešení dynamického systému

Dále jsou uvedeny zdrojové kódy ukazující průběh výpočtu jednoho kroku jednotlivými numerickými metodami. Pro zpřístupnění kódu čtenáři nejdříve následuje stručný popis jednotlivých metod:

- **assembleMatrixM()** - sestaví matici hmotných momentů setrvačnosti $\mathbf{M}$.
- **assembleMatrixQ()** - sestaví matici transformačních momentů setrvačnosti $\mathbf{Q}$.
- **assembleMatrixK()** - sestaví matici tuhostí rotačních pružin $\mathbf{K}$.
- **assembleMatrixD()** - sestaví matici materiálového útlumu $\mathbf{D}_i$.
- **createEquatuion()** - sestaví zbývající vektory v soustavě rovnic: $\mathbf{D}_e, \mathbf{F}, \mathbf{A}$.
- **getSolution()** - vyřeší soustavu rovnic Gaussovou eliminační metodou, vrátí vektor řešení.
- **updateAngles()** - aktualizuje úhly pootočení tuhých dílců.
- **updateCoords()** - aktualizuje souřadnice kloubů spojujících tuhé dílce.
- **updateOmegas()** - aktualizuje úhlové rychlosti tuhých dílců.
- **updateVelocities()** - aktualizuje translační rychlosti kloubů spojujících tuhé dílce.
- **updateCG()** - vypočítá polohu těžiště modelu z aktuálních souřadnic.
- **updateKineticEnergy()** - vypočítá aktuální kinetickou energii modelu.
- **updatePotentialEnergy()** - vypočítá aktuální potenciální energii rotačních pružin.

- **updateOverallEnergy()** - vrátí celkovou energii modelu.
- **updateTime()** - přičte časový krok k předchozímu času simulace.

4.1.1 Eulerova metoda

Z implementace Eulerovy metody je zřejmá jednoduchost numerického řešení nelineární dynamické úlohy. Jediná náročná část je řešení soustavy rovnic Gaussovou eliminační metodou.

```
705     /* Euler method */
706     public void stepEuler()
707     {
708       assembleMatrixM();
709       assembleMatrixQ();
710       assembleMatrixK();
711       assembleMatrixD();
712       //
713       createEquation();
714       //
715       getSolution(leftSide, rightSide);
716       //
717       updateAngles();
718       updateCoords();
719       //
720       updateOmegas();
721       updateVelocities();
722       //
723       updateCG();
724       //
725       updateKineticEnergy();
726       updatePotentinalEnergy();
727       updateOverallEnergy();
728       //
729       updateTime();
730     }
```

4.1.2 Semi-implicitní Eulerova metoda

Jediným rozdílem oproti klasické Eulerově metodě je záměna pořadí metod updateAngles() a updateCoords() s metodami updateOmegas() a updateVelocities(), viz sekce 1.4.3.

```
747     /* Semi-implicit Euler method */
748     public void stepImprovedEuler()
749     {
750       assembleMatrixM();
751       assembleMatrixQ();
752       assembleMatrixK();
753       assembleMatrixD();
754       //
755       createEquation();
756       //
757       getSolution(leftSide, rightSide);
758       //
759       updateOmegas();
```

```
760       updateVelocities();
761       //
762       updateAngles();
763       updateCoords();
764       //
765       updateCG();
766       //
767       updateKineticEnergy();
768       updatePotentinalEnergy();
769       updateOverallEnergy();
770       //
771       updateTime();
772     }
```

4.1.3 Runge-Kuttova metoda

Runge-Kuttova metoda je čtyřbodová numerická metoda. Proto je v rámci jednoho kroku potřeba čtyřikrát sestavit a vyřešit soustavu rovnic. Je zřejmé, že tato skutečnost bude vést ke značné časové náročnosti řešení.

```
800     /** Runge-Kutta method */
801     public void stepRungeKutta()
802     {
803       Vector RKsolutionK1=new Vector();
804       Vector RKsolutionK2=new Vector();
805       Vector RKsolutionK3=new Vector();
806       Vector RKsolutionK4=new Vector();
807       //
808       /* Saving values xi(t) */
809       for (int i=0; i<angles.length; i++)
810       {
811         anglesT[i]=angles[i];
812         omegasT[i]=omegas[i];
813       }
814       for (int i=0; i<coordinates.length; i++)
815       {
816         coordinatesT[i]=coordinates[i];
817         velocitiesT[i]=velocities[i];
818       }
819       //
820       /* k1 x(t) *////////////////////////////
821       assembleMatrixM();
822       assembleMatrixQ();
823       assembleMatrixK();
824       assembleMatrixC();
825       //
826       createEquation();
827       //
828       getSolution(leftSide, rightSide);
829       //
830       updateOmegas();
831       updateVelocities();
832       //
833       updateAngles();
834       updateCoords();
```

```
835         //
836         RKsolutionK1.allocateFor(solution);
837         //
838         for (int row=0; row<solution.getRowCount(); row++)
839         {
840           RKsolutionK1.addValue(row, solution.getValue(row));
841         }
842
843         /* k2 (xi(t)+k1i*h/2) *////////////////////////////
844         /* input for k2*/
845         for (int row=0; row<solution.getRowCount(); row++)
846         {
847           solution.setValue(row, RKsolutionK1.getValue(row)/2d);
848         }
849         //
850         for (int i=0; i<angles.length; i++)
851         {
852           angles[i]=anglesT[i];
853           omegas[i]=omegasT[i];
854         }
855         for (int i=0; i<coordinates.length; i++)
856         {
857           coordinates[i]=coordinatesT[i];
858           velocities[i]=velocitiesT[i];
859         }
860         //
861         /* solving for k2 */
862         updateOmegas();
863         updateVelocities();
864         updateAngles();
865         updateCoords();
866         //
867         //
868         assembleMatrixM();
869         assembleMatrixQ();
870         assembleMatrixK();
871         assembleMatrixC();
872         //
873         createEquation();
874         //
875         getSolution(leftSide, rightSide);
876         //
877         RKsolutionK2.allocateFor(solution);
878         //
879         for (int row=0; row<solution.getRowCount(); row++)
880         {
881           RKsolutionK2.addValue(row, solution.getValue(row));
882         }
883         //
884         //
885         /* k3 (xi(t)+k2i*h/2 *////////////////////////////
886         /* input for k3*/
887         for (int row=0; row<solution.getRowCount(); row++)
888         {
889           solution.setValue(row, RKsolutionK2.getValue(row)/2d);
890         }
```

```
891         //
892         for (int i=0; i<angles.length; i++)
893         {
894           angles[i]=anglesT[i];
895           omegas[i]=omegasT[i];
896         }
897         for (int i=0; i<coordinates.length; i++)
898         {
899           coordinates[i]=coordinatesT[i];
900           velocities[i]=velocitiesT[i];
901         }
902         //
903         /* solving for k3 */
904         //
905         updateOmegas();
906         updateVelocities();
907         updateAngles();
908         updateCoords();
909         //
910         //
911         assembleMatrixM();
912         assembleMatrixQ();
913         assembleMatrixK();
914         assembleMatrixC();
915         //
916         createEquation();
917         //
918         getSolution(leftSide, rightSide);
919         //
920         RKsolutionK3.allocateFor(solution);
921         //
922         for (int row=0; row<solution.getRowCount(); row++)
923         {
924           RKsolutionK3.addValue(row, solution.getValue(row));
925         }
926         //
927         //
928         /* k4 (xi(t)+k3i*h) *//////////////////////////
929         for (int row=0; row<solution.getRowCount(); row++)
930         {
931           solution.setValue(row, RKsolutionK3.getValue(row));
932         }
933         //
934         for (int i=0; i<angles.length; i++)
935         {
936           angles[i]=anglesT[i];
937           omegas[i]=omegasT[i];
938         }
939         for (int i=0; i<coordinates.length; i++)
940         {
941           coordinates[i]=coordinatesT[i];
942           velocities[i]=velocitiesT[i];
943         }
944         //
945         /* solving for k4 */
946         updateOmegas();
```

```
947         updateVelocities();
948         updateAngles();
949         updateCoords();
950         //
951         //
952         assembleMatrixM();
953         assembleMatrixQ();
954         assembleMatrixK();
955         assembleMatrixC();
956         //
957         createEquation();
958         //
959         getSolution(leftSide, rightSide);
960         //
961         RKsolutionK4.allocateFor(solution);
962         //
963         for (int row=0; row<solution.getRowCount(); row++)
964         {
965           RKsolutionK4.addValue(row, solution.getValue(row));
966         }
967         //
968         //
969         /* final solution xi(t+h)=xi(t)+h/6(k1i+2k2i+2k3i+k4i) */
970         for (int row=0; row<solution.getRowCount(); row++)
971           {
972             solution.setValue(row, (   RKsolutionK1.getValue(row)+
973                         2d*RKsolutionK2.getValue(row)+
974                         2d*RKsolutionK3.getValue(row)+
975                            RKsolutionK4.getValue(row))/6d);
976           }
977         //
978         for (int i=0; i<angles.length; i++)
979         {
980           angles[i]=anglesT[i];
981           omegas[i]=omegasT[i];
982         }
983         for (int i=0; i<coordinates.length; i++)
984         {
985           coordinates[i]=coordinatesT[i];
986           velocities[i]=velocitiesT[i];
987         }
988         //
989         updateOmegas();
990         updateVelocities();
991         //
992         updateAngles();
993         updateCoords();
994         //
995         updateCG();
996         //
997         updateKineticEnergy();
998         updatePotentinalEnergy();
999         updateOverallEnergy();
1000        //
1001        updateTime();
1002    }
```

