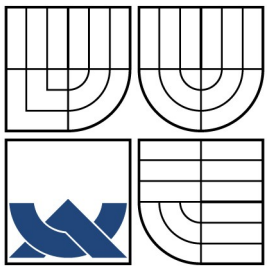
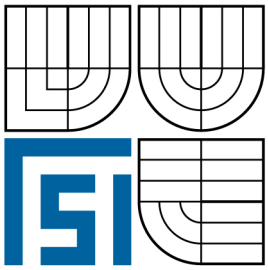


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

APLIKACE MATEMATICKÝCH SOFTWARŮ PŘI ŘEŠENÍ ÚLOH KINEMATIKY A DYNAMIKY

NUMERICAL SOLUTIONS OF EXAMPLES FROM KINEMATICS AND DYNAMICS

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — TOMÁŠ ZEMÁNEK
AUTHOR

VEDOUCÍ PRÁCE — DOC. RNDR. KAREL PELLANT, CSC.
SUPERVISOR

BRNO 2009

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav mechaniky těles, mechatroniky a biomechaniky
Akademický rok: 2008/2009

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

student(ka): Tomáš Zemánek

který/která studuje v **bakalářském studijním programu**

obor: **Strojní inženýrství (2301R016)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Aplikace matematických softwarů při řešení úloh kinematiky a dynamiky

v anglickém jazyce:

Numerical solutions of examples from kinematics and dynamics

Stručná charakteristika problematiky úkolu:

Při výuce kinematiky a dynamiky u řady příkladů není často možné nalézt analytické vyjádření výsledku. Při hledání numerického řešení úloh je proto nutné používání speciálních matematických softwarů.

Cíle bakalářské práce:

Účelem práce je ukázat možnosti využití speciálních softwarů pro numerická rešení úloh mechaniky těles se zaměřením na kinematiku a dynamiku.

Seznam odborné literatury:

Kučera Martin: Řešení úloh statiky v jazyce Python. Bakalářská práce FSI VUT Brno 2007

Vedoucí bakalářské práce: doc. RNDr. Karel Pellant, CSc.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2008/2009.
V Brně, dne 10.12.20008.

L.S.

prof. Ing. Jindřich Petruška, CSc. Ředitel ústavu	doc. RNDr. Miroslav Doupovec, CSc. Děkan fakulty

Abstrakt

Účelem práce je ukázat na možnosti využití speciálních matematických softwarů pro numerické řešení těch příkladů mechaniky těles, kde analytické řešení je obtížné, popř. jej nelze nalézt. Úlohy jsou z kinematiky a dynamiky, jejich případná souvislost s jinými obory vyučovanými na FSI je však zřejmá.

Abstract

The application of special mathematics softwares for numerical solution of examples from mechanics is performed. The submitted examples are from kinematics and dynamics, the relationship with other subjects taught at FME is evident.

Klíčová slova

Scilab, kinematika, dynamika, amplitudová charakteristika, tlumení, řešení úloh

Keywords

Scilab, kinematics, dynamics, amplitude characteristics, damping, task handling

Bibliografická citace dle ČSN ISO 690

ZEMÁNEK, T. *Aplikace matematických softwarů při řešení úloh kinematiky a dynamiky.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2009. 41 s. Vedoucí bakalářské práce doc. RNDr. Karel Pellant, CSc.

Čestné prohlášení

Prohlašuji, že tuto bakalářskou práci „Aplikace matematických softwarů při řešení úloh kinematiky a dynamiky“ jsem vypracoval sám pod vedením doc. RNDr. Karel Pellant, CSc. a všechny použité zdroje ze kterých jsem čerpal informace, jsem uvedl v seznamu literatury.

.....

V Brně dne …..... Tomáš Zemánek

Obsah

1. Úvod - přehled o matematických softwarech zaměřených na vědecko-technické výpočty

1.1 Přehled matematických softwarů

Pro numerické řešení fyzikálních a matematických úloh v současné době existuje celá řada matematických softwarů. Některé z nich jsou stavěny především pro symbolické výpočty, jiné zase pro numerické, ale v převážné většině jsou tyto možnosti kombinovány. Dnes je možné si vybrat z celé řady různých matematických programů, ať už komerčních nebo nekomerčních (*open source, freeware*...).

1.1.1 GNU Octave

Je to asi nejznámější open source software současnosti určený pro numerické výpočty. Do jisté míry je kompatibilní s Matlabem. GNU Octave vznikl v roce 1988 a jeho vývoj se rozjel od roku 1992. Nabízí celou řadu funkcí pro numerické výpočty. Je možné jej provozovat pod systémy jako jsou GNU/Linux, UNIX, Windows a MAC OS. GNU Octave je volně distribuován na internetu, kde je také nabízen volně ke stažení i s oficiální dokumentací [1, 2].

1.1.2 Matlab

Matlab je zkratka slov Matrix Laboratory („laboratoř s maticemi“) [3]. Je to v komerční sféře asi nejznámější matematický program. Nabízí širokou škálu možností pro numerické výpočty, modelování, návrhy algoritmů, simulace, analýzu, prezentaci dat, měření a zpracování signálů, návrhy řídících a komunikačních systémů. Nástavbou Matlabu je Simulink. Simulink je program určený k simulaci a modelování dynamických systémů, přičemž Matlab je využit pro předřešení především nelineárních diferenciálních rovnic. Vlastní programovací jazyk Matlabu vychází z jazyku Fortran.

1.1.3 Mathematica

Mathematica je matematický software určený především pro symbolické výpočty. Ale nabízí také spoustu nástrojů pro numerické počítání. Je to komerční software od firmy Wolfram Research. Mathematica má 2 části – jádro a *frontend*. Jádro interpretuje příkazy a vrací výsledky. Frontend poskytuje GUI (*Graphical user interface* – Grafické rozhraní), ve kterém zobrazuje výsledky z jádra. Mathematica pracuje pod všemi nejrozšířenějšími systémy jako jsou Microsoft Windows, MacOS X, Linux. Solaris (pouze verze UltraSPARC a AMD x86), HP-UX a AIX [4].

1.1.4 wxMaxima

Maxima je matematický software určený pro symbolické výpočty. Pracuje na běžných operačních systémech jako jsou Windows, Linux a MacOS. Maxima má licenci GNU („General Public License“) a je to velice dobrá náhrada za komerční programy jako jsou Maple nebo Mathematica, tedy programy určené především pro symbolické výpočty. Maxima je následníkem slavného algebraického systému Macsyma vyvinutého v 60-tých letech minulého století na univerzitě Massachusetts Institute of Technology. Macsyma se stala také vzorem pro programy Maple a Mathematica. [5, 6].

1.1.5 Scilab

Scilab je vědeckotechnický software určený pro numerické výpočty. Je popsán v kapitolách 2, 3 a praktické příklady jsou v kapitolách 4 a 5.

1.1.6 Yacas

Yacas (yet another computer algebra system) je otevřený počítačový algebraický systém určený pro symbolické výpočty. Yacas nese licenci GPL a veškerá práce s ním probíhá v příkazovém řádku. Je napsaný v C++, díky čemu je snadno přenosný mezi nejrůznějšími systémy (GNU/Linux, Mac OS X, MS Windows a další). Yacas nabízí vlastní programovací jazyk. Celý program obsahuje řadu knihoven obsahující skripty napsané vlastním programovacím jazykem. Díky tomu je Yacas snadno rozšiřitelný [7,8].

1.1.7 Maple

Maple je matematický software určený především pro symbolické výpočty. Vývoj začal již v 80. letech minulého století na několika univerzitách. V současné době je Maple vyvíjen kanadskou firmou a je distribuován jako komerční produkt. Mezi matematické softwary určené pro symboliké výpočty je velmi používaný [9].

1.2 Výběr matematického softwaru pro řešení úloh z kinematiky a dynamiky

Cílem předkládané bakalářské práce je poukázat na možné způsoby numerického řešení úloh z kinematiky a dynamiky pomocí matematických softwarů. Jako vlastní výpočetní proceduru jsem zvolil systém Scilab. Bylo to hlavně z toho důvodu, že jsem se s ním setkal již dříve a líbilo se mi jeho uživatelské prostředí. Nezanedbatelným činitelem byla také ta skutečnost, že je dostupný zdarma ke stažení z internetu a že je v současné době podporován i Evropskou Unií [10]. Je proto zřejmě jej doporučit pro účely současně zaváděné komputerizace výuky mechaniky na technických školách. Vzhledem ke zkušenostem, které jsem během jeho používáním získal, předpokládám, že sám jej využiji i v mé další práci.

2. Scilab

Obr. 1:Logo Scilabu [10]

2.1 Úvod do Slicabu

Scilab je vědecký software určený pro numerické výpočty, který nabízí využití v řadě oborů a má licenci open source software. Vývoj programu začal už v roce 1990 a v současné době za něj odpovídá ústav INRIA (The French National Institute for Research in Computer Science and Control). Scilab zahrnuje velké množství funkcí s možností rozvíjení dalších v řadě jiných jazyků (C, C++, Fortran...). Program umožňuje práci se složitými datovými strukturami (seznamy, polynomy, lineární systémy...). Zahrnuje vlastní interpret a poskytuje vysokou úroveň programovacího jazyka [10].

Scilab byl navržen jako otevřený systém, kde uživatel může definovat nové datové typy a operace s nimi s využitím přetěžování [10].

Scilab zahrnuje množství toolboxů:

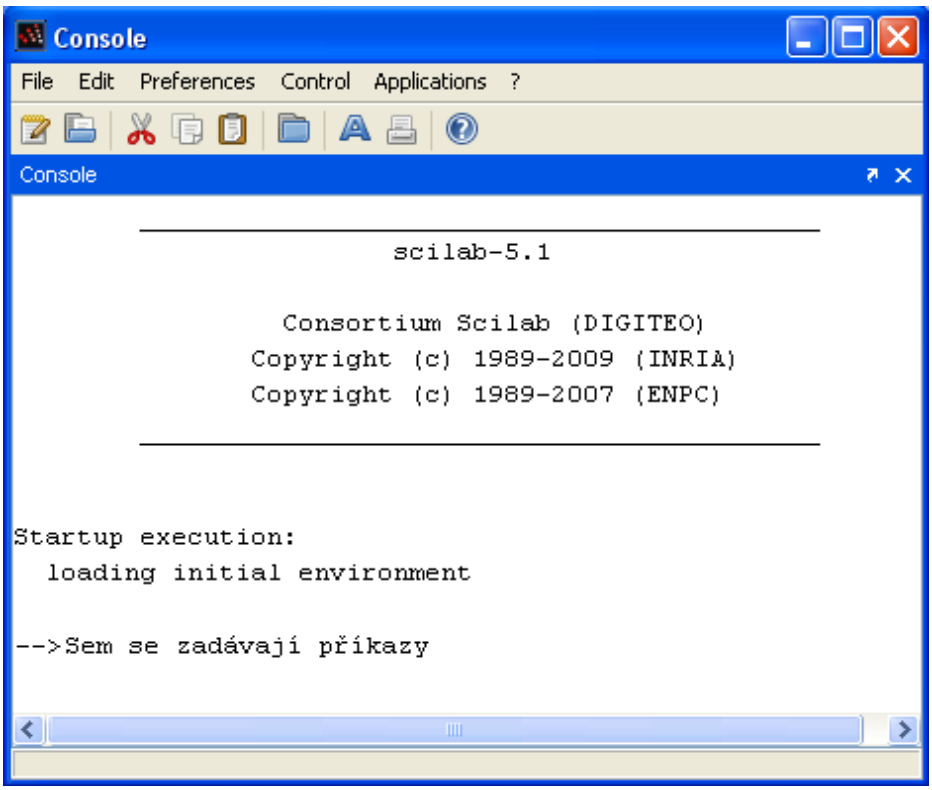
- 2-D a 3-D grafika, animace
- lineární algebra, řídké matice
- polynomiální a logické funkce
- Interpolace, aproximace
- simulace: funkce pro řešení diferenciálních rovnic
- Scicos: modelář a simulátor hybridních dynamických systémů
- klasické a robustní řízení, LMI optimalizace
- diferencovatelná a nediferencovatelná optimalizace
- řízení signálů
- grafy
- paralelní Scilab využívající PVM
- statistika
- prostředí počítačové algebry (Maple, MuPAD)
- prostředí s Tcl/Tk, C, C++, Java, LabVIEW
- A velké množství příspěvků (*contributions*)