4.2 Srovnání metod řešení dynamického systému

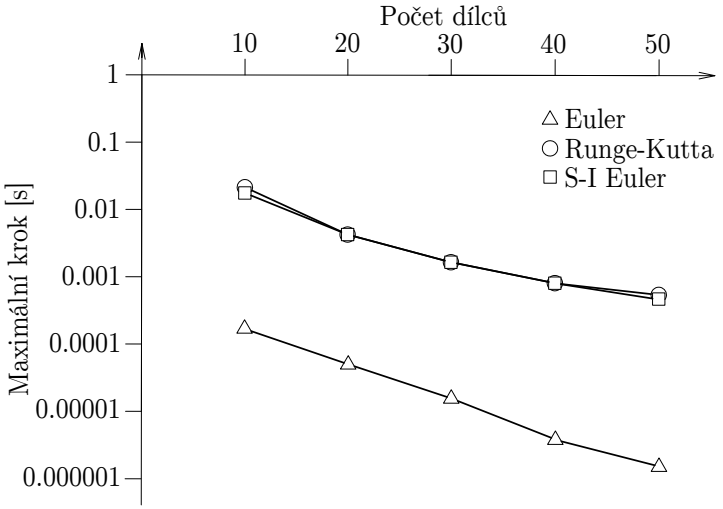
Pro úplnost je na místě nabídnout vzájemné porovnání použitých numerických metod pro řešení dynamického systému.

Maximální časový krok

Maximálním časovým krokem rozumíme takový krok h_{max}, při kterém je numerické řešení ještě stabilní. Porovnání výsledků nabízí tabulka 4.1 a obrázek 4.1.

Počet dílců	h_{max}, Euler [s]	h_{max}, S-I Euler [s]	h_{max}, Runge-Kutta [s]
10	0.000 2	0.02	0.025
20	0.000 06	0.005	0.005
30	0.000 02	0.002	0.002
40	0.000 01	0.001	0.001
50	0.000 002	0.000 6	0.000 7

Tabulka 4.1: Maximální časový krok v závislosti na počtu dílců a metodě.



Obrázek 4.1: Maximální časový krok v závislosti na počtu dílců a metodě.

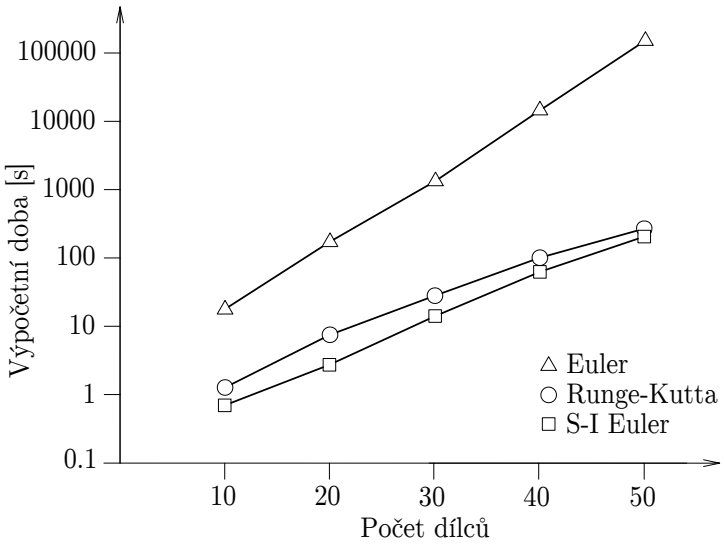
Z výsledků je patrné, že klasická Eulerova metoda je o dva až tři řády méně stabilní než semi-implicitní Eulerova metoda a metoda Runge-Kutta, pro které jsou výsledky velmi podobné a závisí na konkrétních podmínkách simulace.

Časová náročnost výpočtu

Výpočetní doba při maximálním kroku h_{max} závisí také na výpočetní náročnosti jednotlivých metod. Jak bude ukázáno, výpočetní náročnost znevýhodňuje čtyřbodovou metodu Runge-Kutta. Výsledky nabízí tabulka 4.2 a dále ilustruje obrázek 4.2.

Počet dílců	h_{max}, Euler [s]	h_{max}, S-I Euler [s]	h_{max}, Runge-Kutta [s]
10	18.506	0.694	1.328
20	181.768	2.739	8.126
30	1 385.120	14.253	31.485
40	16 253.156	63.996	111.889
50	151 288.201	208.206	263.767

Tabulka 4.2: Výpočetní doba v závislosti na počtu dílců a metodě.



Obrázek 4.2: Výpočetní doba v závislosti na počtu dílců a metodě.

Výsledná časová náročnost výpočtu mírně favorizuje semi-implicitní Eulerovu metodu. Nicméně je na místě připomenout, že oproti metodě Runge-Kutta má semi-implicitní Eulerova metoda pouze empirický původ a nemusí tak být vždy spolehlivá.

4.3 Grafické prostředí modelu

Pro zobrazování průběhu simulace bylo dále vytvořeno grafické prostředí, které umožňuje sledovat chování modelu v průběhu simulace. Zároveň nabízí množství dalších možností zobrazování, jak bude ukázáno dále.

Základní prostředí

Základní grafické prostředí modelu je znázorněno na obrázku 4.3. Graficky zobrazuje:

- vlastní model - tuhé dílce spojené rotačními pružinami,
- těžiště modelu,
- lokální souřadný systém - hlavní osy setrvačnosti,
- působící sledující sílu a znázornění gravitačního pole.