Velké množství příspěvků (*contributions*) lze stáhnout z oficiálních internetových stránek Scilabu. Scilab byl vytvořen s využitím množství externích knihoven, pracuje pod systémy Unix (zahrnující GNU/Linux) a

Windows (9X/2000/XP/Vista). Pro uživatele nabízí také nápovědu přímo v programu, nebo také *online* nápovědu na oficiálních stránkách, kde jsou k dispozici i binární verze programu [10].

2.2 Příkazový řádek

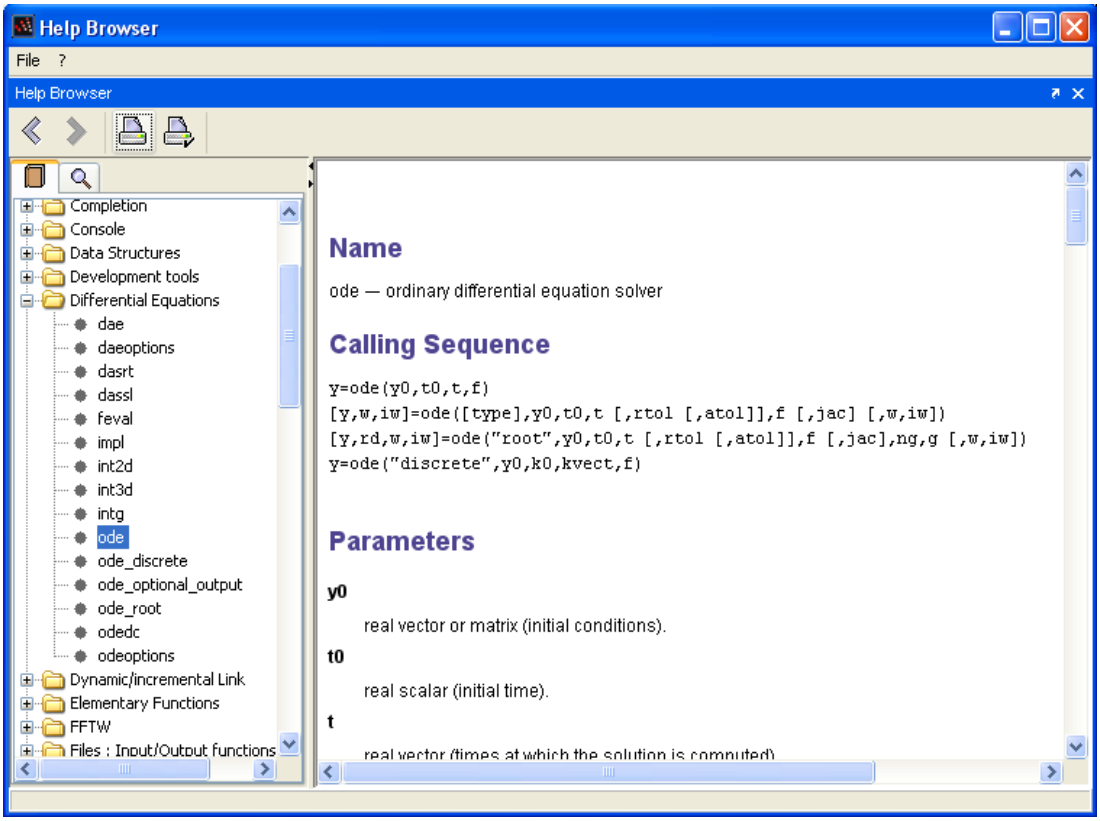
Ihned po spuštění programu nám Scilab nabídne příkazový řádek (konzole), do kterého můžeme psát jakékoliv příkazy (viz obr. 2). Příkazy můžeme buďto psát přímo nebo kopírovat (CTRL+C) a následně vložit (CTRL+V) do příkazového řádku. Šipkami nahoru a dolů můžeme vyvolávat historii příkazů, které jsme od spuštění programu do příkazového řádku zadali [10,11].



Obr. 2: Příkazový řádek v okně Scilabu

2.3 Nápověda

Nápovědu lze vyvolat příkazem help v příkazovém řádku, klávesou F1 nebo v horním menu *?* - *Scilab help*. Nápověda je napsaná v angličtině a francouzštině. Každá položka obsahuje syntaxi, popis vstupních parametrů a popis návratových hodnot funkcí (viz obr. 3). Všechny položky jsou řazeny do různých kategorií, případně podkategorií [10,11].



Obr. 3: Okno nápovědy v programu Scilab

2.4 Proměnné ve Scilabu

Proměnné ve Scilabu mohou být maximálně 24 znaků dlouhé. Název proměnné může obsahovat číslice, písmena nebo znaky %, _, #, !, $, ?, ale nesmí začínat číslicí. Scilab rozlišuje malá a velká písmena (je *case-sensitiv*) [10,11].

2.4.1 Příkaz who a jeho možné syntaxe

Syntaxe:

who

who()
who('local')
who('global')
who('sorted')

Příkaz *who* vypíše všechny aktuálně používané proměnné. Příkazy *who('local')* nebo *who('get')* získáme lokální proměnné (název a paměť) a příkazem *who('global')* získáme globální proměnné (název a paměť). Pokud chceme vypsat všechny proměnné seřazené podle abecedy, použijeme *who('sorted')* [10,11].

2.4.2 Příkaz whos a jeho možné syntaxe

Syntaxe:

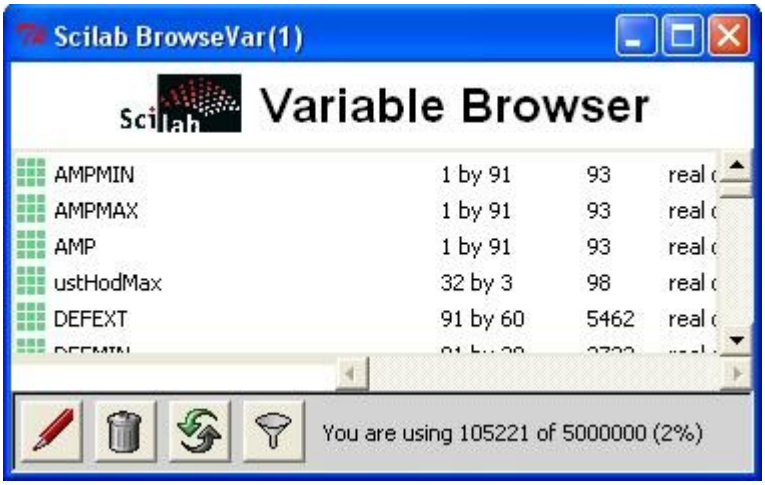
whos()
whos -type *typ* **-name** *jméno*

Příkaz *whos* vypíše seznam proměnných jako u příkazu *who*, ale s tím rozdílem, že výběr můžeme filtrovat podle *typu* (boolean,constant,...) a podle *jména* (prvních znaků názvu proměnné) [10,11].

2.4.3 Příkaz BrowseVar

Syntaxe: **browsevar()**

Seznam uživatelem definovaných proměnných lze také získat příkazem *browsevar.* Výpis proměnných se otevře v novém okně *Scilab Browse Var* (viz obr. 4) [10,11].



Obr. 4: Okno browsevar

Tento prohlížeč umožňuje pracovat s uživatelem definovanými proměnnými v prostředí TCL/TK. Po otevření okna dostaneme podrobný seznam proměnných (podobně jako u příkazu whos). Okno obsahuje 4 tlačítka, kterými můžeme přepisovat hodnotu proměnné, mazat ji, obnovit seznam proměnných a filtrovat zobrazení [11].

2.5 Speciální konstanty

Scilab obsahuje některé předdefinované konstanty (viz tabulka 1). Jejich název vždy začíná %. Konstanty nejdou mazat ani jim nelze upravovat hodnotu.

%i	imaginární jednotka (i^2=-1)
%pi	π = 3.1415927
%e	základ přirozených logaritmů e = 2.7182818
%eps	maximální hodnota, pro kterou platí 1 + %eps = 1 (eps = 2.22 . 10^{-16})
%inf	infinity (nekonečno)
%nan	not-a-number
%t	true (pravda)
%f	false (nepravda)

Tabulka 1: Přehled konstant ve Scilabu [11]

2.6 Datové typy

Proměnné mohou nabývat různých datových typů (viz tabulka 2).

1	konstanta, reálná nebo komplexní konstantní matice
2	polynom, polynomická matice
4	boolean, logická matice
5	řídká matice
6	řídká logická matice
8	integer,celočíselná matice
9	handle (grafický objekt)
10	string (řetězec)
11	nezkompilovaná funkce
13	zkompilovaná funkce
14	knihovna funkcí
15	seznam (list)
16	tlist
17	mlist
18	pointer (ukazatel)

Tabulka 2: Přehled datových typů ve Scilabu [11]

2.6.1 Funkce type

Syntaxe: **type(x)**

Návratová hodnota funkce *type* je identifikační číslo typu proměnné *x* (viz. Tabulka 2) [11].

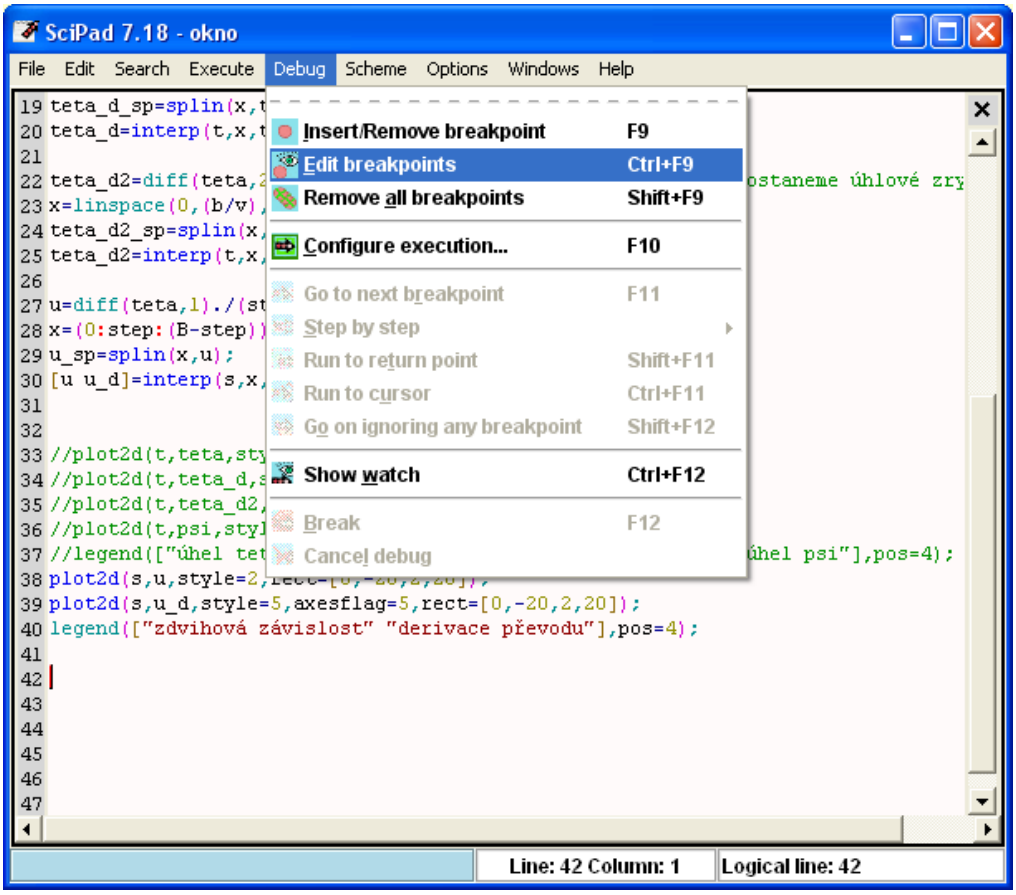
2.6.2 Funkce typeof

Syntaxe: **typeof(object)**

Návratová hodnota funkce *typeof* je název typu *object* [11].

2.7 Editor Scipad

Scipad je vestavěný editor ve Scilabu (viz. Obr 5), ve kterém můžeme psát skripty aniž bychom museli pracovat s příkazovým řádkem. Ovládá spoustu užitečných funkcí jako je barevně zvýrazněná syntaxe, debugger, vyhledávání ve skriptu, ukládání a otvírání skriptů (*.sci, *.sce) apod. Scipad spustíme příkazem *scipad()* nebo pomocí ikonky v levém horním rohu příkazového řádku [11].



Obr. 5: Okno Scipadu

3. Programování ve Scilabu

Ve Scilabu můžeme vytvářet krátké programy (skripty), které obsahují sadu instrukcí. Instrukcemi jsou například funkce, procedury a přiřazování hodnot proměnným. Každá instrukce je zakončena středníkem nebo odřádkováním. Program můžeme vyvolat z příkazového řádku pomocí příkazu *exec*(cesta) nebo v případě vestavěného editoru Scipadu *Execute – Load into Scilab* (CTR+l). Komentáře v programu slouží k popisu jednotlivých instrukcí a zapisují se za dvojité lomítko [11,12].

3.1 Operátory

Používají se pro základní operace s proměnnými jako je sčítání, odčítání, násobení, dělení a umocňování [11,12]. Pro dělení nulou má Scilab 3 různá nastavení. V případě základního nastavení při dělení nulou Scilab ukončí instrukci případně celý skript a vypíše chybu, že se dělí nulou. V mnoha případech je však účelné, aby při dělení čísla nulou byla návratová hodnota *+-%inf* (nekonečno). Toho lze dosáhnout pomocí funkce *ieee(hodnota)*. Následný vstupní parametr *hodnota* může nabývat následujících hodnot:

- 0 – Je základní nastavení při kterém Scilab jak už bylo popsáno výše, ukončí instrukci případně celý skript a vypíše chybu („Division by zero...“)
- 1 – Při tomto nastavení je možné dělit nulou a návratová hodnota je *+-%inf*. Scilab při každém dělení nulou při tomto nastavení vypíše varování, že dělí nulou, ale neukončí instrukci, jako tomu je při nastavení 0.
- 2 – Po dělení nulou je návratová hodnota *+-%inf* a není vypsáno žádné varování o dělení nulou jako při nastavení 1.

3.1.1 Aritmetické operátory

+	sčítání
-	odčítání
*	násobení
/	dělení
^ nebo **	exponent

Tabulka 3: Přehled aritmetických operátorů [11]

3.1.2 Maticové operátory

[ ]	definice matice, zřetězení
;	oddělovač řádků
( )	Když chceme vrátit prvek matice
’	transpozice
+	součet
-	rozdíl
*	násobení
\	levé dělení
/	pravé dělení
^	exponent
.*	násobení jednotlivých prvků
.	levé dělení jednotlivých prvků
./	pravé dělení jednotlivých prvků
.^	exponent jednotlivých prvků
.*.	násobení každý prvek s každým

Tabulka 4: Přehled maticových operátorů [11,12]

3.1.3 Logické operátory

Používají se společně s relačními operátory, když potřebujeme vyhodnotit více podmínek najednou.

&	and (konjunkce)
|	or (disjunkce)
~	not (negace)

Tabulka 5: Přehled logických operátorů [11,12]

3.1.4 Relační operátory

Relační operátory se používají pro porovnání výrazu a vrací *%t* (true) nebo *%f* (false) podle toho, zda je podmínka splněna či nikoliv.

==	rovnost
>	větší než
<	menší než
>=	větší nebo rovno
<=	menší nebo rovno
<> nebo ~=	nerovnost

Tabulka 6: Přehled relačních operátorů [11,12]

3.2 Procedury a funkce

Ve Scilabu si můžeme definovat procedury a funkce, což jsou vlastně bloky programu, které můžeme zavolat jediným příkazem, aniž bychom museli pokaždé vypisovat všechny instrukce. Rozdíl mezi procedurou a funkcí je takový, že procedura stejně jako funkce vyvolá souhrn instrukcí, ale funkce na rozdíl od procedury vrací nějakou hodnotu. Jakmile už je funkce jednou definována a my se jí snažíme předefinovat, Scilab vypíše varování, že funkce byla předefinována. Tomuto zabráníme, pokud nastavíme *funcprot(0)*. Funkce vyvoláme příkazem *název funkce (hodnota1, hodnota2, ...);*

Syntaxe :
function [*návratová hodnota***=] *název funkce* (***parametry***)**
    sada instrukcí
endfunction

Uvnitř funkce můžeme pracovat i s proměnnými, které jsou definované mimo funkci [11,12].

3.3 Podmínky

3.3.1 If – then – else

Syntaxe :
if *výraz1* **then** *akce1*
elseif *výraz2* **then** *akce2*
....
[else *základníAkce***]**
end
Pokud platí *výraz1,* potom se provede *akce1*. Pokud neplatí *výraz1*, ale platí *výraz2,* tak se provede *akce2*. A pokud neplatí ani jeden z vyhodnocených výrazů, tak potom se provede *základníAkce*, pokud je definována [11,12].

3.3.2 Select - case

Syntaxe :
select *výraz*
case *hodnota1* **then** *akce1*
case *hodnota2* **then** *akce2*
...
case *hodnotan* **then** *akcen*
[else *základníAkce***]**
end

Podle hodnoty *výrazu* se provede daná *akce*. V případě, že žádný *výraz* nevyhovuje žádné *hodnotě*, provede se *základníAkce*, pokud je definována [11,12].

3.4 Cykly

Pomocí cyklů můžeme provádět příkaz, nebo blok programu opakovaně s proměnlivým počtem opakování. Cykly se uskutečňují pomocí příkazu for a *while*, přičemž u cyklu *for* nastavíme pevný počet cyklů a u cyklu *while* nastavíme podmínku, která když je splněna, tak se cyklus opakuje do doby, kdy podmínka splněna není. Běh cyklu můžeme také řídit pomocí dvou instrukcí *break* a *continue*. Příkaz *break* přeruší vykonávání cyklu a program pokračuje dál za cyklem. Za to pomocí *continue* přerušíme aktuální cyklus a program pokračuje od začátku cyklu [12].

3.4.1 For

Syntaxe :
for *i* = *výraz* **do**
    instrukce
end

Pomocí *výrazu* zvolíme počet cyklů, které vykonávají všechny *instrukce* uvnitř cyklu. *Výraz* může být například 1:1:100. V takovém to případě by se provedlo 100 cyklů.

3.4.2 While

while *podmínka*
    instrukce
[**else**
    elseInstrukce
]
end

Dokud je *podmínka* splněna, tak se cyklus neustále opakuje a provádějí se příslušné *instrukce* uvnitř cyklu. Jakmile *podmínka* splněna není, tak se provede *elseInstrukce,* pokud je definována, a pokud ne, tak program pokračuje v běhu za cyklem [11,12].

4. Aplikace Scilabu pro řešení některých úloh kinematiky

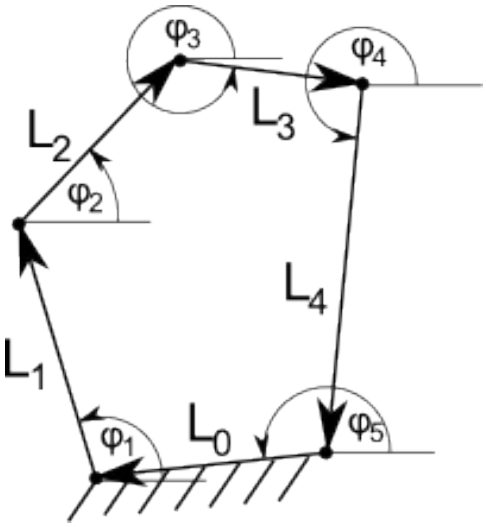
4.1 Použití Scilabu pro řešení kinematiky mechanismů vektorovou metodou

4.1.1 Podstata vektorové metody

Při řešení kinematiky mechanismů je zpravidla zadán pohyb hnacích členů a vyšetřujeme pohyb členů hnaných. Vektorová metoda popisuje schéma rovinných mechanismů pomocí vektorového mnohoúhelníku a dává obecný návod jak získat rovnice pro polohu i rovnice pro rychlosti a zrychlení. Jednoduché mechanismy jsou popsány jedním mnohoúhelníkem, složité mechanismy jsou popsány s mnohoúhelníky:

$$s = d - m + 1$$

kde m je počet členů mechanismu, d je počet kinematických dvojic. Promítáním jednotlivých vektorů do vhodně zvoleného souřadného systému se pak vygeneruje systém rovnic umožňující nalezení vztahů mezi kinematickými veličinami. Analytické řešení soustavy rovnic je však často složité a soustavy je pak možné řešit jen numericky (viz např. [13]).