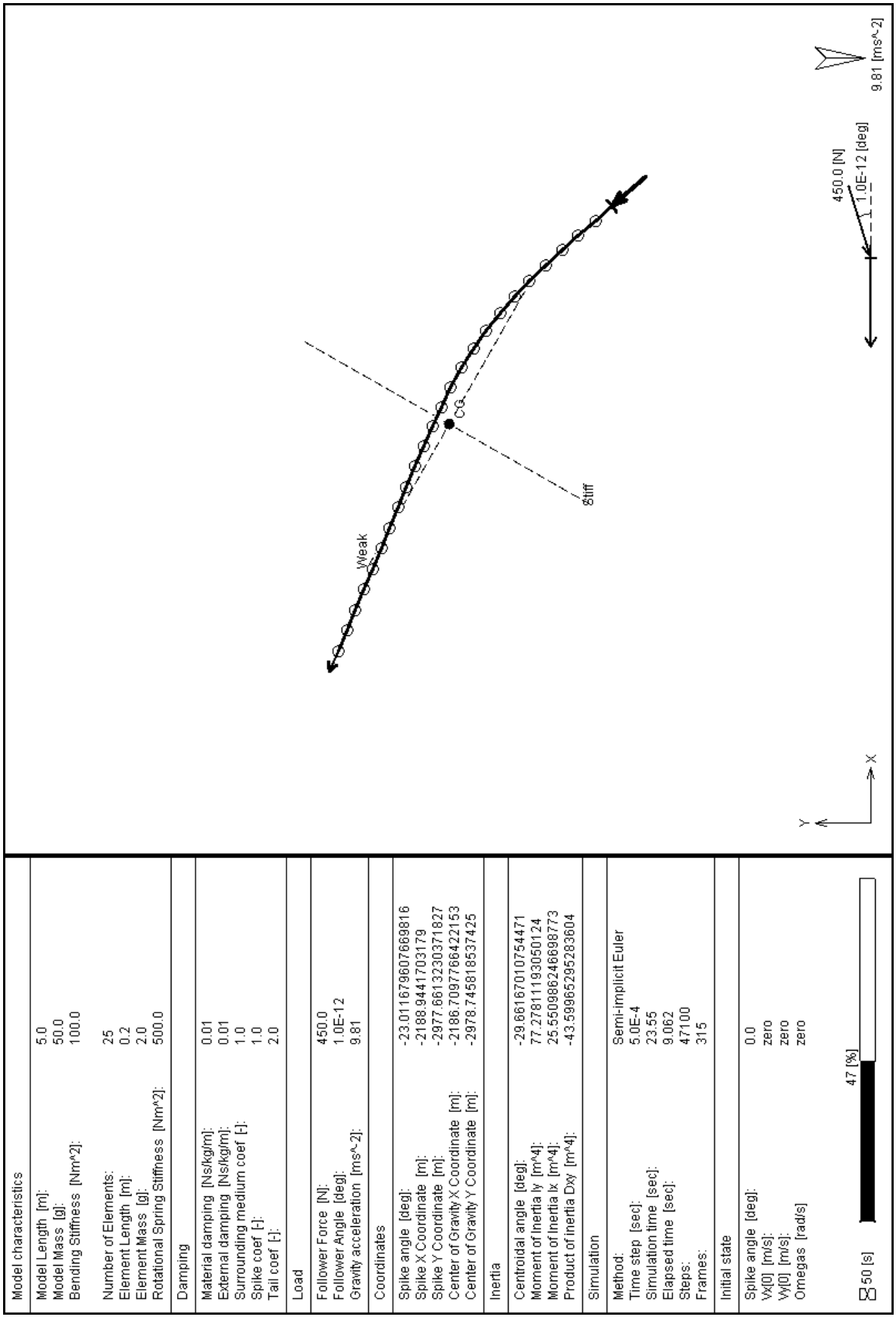
Textově jsou zobrazovány:

- fyzikální vlastnosti modelu - délka, hmotnost, ohybová tuhost a odpovídající parametry tuhých dílců a rotačních pružin,
- parametry tlumení a působícího zatížení
- souřadnice těžiště a směry hlavních os setrvačnosti,
- momenty setrvačnosti a deviační moment,
- nastavení simulace - metoda řešení, časový krok, čas simulace, výpočetní doba a počet kroků.

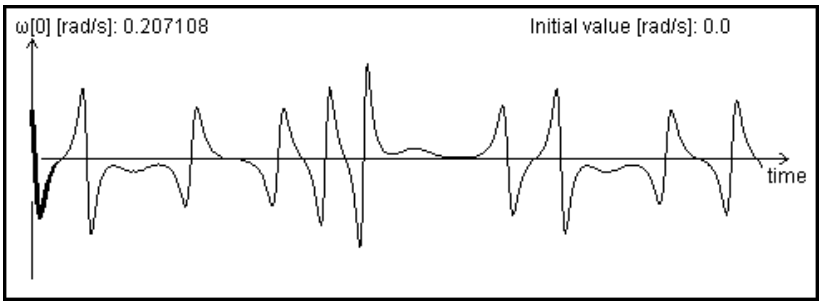
Další možnosti zobrazení

V průběhu výpočtu lze v reálném čase sledovat libovolný počet libovolných stavových proměnných těmito způsoby:

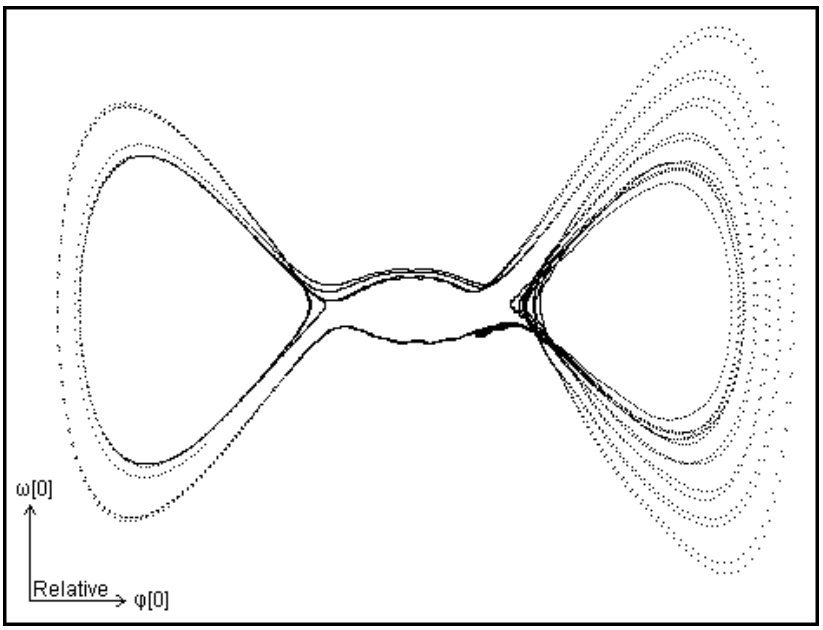
- **v závislosti na čase**, viz obrázek 4.4,
- **ve vzájemné závislosti** na další zvolené proměnné, viz obrázek 4.5,
- **zobrazení proměnných stejného typu**, například úhlových rychlostí ω, viz obrázek 4.6.



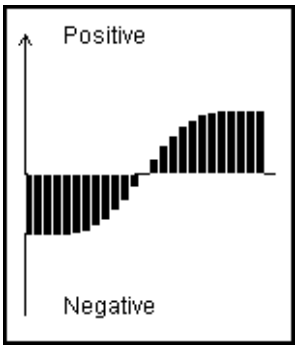
Obrázek 4.3: Základní grafické prostředí modelu.