Obr. 6: Vektorové mnohoúhelníky

Vektorová metoda vychází z předpokladu :

$$\sum \vec{L}_i = \vec{0} \qquad (4.1a)$$

Tuto rovnici rozepíšeme na rovnice jednotlivých složek jako :

$$\sum L_i \cos(\varphi_i) = 0 \qquad (4.1b)$$
$$\sum L_i \sin(\varphi_i) = 0 \qquad (4.1c)$$

Rovnice rychlostí obdržíme derivováním těchto rovnic :

$$\sum \left(\dot{L}_i \cos(\varphi_i) - L_i \dot{\varphi}_i \sin(\varphi_i) \right) = 0 \qquad (4.1d)$$
$$\sum \left(\dot{L}_i \sin(\varphi_i) + L_i \dot{\varphi}_i \cos(\varphi_i) \right) = 0 \qquad (4.1e)$$

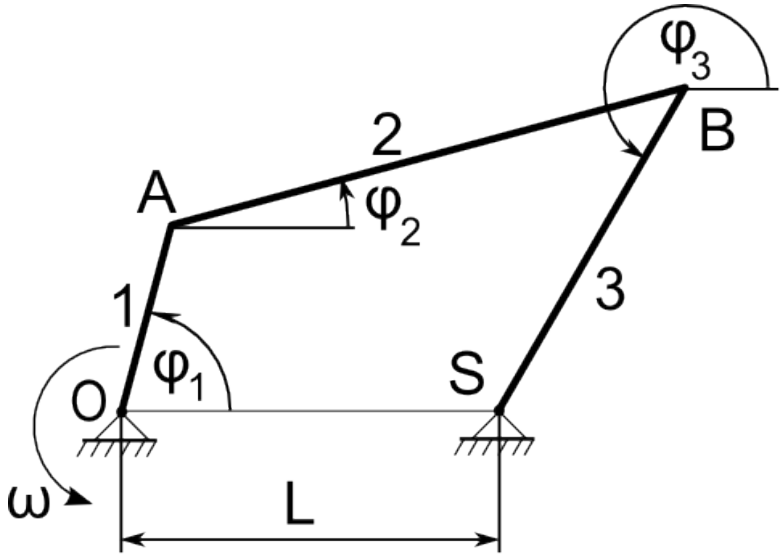
Rovnice zrychlení dostaneme derivací rovnic rychlostí :

$$\sum \left(\ddot{L}_i \cos(\varphi_i) - 2\dot{L}_i \dot{\varphi}_i \sin(\varphi_i) - L_i \dot{\varphi}_i^2 \cos(\varphi_i) - L_i \ddot{\varphi}_i \sin(\varphi_i) \right) = 0 \qquad (4.1f)$$
$$\sum \left(\ddot{L}_i \sin(\varphi_i) - 2\dot{L}_i \dot{\varphi}_i \cos(\varphi_i) - L_i \dot{\varphi}_i^2 \sin(\varphi_i) - L_i \ddot{\varphi}_i \cos(\varphi_i) \right) = 0 \qquad (4.1g)$$

4.1.2 Aplikace Scilabu při řešení čtyřkloubového mechanismu

V minulém odstavci byla vysvětlena podstata vektorové metody. Na následujícím příkladě je uvedena konkrétní aplikace Scilabu pro řešení čtyřkloubového mechanismu.

Příklad 1 : Vyšetřete pohyb vahadla (člen 3 na obr. 4), dané rychlosti a zrychlení při konstantní úhlové rychlosti ω_{21} . $L_1=0.1\text{m}$, $L_2=0.3\text{m}$, $L_3=0.25\text{m}$, $L=0.2\text{m}$, $\omega_{21}=15\text{rad}\cdot s^{-1}$



Obr. 7: Schéma čtyřkloubového mechanismu

Aplikujeme vektorovou metodu na zadaný mechanismus

$$\vec{OA}+\vec{AB}+\vec{BS}+\vec{SO}=0 \qquad (4.1a)$$

Rovnici (4.1a) pak následně rozepíšeme :

$$x: l_1\cos(\varphi_1)+l_2\cos(\varphi_2)+l_3\cos(\varphi_3)-L=0 \qquad (4.1b)$$

$$y: l_1\sin(\varphi_1)+l_2\sin(\varphi_2)+l_3\sin(\varphi_3)=0 \qquad (4.1c)$$

Soustavu budeme řešit prostou iterací, proto sestavíme z rovnic (4.1b) a (4.1c) funkce f_1 a f_2

$$f_2(\varphi_1,\varphi_2,\varphi_3)=l_1\cos(\varphi_1)+l_2\cos(\varphi_2)-l_3\cos(\varphi_3)-L \qquad (4.1d)$$

$$f_1(\varphi_1,\varphi_2,\varphi_3)=l_1\sin(\varphi_1)+l_2\sin(\varphi_2)-l3\sin(\varphi_3) \qquad (4.1e)$$

Využijeme vestavěný editor Scipad a v něm napíšeme skript, který vypočítá příklad a vytiskne na obrazovku výsledek.

Funkcí *funcprot* s parametrem *0* nastavíme, aby Scilab po každém vykonání skriptu nevypisoval varování, že došlo k předefinování funkce při každém vykonání skriptu. Toto nastavení není povinné.

```
funcprot(0);
```

Dle zadání uložíme všechny hodnoty do proměnných a odhadneme první řešení soustavy rovnic. To z toho důvodu, že soustavu rovnic budeme řešit prostou iterací (pomocí funkce *fsolve*), která spočívá v odhadnutí prvních kořenů a následnému přibližování se k výsledku. V našem případě nenastavíme konečný počet cyklů, ani maximální možnou chybu. Funkce tedy bude hledat výsledek podle základního nastavení s maximální chybou $1\cdot10^{-10}$.

```
L1=0.1;
```

```
L2=0.3;
L3=0.25;
L=0.2;
start_fi2=0.5;
start_fi3=4.5;
omg21=15;
fi1=[];
fi2=[];
fi3=[];
fi_act=0;
```

Zapíšeme systém rovnic (4.1d) a (4.1e) do funkce *f*, dle tvaru který vyžaduje funkce *fsolve*.

```
function [f]=f(angle)
  f(1)=L1*cos(fi_act)+L2*cos(angle(1))+L3*cos(angle(2))-L;
  f(2)=L1*sin(fi_act)+L2*sin(angle(1))+L3*sin(angle(2));
endfunction;
```

Vzhledem k tomu, že pro goniometrické funkce platí $\cos(\varphi_i)=\cos(\varphi_i+k\,2\pi)$ a funkce *fsolve* hledá kořeny řešení na základě iterací, může dojít k tomu, že vypočtené hodnoty úhlů φ_2 a φ_3 vyjdou mimo interval $\langle -2\pi\,;2\pi\rangle$. Při výpočtu proto provedeme kontrolu, jestli výsledné úhly leží v intervalu $\langle -2\pi\,;2\pi\rangle$. V případě že ne, posuneme vypočtené závislosti φ_2 a φ_3 o hodnotu $\pm 2k\pi$ do intervalu $\langle -2\pi\,;2\pi\rangle$. Při samotném posunu vycházíme z předpokladu, že členy 2 a 3 jsou vahadla, a že závislosti φ_2 a φ_3 neobsahují „falešné skoky“ o 2π.

```
function upraveny_uhel=upravUhel(uhel)
  if max(uhel)>=0 then
    korekce=floor(max(uhel)/(2*%pi))*2*%pi;
    upraveny_uhel=uhel-korekce;
  else
    korekce=floor(max(uhel)/(2*%pi))*2*%pi;
    upraveny_uhel=uhel-korekce;
  end;
endfunction;
```

Nyní si definujeme časový krok a časový interval $\langle a\,;b\rangle$, pro který chceme analyzovat pohyb. Vytvoříme vektor času, z kterého vypočítáme úhel natočení $\varphi_1=\omega\cdot t$.

```
krok=0.001*%pi;
a=0;
b=%pi/4;
t=a:krok:b;
fi1=omg21*t;
```

Zjistíme, kolik cyklů bude potřeba, pro řešení naší soustavy rovnic podle počtu členů ve vektoru *fi1*.

```
[m n]=size(fi1);
```

Vytvoříme matici *UHLY*, do které budeme ukládat výsledky. Cyklem *for* dopočítáme úhly φ_2 a φ_3 pro každý úhel φ_1 ve vektoru *fi1*. Výsledky úhlů φ_2 a φ_3 požijeme pro odhad kořenů v dalším cyklu.

```
UHLY=[];
uhel_fi2=start_fi2;
uhel_fi3=start_fi3;
for i=1:n
  fi_act=fi1(i);
  UHLY=[UHLY,fsolve([uhel_fi2;uhel_fi3], f)];
  uhel_fi2=UHLY(1,i);   //  pristi odhadované fi2
  uhel_fi3=UHLY(2,i);   //  pristi odhadované fi3
end;
```

Uložíme výsledky do vektorů *fi2*,*fi3* a pomocí funkce *upravUhel* zajistíme, aby všechny úhly začínali v intervalu $\langle -2\pi\,;2\pi\rangle$.

```
fi2=upravUhel(UHLY(1,:));
fi3=upravUhel(UHLY(2,:));
```

Úhel φ_3 (úhel natočení vahadla) derivujeme podle času ($\dot{\varphi}_3 = \frac{d\varphi_3}{dt}$). Pro tento účel použijeme funkci *diff*, která vypočítá diferenciál ($d\varphi_3$) libovolného řádu a ten následně vydělíme časovou diferencí (dt), tedy časovým krokem. Obdobně provedeme i druhou derivaci ($\ddot{\varphi}_3 = \frac{d^2\varphi_3}{dt^2}$). Nesmíme zapomenout, že při numerickém derivování ztratíme, při první derivaci jeden člen vektoru a při druhé derivaci dva členy vektoru. Proto dopočítáme okrajové hodnoty derivovaných vektorů pomocí extrapolace tj. funkcí *splin* a *interp*.

```
fi3_d=diff(fi3,1)./krok;
[m n]=size(fi3_d);
x=linspace(a,b,n);
fi3_d_spline=splin(x,fi3_d);
fi3_d_interp=interp(t,x,fi3_d,fi3_d_spline);

fi3_d2=diff(fi3,2)./krok^2;
[m n]=size(fi3_d2);
x=linspace(a,b,n);
fi3_d2_spline=splin(x,fi3_d2);
fi3_d2_interp=interp(t,x,fi3_d2,fi3_d2_spline);
```

Všechny výsledky jsme vypočítaly a nyní je zobrazíme v grafu. Pro lepší čitelnost grafu zvolíme měřítka pro úhlovou rychlost, úhlové zrychlení a také rozmezí na ose y.

```
meritko_fi3_d=5;
meritko_fi3_d2=100;
minimum=-4;
maximum=7;
```

Nyní upravíme příslušné veličiny do zvolených měřítek.