Obrázek 4.4: Zobrazení stavové proměnné v závislosti na čase.



Obrázek 4.5: Zobrazení stavových proměnných ve vzájemné závislosti.



Obrázek 4.6: Zobrazení proměnných stejného typu.

Kapitola 5

Závěr

V teoretické části práce je čtenáři nabídnut kompaktní teoretický úvod. Jsou popsány základní zákony klasické mechaniky a dynamiky a je osvětlen energetický - Lagrangeovský - přístup k popisu dynamického systému. V návaznosti Lagrangeovské odvození pohybových rovnic systému jsou popsány numerické metody jejich řešení. V úvodu práce byly dále zmíněny zdroje nelinearit v mechanickém systému a vlastnosti konzervativních a nekonzervativních sil. Jako poslední je popsán princip metody tuhých dílců, která je použita pro vytvoření numerického modelu.

V praktické části práce je prezentován postup odvození nelineárního modelu volného prutu za použití metody tuhých dílců. Model je dále doplněn o nezbytné součásti jako je zatížení sledující silou, vliv gravitačního pole a materiálové tlumení.

Zvláštní pozornost byla věnována podrobnému vystižení tlumení vlivem okolního prostředí. S ohledem na reálnou předlohu úlohy byly zavedeny vlivy rozdílného směru natočení a pohybu dílců, a nakonec i možnost stabilizace rakety pomocí křidélek. Tlumení vlivem okolního prostředí má významný vliv na chování modelu na hranici ztráty stability a následné pokritické chování.

Vzhledem k výhodné formulaci modelu volného prutu byla provedena úprava pohybových rovnic a byl vytvořen samostatný model konzoly.

V další části práce jsou prezentovány výsledky numerických simulací modelů:

- analýza vlastních frekvencí volného prutu a konzoly,
- průhyb a pootočení konzoly zatížené vlastní tíhou,
- hodnota kritické sledující síly volného prutu a konzoly.

Při simulacích se potvrdil předpoklad nízké časové náročnosti výpočtu. I simulace s velmi podrobnými modely lze provádět mnohem rychleji než v reálném čase.

Byla provedena vlastní implementace numerického modelu v programovacím jazyce Java a zároveň bylo vytvořeno grafické prostředí umožňující podrobně sledovat průběh simulace.

5.1 Výhledy

S přihlédnutím k rozsáhlým možnostem, které nabízí modely a jejich implementace, je plánováno budoucí použití vytvořených modelů pro:

- kompletní ověření vlastních frekvencí volného prutu díky lokální souřadné soustavě tvořené hlavními osami setrvačnosti,
- ověření vlastních tvarů volného prutu a konzoly,
- podrobné studium pokritického chování konstrukcí,
- studium vlivu tlumení vlivem vnějšího prostředí na pokritické chování konstrukcí.

Vzhledem k trvajícímu zájmu autora o programování se dále nabízí možnost paralelizace úlohy použitím programovacího jazyka Java nebo nVidia CUDA.

Literatura

[1] BECK, M. Die Knicklast des einseitig eingespannten, tangential gedrűckten Stabes, Zeitschrift Angewandte Mathematik und Physik, Volume 3, Issue 3, 1952, p. 225-228.

[2] FRANTÍK, P. Nelineární projevy mechanických konstrukcí, dizertační práce, Ústav stavební mechaniky FAST VUT v Brně, Brno 2004, 117 s.

[3] FRANTÍK, P. Post-critical behaviour of beam loaded by follower force. CD proceedings of national conference Engineering Mechanics 2009, Svratka, Česká republika, 2009, 9 s. ISBN 80-86246-35-2

[4] FRANTÍK, P., MACUR, J. Kritická síla imperfektovaných systémů. Modelování a měření nelineárních jevů v mechanice, sborník příspěvků, Nečtiny, Česká republika, 2006, str. 67-70, ISBN 80-02-01827-3

[5] FRANTÍK, P. Implementace Rychlé Fourierovy transformace, Java balík. Dostupné z: www.kitnarf.cz/java/.

[6] FRANTÍK, P. Aplikace FyDik. Dostupné z: http://fydik.kitnarf.cz/

[7] GOLDSTEIN, H. Classical mechanics, 3 edition. Addison-Wesley, 2001, 680 s. ISBN 978-020-1657-029.

[8] HENRYCH, J. Úplná soustava finitních metod mechaniky a možnosti dalšího rozvoje, studie 6.85 ČSAV, Praha 1985, 165 s.

[9] GURAN, A., OSSIA, K. On the stability of a flexible missile under an end thrust. Math. Comput. Modelling, Vol. 14, 1990, 5 s.

[10] HERMANN, G. Dynamics and stability of mechanical systems with follower forces. Stanford University, Stanford, California, 1971, 234 s.

[11] KADLČÁK, J., KYTÝR J. Statika stavebních konstrukcí: základy stavební mechaniky, staticky určité prutové konstrukce. 3. vyd. Brno: VUTIUM, 2010, 349 s. ISBN 978-80-214-3419-6.

[12] KADLČÁK, J., KYTÝR J. Statika stavebních konstrukcí. Třetí dostisk druhého vyd. V Brně: VUTIUM, 2009, 431 s. ISBN 978-80-214-3428-8.

[13] LEECH, J. W. Klasická mechanika (orig. Butler & Tanner Ltd., Frome, 1958), SNTL - nakladatelství tech. lit., Praha 1970, 133 s.

[14] MACUR, J. Úvod do teorie dynamických systémů a jejich simulace. 1. vyd. Brno: VUT, 1995, 87 s. ISBN 80-214-0698-4.

[15] MACUR, J. Dynamické systémy. [online]. [cit. 2014-05-03]. Dostupné z: http://www.fce.vutbr.cz/studium/materialy/Dynsys/

[16] MACUR, J. Simulace dynamických systémů v jazyce Java. [online]. [cit. 2014-05-03]. Dostupné z: http://www.fce.vutbr.cz/studium/materialy/simul/

[17] MELCER, J. Kuchárová, D.: *Dynamika stavebných konštrukcií*, Žilinská univerzita, Žilina 2004, 240 s.

[18] NEWTON, Isaac. The principia: mathematical principles of natural philosophy. S.l.: Snowball Pub, New York 2010. 991 s.ISBN 16-079-6240-3.

[19] TAYLOR, John R. Classical mechanics. Sausalito, Calif.: University Science Books, c2005, xiv, 786 s. ISBN 18-913-8922-X.

[20] TORBY, B. *Energy Methods. Advanced Dynamics for Engineers*, CBS College Publishing, New York 1984, 426 s.

[21] http://www.oracle.com/technetwork/java/

[22] https://www.eclipse.org/downloads/

Seznam symbolů

x, y, z	souřadné osy pravoúhlé souřadné soustavy
$\mathbf{r}$	polohový vektor bodu
$\hat{\mathbf{i}}, \hat{\mathbf{j}}, \hat{\mathbf{k}}$	jednotkové vektory
$\mathbf{v}$	vektor rychlosti bodu
t	čas
v_x, v_y, v_z	složky vektoru rychlosti
O	počátek souřadné soustavy
φ	úhel od horizontály
ω	úhlová rychlost
x_P, y_P	souřadnice bodu P v pravoúhlé souřadné soustavě
$\mathbf{F}_{ext}$	vnější síla působící na hmotný bod
$\mathbf{a}$	zrychlení
m	hmotnost hmotného bodu
$\mathbf{p}$	hybnost hmotného bodu
$\mathbf{P}$	celková hybnost soustavy
$\mathbf{v}_{ex}$	rychlost tryskání spalin
E_p, T	potenciální energie
E_k, V	kinetická energie
k	tuhost rotační nebo translační pružiny
F_{ts}	síla působící na translační pružinu
M_{rs}	moment působící na rotační pružinu
$E_{p,ts}$	potenciální energie akumulovaná v translační pružině
$E_{p,rs}$	potenciální energie akumulovaná v rotační pružině
g	gravitační zrychlení
l	délka dílce, kyvadla
q_i	zobecněná souřadnice systému
$\dot{q}_i$	zobecněná rychlost systému
$\mathcal{L}$	Lagrangeova funkce
δr	virtuální posunutí
δW	virtuální práce
δs	virtuální posun

$\delta\varphi$	virtuální pootočení
$\mathbf{R}$	výslednice reakčních sil
$\mathbf{Z}$	setrvačná síla
$\mathcal{S}$	akce systému
q_{real}	uskutečněná trajektorie
q_{virt}	neuskutečněná trajektorie
h	časový krok
k_1, k_2, k_3, k_4	koeficienty R-K metody
E	modul pružnosti
ε	poměrné převoření
σ	napětí
W	práce
∇	gradient
Φ	potenciál síly
L	délka prutu
M	hmotnost prutu
I	moment setrvačnosti
n	počet dílců
$M_{s,s+1}$	napjatost pružiny $s, s+1$
$F_{follower}$	sledující síla
c	koeficient tlumení
c_{int}	koeficient materiálového tlumení
c_{ext}	koeficient tlumení vlivem vnějšího prostředí
a	koeficient zohledňující rozdílné natočení dílců a směru jejich pohybu
$\mathbf{D}$	tlumící síla
$\mathbf{F}_g$	gravitační síla
a_g	zrychlení působící na dílec
f	frekvence
δ	průhyb volného konce konzoly
Θ	pootočení volného konce konzoly
F_{cr}	kritická síla

Matice a vektory

$\mathbf{M}$	matice hmotných momentů setrvačnosti
$\mathbf{Q}$	matice transformačních momentů setrvačnosti
$\mathbf{K}$	matice tuhostí rotačních pružin
$\mathbf{D}_i$	matice materiálového útlumu
$\mathbf{F}$	vektor zatížení sledující silou
$\mathbf{v}$	vektor translačních rychlostí
ω	vektor úhlových rychlostí
$\mathbf{D}_e$	vektor tlumení vlivem vnějšího prostředí
$\mathbf{A}$	vektor vlivu gravitačního pole