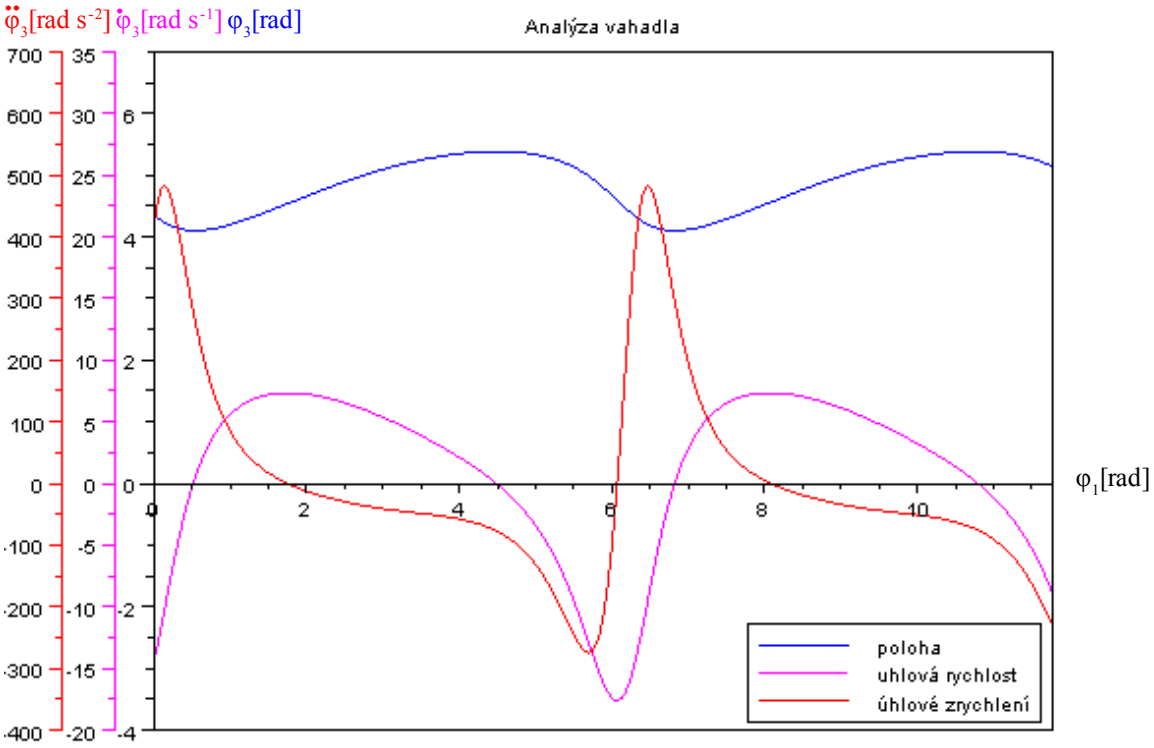
```
fi3_d_disp=fi3_d_interp/meritko_fi3_d;
fi3_d2_disp=fi3_d2_interp/meritko_fi3_d2;
```

Teď zobrazíme výsledek. Nejprve vytvoříme nové grafické okno. Poté vykreslíme osy pro úhlovou rychlost a úhlové zrychlení vahadla s upravenými hodnotami podle zvolených měřítek. Nakonec zakreslíme jednotlivé křivky a legendu.

```
okno = scf(1);
osy=gca();
osy.y_location="left";
osy.x_location="middle";
osy.box="on";
osy.foreground=1;
drawaxis(x=fi1(1)-0.5,y=minimum:1:maximum,dir='l',tics='v',val=string(minimum*meritko_fi3_d:meritko_fi3_d:maximum*mer
itko_fi3_d),ticscolor=6,fontsize=1);
drawaxis(x=fi1(1)-1.2,y=minimum:1:maximum,dir='l',tics='v',val=string(minimum*meritko_fi3_d2:meritko_fi3_d2:maximum*m
eritko_fi3_d2),ticscolor=5,fontsize=1);
plot2d(fi1,fi3,style=2);
plot2d(fi1,fi3_d_disp,style=6);
plot2d(fi1,fi3_d2_disp,style=5,rect=[0,minimum,b,maximum]);
legend('poloha', 'uhlová rychlost', 'úhlové zrychlení', 4);
xtitle("Analýza vahadla");
```

Poznámka : Řádky, které se nevešly do jednoho, je nutné při vkládání do Scilabu vložit jako jeden, jinak by se skript nevykonal. Je to z toho důvodu, že jednotlivé instrukce končí buď středníkem, nebo odřádkováním. To se týká funkcí drawaxis, které jsou rozepsané na 2 řádky.

Průběh úhlového natočení, úhlové rychlosti a úhlového zrychlení vahadla (člen 3) v závislosti na úhlovém natočení hnací kliky (člen 1) je na obr. 8.



Obr. 8: Úhlové natočení , úhlová rychlost a úhlové zrychlení vahadla (člen 3) v závislosti na úhlovém natočení kliky (člen 1)

U tohoto příkladu je možné nalézt i řešení analytické. Hodnoty kinematických veličin získané na základě analytického řešení (jehož zápis je poměrně komplikovaný) a numerického řešení pomocí funkce *fsolve* byli prakticky stejné tj. jednotlivé křivky se ve sledovaném intervalu úhlů přesně překrývaly. V našem příkladě funkce *fsolve* vypočítala výsledek za velmi krátkou dobu a nebylo třeba výpočet urychlovat například stanovením větší možné odchylky v hodnotách vypočítávaných úhlů natočení, omezením počtu iterací nebo prodloužením časového kroku. Jediný problém u tohoto příkladu byl, že výsledek vycházel mimo interval $\langle -2\pi\,;2\pi\rangle$ vlivem přítomnosti goniometrických funkcí v systému transcendentních rovnic. Tento problém byl ošetřen jednoduchou funkcí *upravUhel*, která posunula výsledek úhlového natočení tak, aby její počátek ležel v intervalu $\langle -2\pi\,;2\pi\rangle$.
V případě složitějších mechanismů analytické řešení již zpravidla nalézt nelze. Výhody numerického řešení pomocí funkce *fsolve* pro tyto případy je zcela evidentní.

4.2 Použití Scilabu pro řešení mechanismů s nekonstantními převody

4.2.1 Veličiny charakterizující mechanismy s nekonstantním převodem

Podle převodu se mechanismy dělí na mechanismy s konstantním převodem (závislost hnané souřadnice na hnací je lineární) a mechanismy s nekonstantním převodem (závislost hnané souřadnice na hnací je nelineární). Závislost souřadnice hnaného členu ψ. na hnací souřadnici φ nazýváme zdvihovou závislostí [14].

zdvihová závislost se určí podle vztahu :

$$\psi = \varphi(s) \tag{4.3a}$$

převodová funkce se určí podle vztahu :

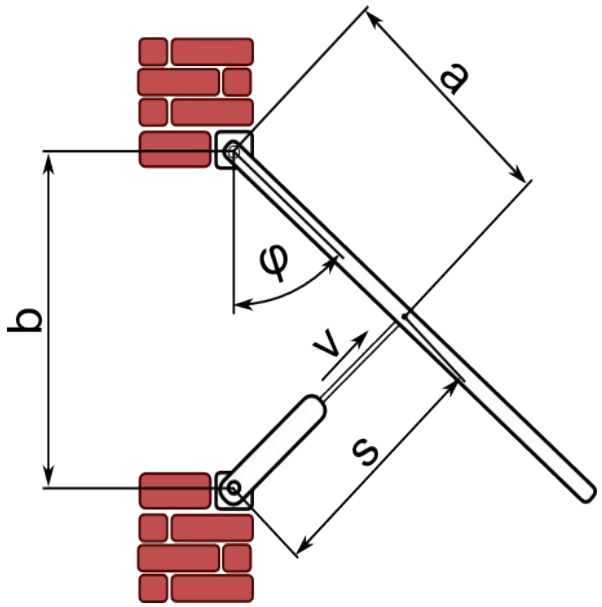
$$\mu = \frac{d\,\varphi}{d\,s} \tag{4.3b}$$

a derivace převodu se určí podle vztahu :

$$\nu = \frac{d^2\varphi}{d\,s^2} \tag{4.3c}$$

4.2.2 Výpočet převodové funkce při otvírání okna hydraulickým válcem

Příklad 2 : Okno (obr. 9) se otevírá za pomocí hydraulického válce při konstantní rychlosti válce. Určete úhlovou rychlost, úhlové zrychlení a převodový poměr okna. $a = 1\text{m}\,,\, b = 2\text{m}\,,\, v = 0.5\text{ms}^{-1}$



Obr. 9: Schéma okna

Vycházíme z Kosinovy věty :

$$s^2 = a^2 + b^2 - 2\,a\,b\cos(\varphi) \tag{4.4a}$$

Protože je rychlost zdvihání válce konstantní, tak můžeme napsat :

$$s = b - a + v\cdot t \tag{4.4b}$$

Časový interval od zahájení pohybu až po úplné otevření :

$$t \in \left\langle 0, \frac{2\text{a}}{v} \right\rangle$$

Z rovnic (4.4a) a (4.4b) vyjádříme φ jako funkci času :

$$\varphi = a\cos\left[\frac{a^2 + b^2 - (b - a + v\cdot t)^2}{2\cdot a\cdot b}\right] \tag{4.4c}$$

zdvihovou závislost, převodovou funkci a derivaci převodu určíme podle vztahů (4.3a), (4.3b), (4.3c).

Máme zapsané všechny potřebné rovnice a nyní s nimi budeme pracovat ve Scilabu. Vytvoříme skript ve vestavěném editoru Scipad, který vypočítá všechny závislosti a zobrazí je v grafu.

Jelikož Scilab při dělení nulou vyhodí chybu a ukončí tak provádění skriptu, nastavíme *ieee(2)*. To způsobí, že po dělení nulou bude návratová hodnota +-*%inf* (nekonečno).

```
ieee(2);
```

Definujeme proměnné podle zadání.

```
a=1;
b=2;
v=0.5;
```

Vytvoříme vektor času t, do kterého vložíme hodnoty, pro které chceme analyzovat pohyb a dále vytvoříme vektor s z rovnice (4.4b).

```
krok=0.01;
t1=0;
t2=2*a/v;
t=t1:krok:t2;
s=b-a+v*t;
```

Nyní vypočítáme úhel φ v závislosti na čase z rovnice (4.4c).

```
fi=acos((a^2+b^2-(b-a+v*t).^2)/(2*a*b));
```

Nyní vypočítáme $\dot{\varphi}$ a $\ddot{\varphi}$. Protože po derivaci se vektor zmenší o jeden prvek odpovídající hodnotě posledního času , určíme jej pomocí extrapolace, to samé provedeme i u druhé derivace vektoru φ.