Seznam příloh

Příloha A: Implementace modelu v jazyce Java
Příloha B: Kompaktní disk

Přílohy

Příloha A: Implementace modelu v jazyce Java

```
1  package cz.dl.freerodmodel;
2
3  import java.io.FileNotFoundException;
4  import java.io.ObjectInputStream.GetField;
5  import java.io.PrintWriter;
6
7  import cz.kitnarf.fft.AmplitudeAnalyser;
8  import cz.kitnarf.fft.FFT;
9  import cz.kitnarf.fft.Peak;
10 import cz.kitnarf.matrix.*;
11
12
13 public class FreeRodModel {
14
15     /* Variables */
16     private int elementCount;
17     //
18     private double mass;
19     private double length;
20     private double stiffness;
21     private double innerDampingCoef;
22     private double exterDampingCoef;
23     private double spikeDampingCoef;
24     private double tailDampingCoef;
25     private double mediumDampingCoef;
26     //
27     private double elementMass    =mass/elementCount;
28     private double elementLength  =length/elementCount;
29     private double elementStiffness =stiffness/elementLength;
30     //
31     private double step;
32     private double time;
33     //
34     private double kineticEnergy;
35     private double potentialEnergy;
36     private double overallEnergy;
37     //
38     private double followerForce;
39     private double followerAngle;
40     //
41     private double gravityAcceleration;
42
43     /* Solution */
44     private Vector solution;
```

```
45      //RK
46      private Vector RKsolutionK1;
47      private Vector RKsolutionK2;
48      private Vector RKsolutionK3;
49      private Vector RKsolutionK4;
50
51      /* Arrays */
52      double[] angles     =new double [elementCount];
53      double[] anglesSin    =new double [elementCount];
54      double[] anglesCos    =new double [elementCount];
55
56      double[] omegas     =new double [elementCount];
57      double[] coordinates  =new double [2*elementCount+2];
58      double[] velocities   =new double [2*elementCount+2];
59      double[] centerOfGravity=new double [2];
60      double[] dampingForcesX =new double [elementCount+1];
61      double[] dampingForcesY =new double [elementCount+1];
62      double[] aeroDampingCoef=new double [elementCount+1];
63      double[] velocityAngles=new double [elementCount+1];
64
65      /* RK arrays */
66      double[] omegasT    =new double [elementCount];
67      double[] anglesT    =new double [elementCount];
68      double[] coordinatesT =new double [2*elementCount+2];
69      double[] velocitiesT  =new double [2*elementCount+2];
70
71      /* Matrices */
72      int rows=elementCount+2;
73      int columns=rows;
74      double[][] matrixM    =new double [rows][columns];
75      double[][] matrixQ    =new double [rows][elementCount];
76      double[][] matrixK    =new double [rows][elementCount];
77      double[][] matrixC    =new double [rows][elementCount];
78
79      /**/
80      Matrix leftSide=new Matrix();
81      Vector rightSide=new Vector();
82
83
84
85      /** Initial state */
86      public void setInitialState (double vx0Init, double vy0Init, double[]
            omegasInit, double[]anglesInit,
87                  double massInit, double lengthInit, double stiffnessInit,
                        double innerDampingCoefInit,
88                  double followerForceInit, double gravityAccelerationInit,
                        double followerAngleInit, double exterDampingCoefInit,
89                  double spikeDampingCoefInit, double tailDampingCoefInit,
                        double mediumDampingCoefInit)
90      {
91        velocities[0]=vx0Init;
92        velocities[1]=vy0Init;
93        //
94        for (int i=0; i<omegas.length; i++) omegas[i]=omegasInit[i];
95        //
96        for (int i=0; i<angles.length; i++) angles[i]=anglesInit[i];
```

```
97        for(int i=0; i<anglesSin.length; i++) anglesSin[i]=Math.sin(angles[i]);
98        for(int i=0; i<anglesCos.length; i++) anglesCos[i]=Math.cos(angles[i]);
99        //
100       mass=massInit;
101       length=lengthInit;
102       stiffness=stiffnessInit;
103       innerDampingCoef=innerDampingCoefInit;
104       followerForce=followerForceInit;
105       followerAngle=followerAngleInit;
106       gravityAcceleration=gravityAccelerationInit;
107       exterDampingCoef=exterDampingCoefInit;
108       spikeDampingCoef=spikeDampingCoefInit;
109       tailDampingCoef=tailDampingCoefInit;
110       mediumDampingCoef=mediumDampingCoefInit;
111     }
112
113     /** Constructor */
114     public FreeRodModel (int elementCountInit, double stepInit)
115     {
116       step=stepInit;
117       elementCount=elementCountInit;
118     }
119
120     /** Calculates current model time */
121     private void updateTime ()
122     {
123       time+=step;
124     }
125
126     /** Calculates current angular velocities */
127     private void updateOmegas ()
128     {
129       for(int i=0; i<omegas.length; i++)
130       {
131         omegas[i]+=step*solution.getValue(i+2);
132       }
133     }
134
135     /** Calculates current angles */
136     private void updateAngles ()
137     {
138       for(int i=0; i<angles.length; i++)
139       {
140         angles[i]+=step*omegas[i];
141       }
142
143       for(int i=0; i<anglesSin.length; i++) anglesSin[i]=Math.sin(angles[i]);
144       for(int i=0; i<anglesCos.length; i++) anglesCos[i]=Math.cos(angles[i]);
145
146     }
147
148     /** Calculates current coordinates of mass points */
149     private void updateCoords()
150     {
151       for(int i=0; i<coordinates.length; i++)
152       {
```

```
153         if(i<2) coordinates[i]+=step*velocities[i];
154         /* x */ else if (i%2==0) coordinates[i]=coordinates[i-2]+
                elementLength*anglesCos[(i-i/2-1)];
155         /* y */ else if (i%2!=0) coordinates[i]=coordinates[i-2]+
                elementLength*anglesSin[(i-i/2-2)];
156       }
157     }
158
159     /** Calculates current velocities */
160     private void updateVelocities ()
161     {
162       /* velocities[0], velocities[1]*/
163       for(int i=0; i<2; i++) velocities[i]+=step*solution.getValue(i);
164       /* other velocities */
165       for(int i=2; i<velocities.length; i++)
166       {
167       /* x */ if (i%2==0) velocities[i]=velocities[i-2]-omegas[(i-i/2-1)]*
              elementLength*anglesSin[(i-i/2-1)]; /* a%b == 0   A divisible by B
              if zero .  */
168       /* y */ else if (i%2!=0) velocities[i]=velocities[i-2]+omegas[(i-i/2-2)
              ]*elementLength*anglesCos[(i-i/2-2)];
169       }
170     }
171
172     /** Calculates current center of gravity coordinates */
173     private void updateCG ()
174     {
175       /* Center of gravity coordinates */
176       /* x */
177       double topX=coordinates[0]+coordinates[coordinates.length-2];
178       for(int i=2; i<coordinates.length-2; i+=2) topX+=2*coordinates[i];
179       double bottomX=2*elementCount;
180       centerOfGravity[0]=topX/bottomX;
181       /* y */
182       double topY=coordinates[1]+coordinates[coordinates.length-1];
183       for(int i=3; i<coordinates.length-1; i+=2) topY+=2*coordinates[i];
184       double bottomY=2*elementCount;
185       centerOfGravity[1]=topY/bottomY;
186     }
187
188     /** Calculates current kinetic energy */
189     private void updateKineticEnergy ()
190     {
191       kineticEnergy=0;
192       kineticEnergy+=elementCount*(velocities[0]*velocities[0]+velocities[1]*
              velocities[1]);
193       for (int i=0; i<elementCount; i++)    kineticEnergy+=omegas[i]*omegas[i
              ]*elementLength*elementLength*(1/3d+(elementCount-i-1));
194       for (int i=0; i<elementCount; i++)    kineticEnergy-=velocities[0]*
              omegas[i]*elementLength*anglesSin[i]*(1+2*(elementCount-i-1));
195       for (int i=0; i<elementCount; i++)    kineticEnergy+=velocities[1]*
              omegas[i]*elementLength*anglesCos[i]*(1+2*(elementCount-i-1));
196       for (int i=0; i<elementCount; i++)
197       {
198         for (int l=0; l<=i; l++)
199         {
```

```
200           for (int k=0; k<=l; k++)
201           {
202             if (k<l)
203             {
204               if(l==i)    kineticEnergy+=elementLength*elementLength*omegas[k
                      ]*omegas[l]*Math.cos(angles[l]-angles[k]);
205               else      kineticEnergy+=2*elementLength*elementLength*omegas[k
                      ]*omegas[l]*Math.cos(angles[l]-angles[k]);
206             }
207           }
208         }
209       }
210       kineticEnergy*=0.5*elementMass;
211
212     }
213
214     /** Calculates current potential energy */
215     private void updatePotentinalEnergy()
216     {
217       potentialEnergy=0;
218
219       for (int i=0; i<elementCount-1; i++)
220       {