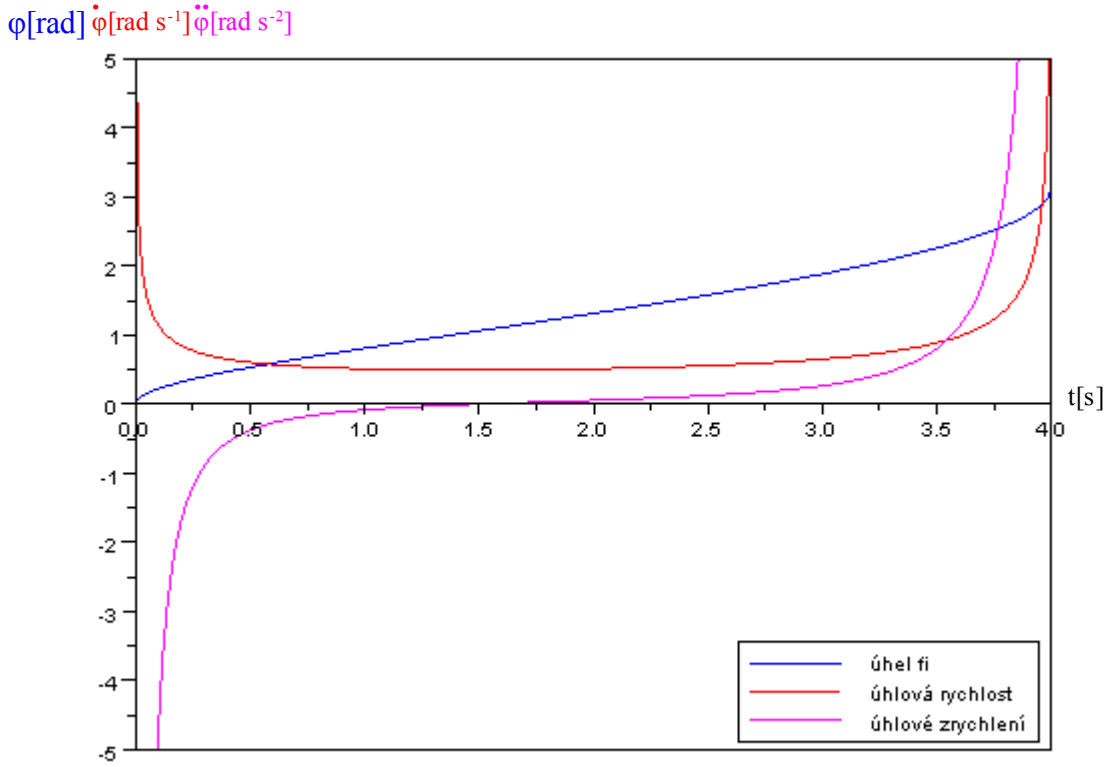
```
[m n]=size(fi);
fi_d=diff(fi,1)./krok;
x=linspace(0,(b/v),n-1);
fi_d_splin=splin(x,fi_d);
fi_d_interp=interp(t,x,fi_d,fi_d_splin,"natural");

fi_d2=diff(fi,2)./krok^2;
x=linspace(t1,t2,n-2);
fi_d2_splin=splin(x,fi_d2);
fi_d2_interp=interp(t,x,fi_d2,fi_d2_splin,"natural");
```

Nyní určíme převodovou funkci podle rovnice (4.3b). Opět extrapolujeme poslední člen a tentokrát využijeme u funkce *interp* možnost, že návratová hodnota je jak extrapolovaný vektor (převodová funkce) tak i jeho první derivace (derivace převodu).

```
u=diff(fi,1)./(krok*v);
x=linspace(a,b+a,n-1);
u_sp=splin(x,u);
[u u_d]=interp(s,x,u,u_sp,"natural");
```

Všechny výsledky jsou vypočítané a teď si je zobrazíme v grafu následujícím způsobem.

```
okno1=scf(1);
plot2d(t,fi,style=2,rect=[0,-5,4,5]);
plot2d(t,fi_d_interp,style=5,rect=[0,-5,4,5]);
plot2d(t,fi_d2_interp,style=6,axesflag=5,rect=[0,-5,4,5]);
legend(["úhel fi" "úhlová rychlost" "úhlové zrychlení"],pos=4);
```

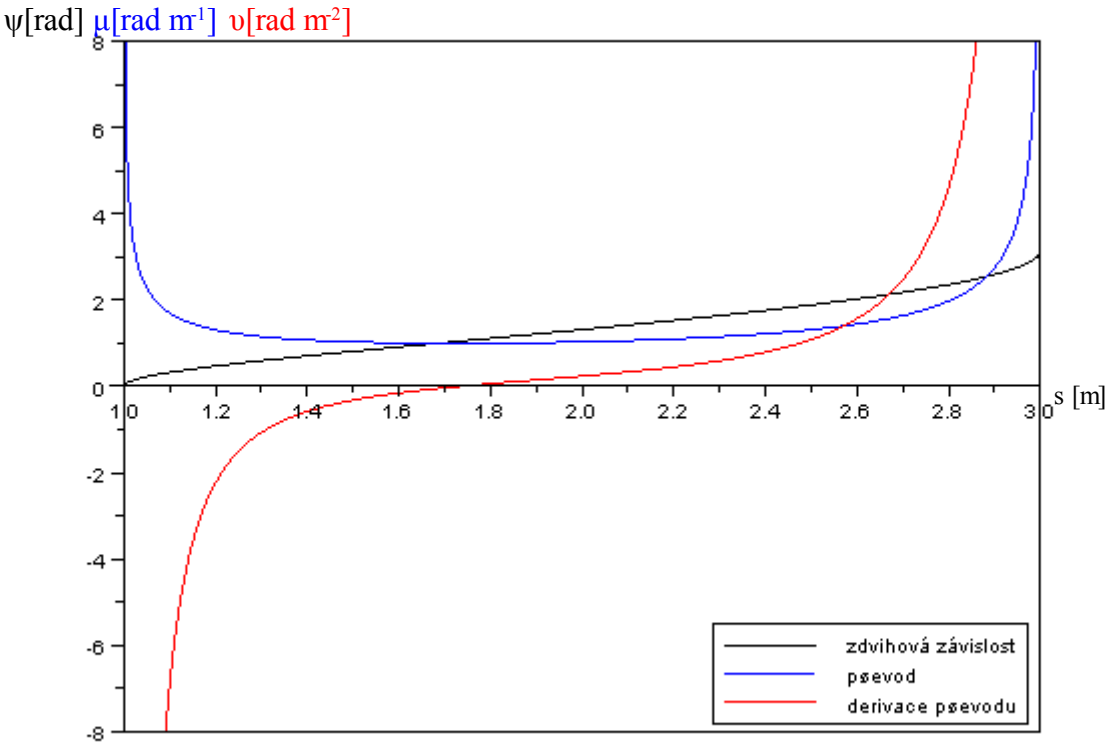


Obr. 10: Závislost úhlového natočení , úhlové rychlosti a úhlového zrychlení na čase při zvedání okna

Z obr. 10 je zřejmé, že okno má na počátku pohybu a na konci vysoké hodnoty rychlosti a zrychlení. Singularity v rychlosti odpovídají „mrtvým“ polohám, kdy daná soustava je ve statické rovnováze. Vysoké hodnoty úhlového zrychlení z hlediska dynamiky představují vysoké hodnoty setrvačných silových účinků tj. značné namáhání čepů. Okno by tedy muselo být zvedáno z počátečního nenulového úhlu natočení a jeho pohyb ukončen před úplným zvednutím.

Grafy zdvihové závislosti, převodového poměru a derivaci převodu si zobrazíme následujícím způsobem.

```
okno2=scf(2);
osy=gca();
osy.x_location="middle";
plot2d(s,fi,style=1);
plot2d(s,u,style=2,rect=[1,-8,3,8]);
plot2d(s,u_d,style=5,axesflag=1,rect=[1,-8,3,8]);
legend(["zdvihová závislost" "převod" "derivace převodu"],pos=4);
```



Obr. 11: Průběh zdvihové závislosti, převodu a derivace převodu

Z obr. 11 je zřejmé, že funkce převodu a derivace převodu se v daném případě podobají závislostem úhlové rychlosti a úhlového zrychlení, jen jsou v jiném měřítku. To lze dokázat :

$$\mu = \frac{d\varphi}{ds} = \frac{d\varphi}{dv \cdot t} = \frac{1}{v} \cdot \frac{d\varphi}{dt} = \frac{1}{v}\dot{\varphi}$$

Rychlost zdvihání pístu je totiž v našem případě konstantní a je rovna 0,5. Takže ve výsledku hodnoty převodu a derivace převodu se od závislostí úhlové rychlosti a úhlového zrychlení liší jen o dvojnásobek.

5. Aplikace Scilabu pro výpočet frekvenčních charakteristik mechanických soustav

5.1 Vynucené kmity při harmonickém buzení

Pohybová rovnice vynucených kmitů tělesa hmotnosti *m,* na které působí harmonicky proměnná síla $F(t) = F_0 \sin \omega t$ je dána vztahem

$$\ddot{x} + 2\delta \dot{x} + \Omega_0^2 x = \frac{F(t)}{m} = \frac{F_0}{m} \sin \omega t \qquad (5.1a)$$

kde δ je součinitel doznívání , Ω_0 je frekvence netlumených kmitů. Pohybová rovnice je tedy nehomogenní diferenciální rovnicí 2.řádu. Řešení této rovnice je dáno součtem řešení rovnice homogenní x_h a partikulárního řešení rovnice x_p tj. $x = x_h + x_p$. *Při podkritickém tlumení* ($\frac{\delta}{\Omega_0} < 1$) řešení homogenní x_h po čase zaniká, kmity se ustálí, průběh těchto ustálených vynucených kmitů je dána řešením partikulárním. Vzhledem ke tvaru pravé straně rovnice ($\frac{F_0}{m} \sin \omega t$) toto partikulární řešení x_p hledáme na bázi harmonických funkcí. tj. např. $x_p = s_0 \sin(\Omega t + \varphi_0)$, kde $\Omega = \sqrt{\Omega_0^2 - \delta^2}$ je vlastní frekvence tlumených kmitů. Po vložení do rovnice (5.1a) a úpravách pro závislost amplitudy ustálených vynucených kmitů s_0 na frekvenci dostáváme vztah [14]

$$s_0 = \frac{F_0}{m\sqrt{\left(\Omega_0^2 - \omega^2\right)^2 + \left(2\delta\omega\right)^2}} \qquad (5.1b)$$

Fázový úhel φ mezi výchylkou a budící silou je

$$\varphi = -\operatorname{arctg} \frac{2\delta\omega}{\Omega_0^2 - \omega^2} . \qquad (5.1c)$$

Pro budící frekvence rovné frekvenci vlastních tlumených kmitů ($\omega = \Omega$) nabývá závislost $s_0(\omega)$ maxima tj. soustava je *v rezonanci*. Při průchodu přes rezonanční kmitočet se hodnota fáze rychle mění (při nulovém tlumení skokem) o π. Bez tlumení by v rezonanci amplituda kmitů s časem neustále lineárně narůstala tj. byla by dána vztahem

$$s_r = \frac{F_0}{2\Omega_0 m} t \sin \Omega_0 t \qquad (5.1d)$$

5.2 Aplikace Scilabu pro výpočet kmitů soustav s 1° volnosti

Pro ukázku výpočtu kmitající soustavy při harmonickém buzení byl vybrán následující příklad.