221         potentialEnergy+=(angles[i+1]-angles[i])*(angles[i+1]-angles[i]);
222       }
223
224       potentialEnergy*=0.5*elementStiffness;
225     }
226
227     /** Calculates current overall energy */
228     private void updateOverallEnergy()
229     {
230       overallEnergy=0;
231       overallEnergy=kineticEnergy+potentialEnergy;
232     }
233
234
235     /** Fills the M matrix */
236     private void assembleMatrixM ()
237     {
238
239
240       for(int r=0; r<rows; r++)
241       {
242         for(int c=0; c<columns; c++)
243         {
244           /* filling the first symmetric half */
245           /* velocities[0] row */
246           if(r==0)
247           {
248             if(c==0)         matrixM[0][0] =2d*elementCount;
249             if(c==1)         matrixM[0][1] =0d;
250             if(c>=2)         matrixM[0][c] =-elementLength*anglesSin[c
                    -2]*(1+2*(elementCount-(c-1)));
251             if(c==columns-1)     matrixM[0][c] =-elementLength*anglesSin[c-2];
252           }
```

```
253
254           /* velocities[1] row */
255           if(r==1)
256           {
257             if(c==0)         matrixM[1][0] =0d;
258             if(c==1)         matrixM[1][1] =2d*elementCount;
259             if(c>=2)         matrixM[1][c] =elementLength*anglesCos[c
                  -2]*(1+2*(elementCount-(c-1)));
260             if(c==columns-1)    matrixM[1][c] =elementLength*anglesCos[c-2];
261           }
262           /* generic omega rows */
263           if(r>=2 && r<rows-1)
264           {
265             if (r==c)          matrixM[r][r] =2d*elementLength*elementLength
                  *(1/3d+(elementCount+(r-(2*r-1))));
266             if (c>r && c<columns-1)   matrixM[r][c] =elementLength*
                  elementLength*Math.cos(angles[c-2]-angles[r-2])*(1+2*(
                  elementCount-(c-1)));
267             if (c>r && c==columns-1)  matrixM[r][c] =elementLength*
                  elementLength*Math.cos(angles[c-2]-angles[r-2]);
268           }
269           /* last omega row */
270           if(r==rows-1 && c==r)     matrixM[r][c] =2/3d*elementLength*
                elementLength;
271
272         }
273
274       }
275       ////////////////////////////////////////
276       /* generating the other symmetric half */
277       for(int r=0; r<rows; r++)
278       {
279         for(int c=0; c<columns; c++)
280         {
281           if(r!=c) matrixM[c][r]=matrixM[r][c];
282         }
283
284       }
285
286     }
287
288     /** Fills the Q matrix */
289     private void assembleMatrixQ ()
290     {
291       for (int r=0; r<rows; r++)
292       {
293         for (int c=0; c<rows-2; c++)
294         {
295           if (r==0)         matrixQ[r][c]=elementLength*anglesCos[c]*(1+2*(
                elementCount-(c+1)));
296           if (r==1)         matrixQ[r][c]=elementLength*anglesSin[c]*(1+2*(
                elementCount-(c+1)));
297
298           if (r>=2 && r<rows-1 )
299           {
```

```
300             if(c>=r-2)        matrixQ[r][c]=elementLength*elementLength*Math.
                    sin(angles[c]-angles[r-2])*(1+2*(elementCount-(c+1)));
301           }
302         }
303       }
304
305       /* generating the other antisymmetric half */
306       for(int r=2; r<rows; r++)
307       {
308         for(int c=0; c<rows-2; c++)
309         {
310           if(r-2!=c) matrixQ[c+2][r-2]=-matrixQ[r][c];
311         }
312       }
313     }
314
315     /** Fills the K matrix */
316     private void assembleMatrixK ()
317     {
318       for (int r=2; r<rows; r++)
319       {
320         for (int c=0; c<elementCount; c++)
321         {
322           if (r==2)
323           {
324             if (c==0) matrixK[r][c]=1;
325             if (c==1) matrixK[r][c]=-1;
326           }
327           if (r>2 && r<rows-1)
328           {
329             if (c==r-3) matrixK[r][c]=-1;
330             if (c==r-2) matrixK[r][c]=2;
331             if (c==r-1) matrixK[r][c]=-1;
332           }
333           if (r==rows-1)
334           {
335             if (c==r-3) matrixK[r][c]=-1;
336             if (c==r-2) matrixK[r][c]=1;
337           }
338
339         }
340       }
341     }
342
343     /** Fills the C matrix */
344     private void assembleMatrixC ()
345     {
346       for (int r=2; r<rows; r++)
347       {
348         for (int c=0; c<elementCount; c++)
349         {
350           if (r==2)
351           {
352             if (c==0) matrixC[r][c]=1;
353             if (c==1) matrixC[r][c]=-1;
354           }
```

```
355           if (r>2 && r<rows-1)
356           {
357             if (c==r-3) matrixC[r][c]=-1;
358             if (c==r-2) matrixC[r][c]=2;
359             if (c==r-1) matrixC[r][c]=-1;
360           }
361           if (r==rows-1)
362           {
363             if (c==r-3) matrixC[r][c]=-1;
364             if (c==r-2) matrixC[r][c]=1;
365           }
366
367         }
368       }
369     }
370
371
372     /** Solves the equation, gets the solution vector */
373     private void createEquation ()
374     {
375       /* Creating the matrices and vectors for solving */
376       /* Matrix M*/
377       Matrix kitMatrixM=new Matrix();
378       kitMatrixM.allocateFor(matrixM);
379       kitMatrixM.setValues(matrixM);
380       kitMatrixM.multiply(0.5*elementMass);
381       //
382       leftSide=kitMatrixM;
383       //System.out.println(kitMatrixM);
384
385       /* Matrix Q */
386       Matrix kitMatrixQ=new Matrix();
387       kitMatrixQ.allocateFor(matrixQ);
388       kitMatrixQ.setValues(matrixQ);
389       kitMatrixQ.multiply(0.5*elementMass);
390       //System.out.println(kitMatrixQ);
391
392       /* Matrix K */
393       Matrix kitMatrixK=new Matrix();
394       kitMatrixK.allocateFor(matrixK);
395       kitMatrixK.setValues(matrixK);
396       kitMatrixK.multiply(elementStiffness);
397       //System.out.println(kitMatrixK);
398
399       /* Matrix C */
400       Matrix kitMatrixC=new Matrix();
401       kitMatrixC.allocateFor(matrixC);
402       kitMatrixC.setValues(matrixC);
403       kitMatrixC.multiply(innerDampingCoef*elementMass);
404       //System.out.println(kitMatrixC);
405
406       /* Vector omega^2 */
407       Vector kitOmegaSq=new Vector();
408       kitOmegaSq.allocate(elementCount, 1);
409       for (int r=0; r<elementCount; r++)  kitOmegaSq.addValue(r, omegas[r]*
            omegas[r]);
```

```
410       //System.out.println(kitOmegaSq);
411
412       /* Vector of angles */
413       Vector kitFi=new Vector();
414       kitFi.allocate(elementCount, 1);
415       for (int r=0; r<elementCount; r++)  kitFi.addValue(r, angles[r]);
416       //System.out.println(kitFi);
417
418       /* Vector of angular velocities */
419       Vector kitOmega=new Vector();
420       kitOmega.allocate(elementCount, 1);
421       for (int r=0; r<elementCount; r++)  kitOmega.addValue(r, omegas[r]);
422       //System.out.println(kitOmega);
423
424       double followerForceX=followerForce*Math.cos(angles[elementCount-1]+
              followerAngle);
425       double followerForceY=followerForce*Math.sin(angles[elementCount-1]+
              followerAngle);
426
427
428       /* Vector Fx*l*sin(fi) */
429       Vector kitFollowerX=new Vector();
430       kitFollowerX.allocate(elementCount+2, 1);
431       for (int r=0; r<elementCount+2; r++)
432       {
433         if (r==0) kitFollowerX.addValue(r, -followerForceX);
434         if (r>1)  kitFollowerX.addValue(r, followerForceX*elementLength*
                anglesSin[r-2]);
435       }
436
437
438       /* Vector -Fy*l*cos(fi) */
439       Vector kitFollowerY=new Vector();
440       kitFollowerY.allocate(elementCount+2, 1);
441       for (int r=0; r<elementCount+2; r++)
442       {
443         if (r==1) kitFollowerY.addValue(r, -followerForceY);
444         if (r>1)  kitFollowerY.addValue(r, -followerForceY*elementLength*
                anglesCos[r-2]);
445       }
446
447       /* Vector of moments of gravity forces X */
448       Vector kitGravityXmoment=new Vector();
449       kitGravityXmoment.allocate(elementCount+2, 1);
450       for (int r=0; r<elementCount+2; r++)
451       {
452         if (r>1)
453         {
454           double value=0;
455           kitGravityXmoment.addValue(r, value);
456         }
457
458       }
459
460
461       /* Vector of moments of gravity forces Y */
```

```
462       Vector kitGravityYmoment=new Vector();
463       kitGravityYmoment.allocate(elementCount+2, 1);
464       for (int r=0; r<elementCount+2; r++)
465       {
466         if (r>1)
467         {
468
469           kitGravityYmoment.addValue(r, -elementMass*gravityAcceleration*
                elementLength*anglesCos[r-2]*(elementCount-0.5-(r-2)));
470         }
471
472       }
473       //System.out.println(kitGravityYmoment);
474
475
476       /* Vector of translational gravity forces */
477       Vector kitGravityTrans=new Vector();
478       kitGravityTrans.allocate(elementCount+2, 1);
479       for (int r=0; r<elementCount+2; r++)
480       {
481         if (r==0) kitGravityTrans.addValue(r, 0);
482         if (r==1) kitGravityTrans.addValue(r, -mass*gravityAcceleration);
483       }
484
485       /* Vector of overall gravity effect */
486       Vector kitGravityOverall=new Vector();
487       kitGravityOverall.allocate(elementCount+2, 1);
488       for (int r=0; r<elementCount+2; r++)  kitGravityOverall.addValue(r,
            kitGravityXmoment.getValue(r)+kitGravityYmoment.getValue(r)+
            kitGravityTrans.getValue(r));
489       //System.out.println(kitGravityOverall);
490
491       /* Aero damping coefficients */
492       for (int i=0; i<=elementCount; i++)
493       {
494         if (velocities[2*i]>0 && velocities[2*i+1]>0 ) velocityAngles[i]=Math
              .atan(velocities[2*i+1]/velocities[2*i]);
495         if (velocities[2*i]<0 && velocities[2*i+1]>0 ) velocityAngles[i]=Math
              .PI-Math.atan(-velocities[2*i+1]/velocities[2*i]);
496         if (velocities[2*i]<0 && velocities[2*i+1]<0 ) velocityAngles[i]=-
              Math.PI+Math.atan(-velocities[2*i+1]/-velocities[2*i]);
497         if (velocities[2*i]>0 && velocities[2*i+1]<0 ) velocityAngles[i]=-
              Math.atan(velocities[2*i+1]/-velocities[2*i]);
498
499       if (velocities[2*i]==0 && velocities[2*i+1]>0 )velocityAngles[i]=0.5*
            Math.PI;
500         if (velocities[2*i]==0 && velocities[2*i+1]<0 )velocityAngles[i
              ]=-0.5*Math.PI;
501         if (velocities[2*i]>0 && velocities[2*i+1]==0 )velocityAngles[i]=0;
502         if (velocities[2*i]<0 && velocities[2*i+1]==0 )velocityAngles[i]=Math
              .PI;
503
504         if(i==0)
505           aeroDampingCoef[i]=Math.abs(Math.sin(Math.abs(velocityAngles[i]-
                angles[i])));
506
```

```
507         if(i==elementCount)
508           aeroDampingCoef[i]=Math.abs(Math.sin(Math.abs(velocityAngles[i]-
                angles[i-1])));
509
510         if(i!=0 && i!=elementCount)
511           aeroDampingCoef[i]=Math.abs(Math.sin(Math.abs(velocityAngles[i]-(
                angles[i-1]+angles[i])/2d)));
512
513       /* The damping forces */
514       for (int s=0; s<=elementCount; s++)
515       {
516
517         if (s==0)           dampingForcesX[s]=-exterDampingCoef*(
              spikeDampingCoef*mediumDampingCoef*aeroDampingCoef[s])*0.5*
              elementMass*velocities[s*2];
518         if (s>=1 && s<elementCount) dampingForcesX[s]=-exterDampingCoef*(
              mediumDampingCoef*aeroDampingCoef[s])*elementMass*velocities[s
              *2];
519         if (s==elementCount)    dampingForcesX[s]=-exterDampingCoef*(
              tailDampingCoef*mediumDampingCoef*aeroDampingCoef[s])*0.5*
              elementMass*velocities[s*2];
520
521       }
522
523       for (int s=0; s<=elementCount; s++)
524       {
525         if (s==0)           dampingForcesY[s]=-exterDampingCoef*(
              spikeDampingCoef*mediumDampingCoef*aeroDampingCoef[s])*0.5*
              elementMass*velocities[s*2+1];
526         if (s>=1 && s<elementCount) dampingForcesY[s]=-exterDampingCoef*(
              mediumDampingCoef*aeroDampingCoef[s])*elementMass*velocities[s
              *2+1];
527         if (s==elementCount)    dampingForcesY[s]=-exterDampingCoef*(
              tailDampingCoef*mediumDampingCoef*aeroDampingCoef[s])*0.5*
              elementMass*velocities[s*2+1];
528
529       }
530
531       /* Vector of external translational damping forces */
532       Vector kitExtDampingTrans=new Vector();
533       kitExtDampingTrans.allocate(elementCount+2, 1);
534       for (int r=0; r<elementCount+2; r++)
535       {
536         if (r==0)
537         {
538           double value=0;
539           for (int i=0; i<=elementCount; i++) value+=dampingForcesX[i];
540           kitExtDampingTrans.addValue(r, value);
541         }
542
543         if (r==1)
544         {
545           double value=0;
546           for (int i=0; i<=elementCount; i++) value+=dampingForcesY[i];
547           kitExtDampingTrans.addValue(r, value);
548         }
```

```
549       }
550
551
552       /* Vector of moments of external damping forces X */
553       Vector kitExtDampingXmoment=new Vector();
554       kitExtDampingXmoment.allocate(elementCount+2, 1);
555       for (int r=0; r<elementCount+2; r++)
556       {
557         if (r>1)
558         {
559           double value=0;
560           for (int i=r-1; i<=elementCount; i++) value+=dampingForcesX[i];
561           kitExtDampingXmoment.addValue(r, elementLength*anglesSin[r-2]*value
                  );
562         }
563       }
564
565
566       /* Vector of moments of external damping forces Y */
567       Vector kitExtDampingYmoment=new Vector();
568       kitExtDampingYmoment.allocate(elementCount+2, 1);
569       for (int r=0; r<elementCount+2; r++)
570       {
571         if (r>1)
572         {
573           double value=0;
574           for (int i=r-1; i<=elementCount; i++) value+=dampingForcesY[i];
575           kitExtDampingYmoment.addValue(r, elementLength*anglesCos[r-2]*value
                  );
576         }
577       }
578
579
580       /* Vector of overall external damping effect */
581       Vector kitExtDampingOverall=new Vector();
582       kitExtDampingOverall.allocate(elementCount+2, 1);
583       for (int r=0; r<elementCount+2; r++)  kitExtDampingOverall.addValue(r,
              -kitExtDampingXmoment.getValue(r)+kitExtDampingYmoment.getValue(r)+
              kitExtDampingTrans.getValue(r));
584
585
586
587
588       //************************//
589       /* Solving the equation */
590       /* Multiplying matrices and vectors */
591       Vector QtimesOmegaSq=Vector.multiply(kitMatrixQ, kitOmegaSq);
592       Vector KtimesFi=Vector.multiply(kitMatrixK, kitFi);
593       Vector CtimesOmega=Vector.multiply(kitMatrixC, kitOmega);
594
595       /* Summation of right side */
596       Vector rightSide1=new Vector();
597       rightSide1.allocate(elementCount+2, 1);
598       for (int r=0; r<rightSide1.getRowCount(); r++)
599       {
600         //if (r<3) rightSide.setValue (r, 0);
```

```
601         rightSide1.setValue(r, QtimesOmegaSq.getValue(r)-KtimesFi.getValue(r)
                -CtimesOmega.getValue(r)+kitFollowerX.getValue(r)+kitFollowerY.
                getValue(r)+kitGravityOverall.getValue(r)+kitExtDampingOverall.
                getValue(r));
602
603       }
604       rightSide=rightSide1;
605
606     }
607     private void getSolution(Matrix leftSide, Vector rightSide){
608       /* Solution vector */
609       GaussElimination solver=new GaussElimination();
610       solution=solver.solve(leftSide, rightSide);
611
612       //System.out.println(solution);
613
614     }
615
616
617     public double getCoordinate(int index) {
618       return coordinates[index];
619     }
620
621     public double getVelocity(int index) {
622       return velocities[index];
623     }
624
625     public double[] getCoordinates(){
626       return coordinates;
627
628     }
629
630     public double[] getAngles(){
631       return angles;
632     }
633
634
635     public double[] getCenterOfGravityCoordinates(){
636       return centerOfGravity;
637     }
638
639     public double[] getOmegas (){
640       return omegas;
641     }
642
643     public double getPotentialEnergy(){
644       return potentialEnergy;
645     }
646
647     public double [] getAeroCoefficients(){
648     return aeroDampingCoef;
649     }
650
651     public double [] getVelocityAngles(){
652       return velocityAngles;
653       }
```

```
654
655     public double [] getExternalDampingX(){
656
657       return dampingForcesX;
658     }
659
660     public double [] getExternalDampingY(){
661
662       return dampingForcesY;
663     }
664
665     public double [] getVelocitiesX(){
666       double [] velocitiesX=new double [elementCount+1];
667       for (int i=0; i<velocitiesX.length; i++)
668       {
669         velocitiesX[i]=velocities[2*i];
670       }
671       return velocitiesX;
672     }
673
674     public double [] getVelocitiesY(){
675       double [] velocitiesY=new double [elementCount+1];
676       for (int i=0; i<velocitiesY.length; i++)
677       {
678         velocitiesY[i]=velocities[2*i+1];
679       }
680       return velocitiesY;
681     }
```

Příloha B: Kompaktní disk

Na přiloženém kompaktním disku se nachází:

- elektronická verze bakalářské práce,
- kompletní zdrojové kódy numerického modelu a grafického rozhraní.