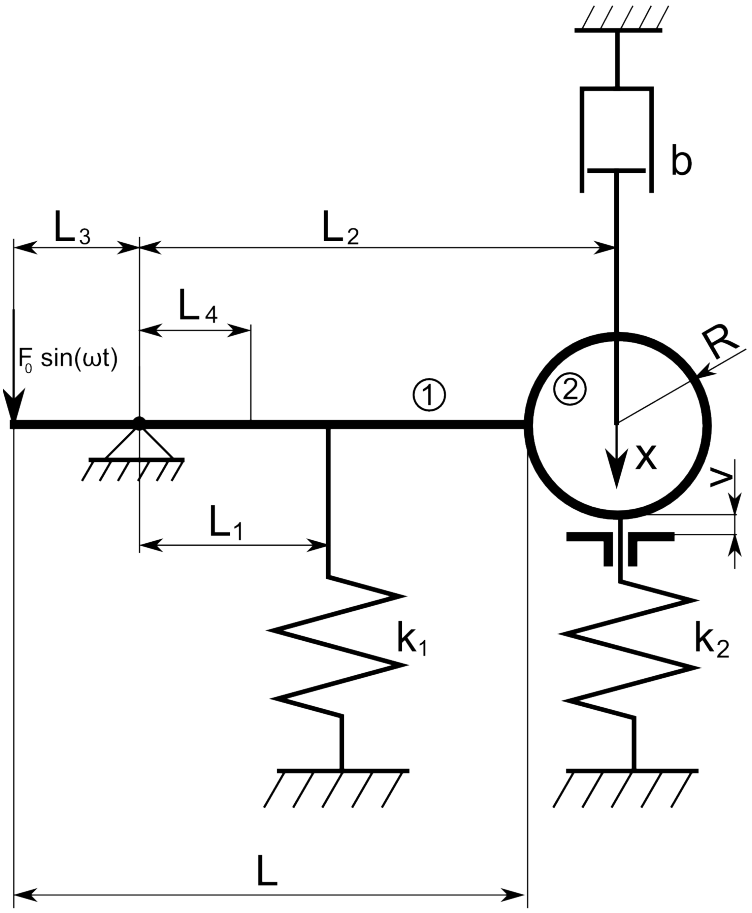
Příklad 3: Je dána soustava těles dle obr.12. Těleso 1 považujte za tyč s konstantní tuhostí, těleso 2 za kotouč. U tyče zanedbejte vlastní deformaci (uvažujte pohyb celku). Na těleso 1 působí na levém okraji tyče časově proměnná síla $F(t) = F_0 \cdot \sin(\omega \cdot t)$ v pásmu frekvence $\omega \in \langle \omega_1 ; \omega_2 \rangle$.

$\Delta\omega \equiv \{10 \le \omega \le 100\} \left[rad \cdot s^{-1}\right]$, $v = 30\text{mm}$, $F_0 = 200\text{N}$, $m_1 = 10\text{kg}$, $m2 = 5\text{kg}$, $L_1 = 0.2\text{m}$, $L_2 = 0.4\text{m}$, $L = 0.6\text{m}$, $L_3 = 0.2\text{m}$, $L_4 = 0.1$, $b = 100\text{Ns}^{-1} m$, $k_1 = 10^4\ Nm^{-1}$, $k_2 = 5 \cdot 10^3\ Nm^{-1}$, soustava je v montážní poloze.

Řešte :

- Sestavte pohybovou rovnici a charakterizujte ji.
- Nakreslete $A - \omega$ charakteristiku.

- Zkontrolujte zda v pásmu provozního buzení nedojde k vymezení vůle.
- Dojde-li k vymezení vůle, navrhněte optimální řešení.
- Proveďte parametrické výpočty umožňující posoudit vliv změn tuhosti, hmotnosti a tlumení na průběh amplitudové frekvenční charakteristiky.



Obr. 12: Schéma studované kmitající soustavy

Pro znázornění amplitudové frekvenční charakteristiky této úlohy lze použít analytické řešení rovnice (5.1a) tj. vztah (5.1b). Tento způsob však je omezen na případ , kdy tuhosti pružin k_i a hodnota tlumení b jsou konstantní. Pokud by tyto veličiny byly závislé na výchylce, tak by bylo nutné nalézt odpovídající jiné analytické řešení rovnice (5.1a) s tím, že pro komplikované závislosti $k_i(x)$ popř. $b(x)$ by jej ani nebylo možné zřejmě nalézt. Pro tyto případy byl proto odladěn algoritmus sledující výchylku kmitů na základě přímého numerického řešení diferenciální rovnice (5.1a) v systému Scilab a určující amplitudovou charakteristiku studovaného systému z ustálené hodnoty amplitudy tohoto numerického řešení. Další výhodou tohoto přístupu je to, že umožňuje zároveň vyhodnotit čas, za který se výchylka vynucených kmitů ustálí, což by mohlo mít význam pro případné experimentální zjišťování amplitudové charakteristiky. Oba přístupy k řešení pro zadaný příklad porovnáme a zhodnotíme.

Pro nalezení pohybové rovnice zadané soustavy použijeme Lagrangeovy rovnice II. druhu :

$$\frac{d}{dt}\left(\frac{\partial E_k}{\partial \dot{x}}\right)-\frac{\partial E_k}{\partial x}+\frac{\partial E_b}{\partial \dot{x}}+\frac{\partial E_p}{\partial x}=\frac{\partial A}{\partial x} \qquad (5.2a)$$

Napíšeme výrazy pro kinetickou energii, energii zatlumené funkce a potenciální energii soustavy :

$$E_k=\frac{1}{2}\left(\frac{1}{12}m_1 l^2+m_1 l_4^2\right)\frac{\dot{x}^2}{l_2^2}+\frac{1}{2}m_2\dot{x}^2+\frac{1}{2}\left(\frac{1}{2}m_2 R^2\right)\frac{\dot{x}^2}{l_2^2} \qquad (5.2b)$$

$$E_b = \frac{1}{2} b \dot{x}^2 \tag{5.2c}$$

$$E_p = \frac{1}{2} k_1 \left(\frac{l_1}{l_2} \right)^2 x^2 + \frac{1}{2} k_2 x^2 - m_1 g \frac{l_4}{l_2} x - m_2 g x \tag{5.2d}$$

Pro jednotlivé členy rovnice pak dostáváme výrazy :

$$\frac{d}{dt} \left(\frac{\partial E_k}{\partial \dot{x}} \right) = M \ddot{x} \tag{5.2e}$$

$$\frac{\partial E_k}{\partial x} = 0 \tag{5.2f}$$

$$\frac{\partial E_b}{\partial \dot{x}} = B \dot{x} \tag{5.2g}$$

$$\frac{\partial E_p}{\partial x} = K x - \overline{m} g x \tag{5.2h}$$

$$\frac{\partial A}{\partial x} = Q_0 \sin(\omega t) \tag{5.2i}$$

Kde :

$$K = k_1 \left(\frac{l_1}{l_2} \right)^2 + k_2 \tag{5.2j}$$

$$B = b \tag{5.2k}$$

$$M = \frac{1}{12} m_1 \left(\frac{l}{l_2} \right)^2 + m_1 \left(\frac{l_4}{l_2} \right)^2 + m_2 + \frac{1}{2} m_2 \left(\frac{R}{l_2} \right)^2 \tag{5.2l}$$

$$Q_0 = F_0 \left(\frac{l_3}{l_2} \right) \tag{5.2m}$$

$$\overline{m} = m_1 \left(\frac{l_4}{l_2} \right) + m_2 \tag{5.2n}$$

Po dosazení do Lagrangeových rovnic II. druhu dostáváme pohybovou rovnici soustavy :

$$M \ddot{x} + B \dot{x} + K x = Q_0 \sin(\omega t) + \overline{m} g \tag{5.2o}$$

U přímé metody vyřešíme diferenciální rovnici numericky. Využijeme k tomu funkci *ode*, pro kterou vyjádříme zrychlení $\ddot{x}$ ze vztahu pro diferenciální rovnici (5.2o).

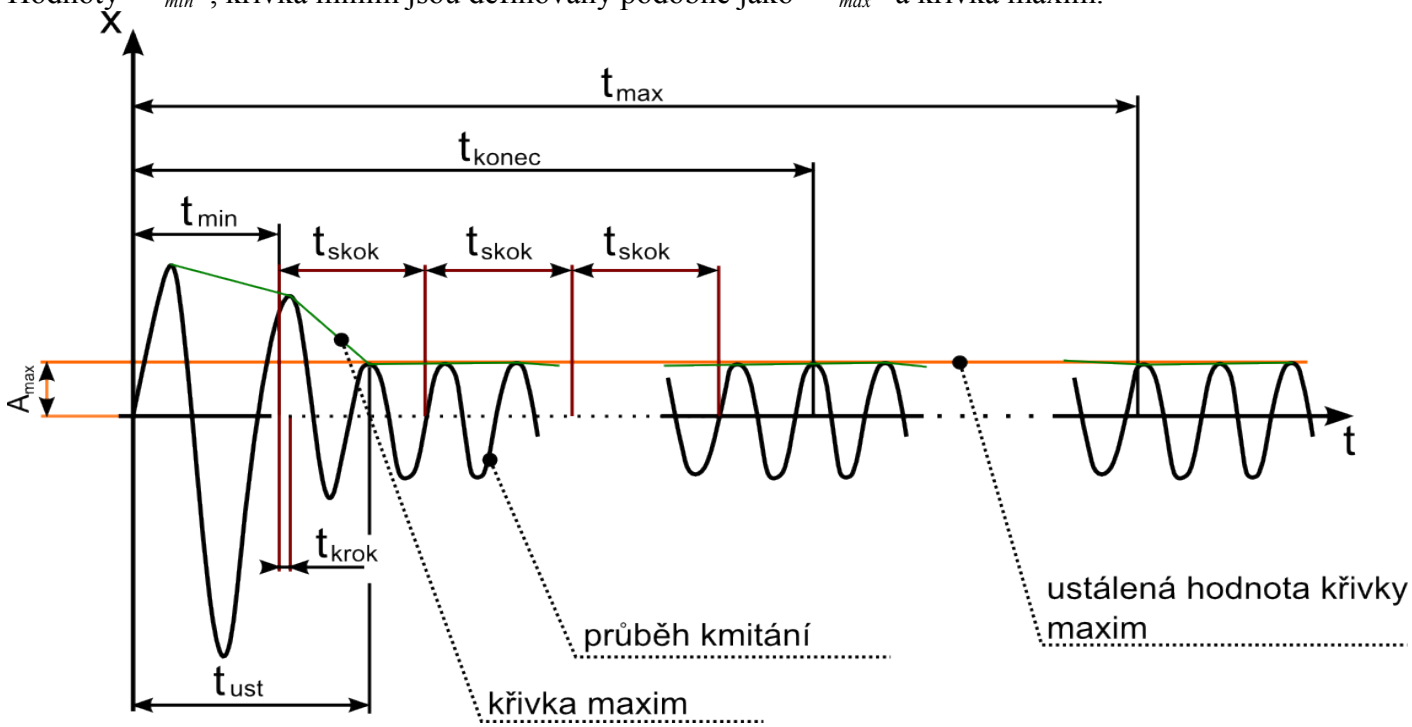
$$\ddot{x} = \frac{Q_0 \sin(\omega t) + \overline{m} g - B \dot{x} - K x}{M} \tag{5.2p}$$

Popis použitých veličin u přímé metody je popsán v tabulce 7 a zobrazen na obr. 13.

označení	popis	označení ve skriptu
t_{min}	Minimální čas, pro který se bude diferenciální rovnice řešit pokaždé.	minimalniCas
t_{konec}	Potřebný čas k nalezení minimálního počtu maxim a minim (n_{min}).	TKONEC
t_{max}	Maximální čas, pro který může skript řešit diferenciální rovnici. Je to z toho důvodu, abychom zabránily případu, kdy zacyklení v programu by dosáhlo takového rozměru, že by Scilab padnul.	maximalniCas
t_{ust}	Hodnota času kdy se amplituda kmitů ustálí. Vyčteme ji z grafu pomocí funkce *Aw*, která zobrazí průběh kmitání pro danou frekvenci.	-
t_{krok}	Časový krok pro řešení diferenciální rovnice.	tkrok
t_{skok}	Délka časového intervalu, ve kterém se řeší studovaná diferenciální rovnice v jednom cyklu. V každém cyklu se hledají maxima , minima a průběžně se zapisují do výsledků do té doby než je splněna podmínka minimálního počtu maxim a minim. Tento interval slouží k urychlení výpočtu úlohy.	tskok
A_{max}	Ustálená hodnota křivky maxim je vypočítaná pomocí váženého průměru jednotlivých maxim. Váhy a hodnoty jednotlivých maxim jsou určeny pomocí funkce *ustalenaHodnota*, která je popsána níže.	AMPMAX
Křivka maxim	Křivka maxim vzniklá spojením lokálních maxim křivky průběhu kmitání.	maxima
n_{min}	Minimální počet maxim a minim nutný k vyhodnocení maximální a minimální výchylky (A_{max} a A_{min}).	minPocExt

Tabulka 7: Popis veličin použitých pro nalezení amplitudy výchylky kmitů na základě přímého řešení pohybové rovnice

Hodnoty A_{min} , křivka minim jsou definovány podobně jako A_{max} a křivka maxim.



Obr. 13: Význam jednotlivých veličin použitých pro nalezení amplitudy výchylky kmitů na základě přímého řešení pohybové rovnice

Napíšeme skript ve vestavěném editoru Scipad, který vypočítá úlohu a zobrazí výsledky.

Opět nastavíme *funcprot(0)*, aby se při opětovném vykonávání skriptu nevypisovalo varování, že byla předefinována již existující funkce. Tento skript už potřebuje delší čas k výpočtu, proto si stopneme jeho délku pomocí funkce *timer()*.

```
funcprot(0);
timer();
```

Nyní zapíšeme zadání do proměnných.

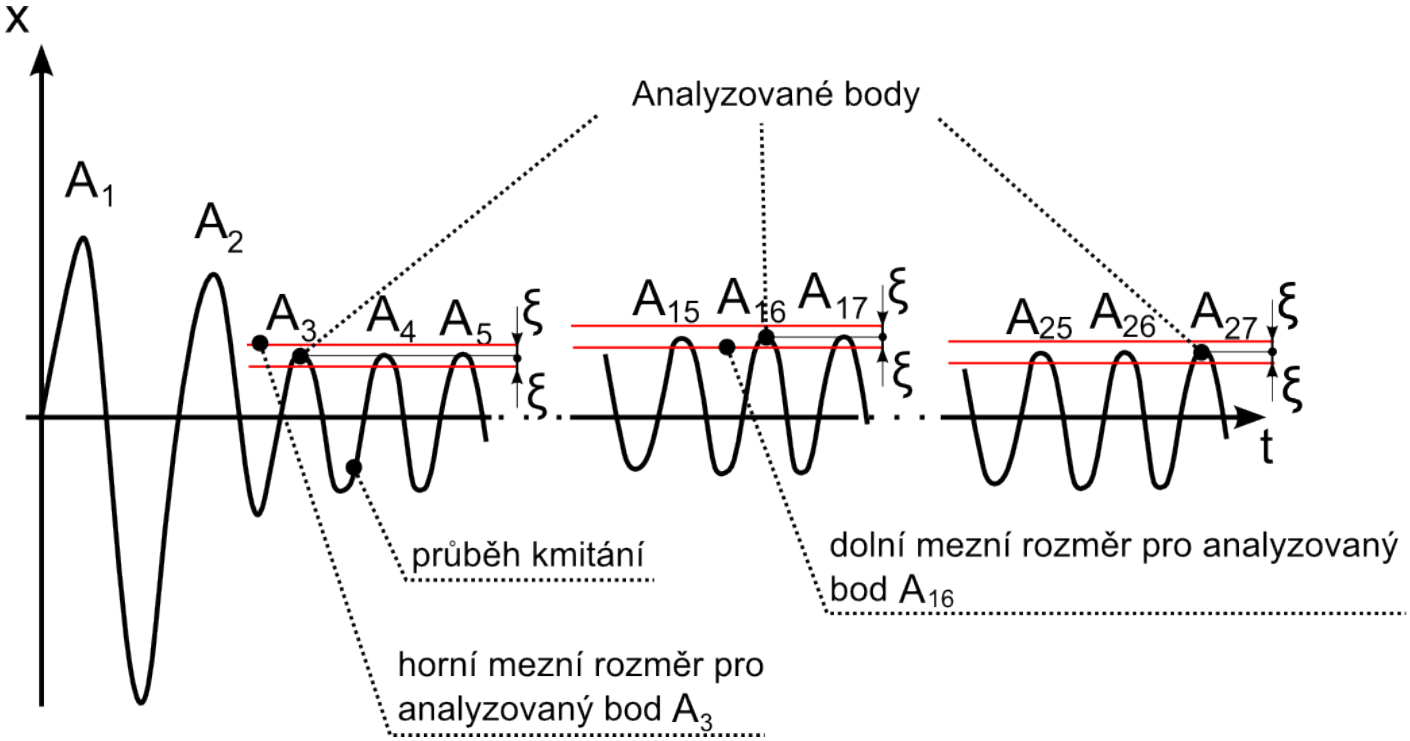
```
v=0.03;
F0=200;
m1=10;
m2=5;
L1=0.2;
L2=0.4;
L=0.6;
L3=0.2;
L4=0.1;
R=0.1;
b=100;
k1=10000;
k2=5000;
g=9.81;
OMGL=10;
OMGR=100;
```

Vytvoříme funkci, která najde lokální maxima a minima, vyjma krajních bodů v definičním oboru. Vstupní parametry funkce jsou vektor funkčních hodnot a vektor, na které jsou tyto hodnoty definovány. Vstupní vektory musí mít minimálně 3 členy, protože při menším počtu nejsme schopni rozhodnout o maximu a minimu. Návratová hodnota funkce je potom 3 dvouřádkové matice (*[ext,maxima,minima]*) obsahující souřadnice minim (*minima*), maxim (*maxima*), všech extrémů (*ext*), přičemž v prvním řádku jsou souřadnice *x* a ve druhém řádku souřadnice y.

```
function [ext,maxima,minima]=najdiExtremy(y,x)
  [m,n]=size(y);
  [o,p]=size(x)
  //  Vektory musí mít minimálně 3 prvky a musí být stejně velké
  if n>=3 & n==p then
    ext=[];
    maxima=[];
    minima=[];
    y_dot=diff(y);
    //  Cyklem projedeme všechny prvky od druhého až po poslední člen
    for i=2:(n-1)
      //  Zapíšeme předchzí a aktuální přírustek
      dy1=y_dot(i-1);
      dy2=y_dot(i);
      //  Pokud je dy1*dy2<0, tak jsme našli extrém
      if (dy2*dy1)<=0 then
        //  Jeli dy1 záporný, tak dy2 je kladný. Našli jsme minimum
        if dy1<0 then
           ext=[ext,[x(i);y(i)]];
           minima=[minima,[x(i);y(i)]];
        //  Pokud je dy1 kladný, tak dy2 je záporný .Našli jsme maximum
        elseif dy1>0 then
           ext=[ext,[x(i);y(i)]];
           maxima=[maxima,[x(i);y(i)]];
        // Pokud je dy1 roven nule, tak jsme našli extrém v [x(i-1),y(i-1)]
        elseif dy1==0 then
          ext=[ext,[x(i-1);y(i-1)]];
```

```
      // Pokud je dy2 roven nule, tak jsme našli extrém v [xi,yi]
      else
        ext=[ext,[x(i);y(i)]];
      end;
    end;
  end;
 else
  disp("Pro nalezení extrému je potřeba minimálně 3 bodů");
 end;
endfunction;
```

Nyní vytvoříme funkci, která najde ustálené hodnoty. Algoritmus postupně analyzuje všechny body křivky tak, že kontroluje sousední body, jestli leží v rámci předem zadané odchylce. Pokud sousední bod leží v dané odchylce od analyzovaného bodu, tak zkontroluje i jeho sousední bod, dokud algoritmus nenarazí na bod, který neleží v dané odchylce. Návratová hodnota je matice o rozměrech *2 x počet členů vstupního vektoru*, kde v první sloupci je údaj o četnosti výskytů bodů (W_i), které leží v odchylce od analyzovaného bodu a ve druhém se nachází hodnota analyzovaného bodu (A_{pi}). Vstupní parametry pro funkci je vektor hodnot (v našem případě vektor maxim a vektor minim) a maximální možná odchylka ξ (ve skriptu *maxChyba*). Jednoduchý obrázek popisující analyzované veličiny ξ , Ap_i jsou zobrazeny na obr.14.



Obr. 14:Analyzování jednotlivých bodů dle funkce ustalenaHodnota

```
function ustaleneHodnoty=ustalenaHodnota(vec,odchylka)

 [m,n]=size(vec);
 //  Zjistíme jeslti má vektor alespoň jeden bod
 if n>=1 then
  ustaleneHodnoty=[];
  //  Projedeme celý vektor a zanalyzujeme každý bod
  for i=1:(n) do

   pocetSouctu=1;

   if i<n then
    //  Od analyzovaného bodu pojedeme doprava a zjistíme kolik vedlejších bodů je v toleranci.
```

```
      //  Pokud nějaký bod v toleranci není tak cyklus přerušíme.
      for j=(i+1):n do

        if abs(vec(j)-vec(i))<odchylka then
          pocetSouctu=pocetSouctu+1;
        else
          break;
       end;
      end;
    end;

    if i>1 then
      //  Tentokrát od analyzovaného bodu pojedeme doleva a zjistíme kolik vedlejších bodů je v toleranci.
      //  Pokud nějaký bod v toleranci není tak cyklus opět přerušíme.
      for j=(i-1):-1:1 do

       if abs(vec(j)-vec(i))<odchylka then
          pocetSouctu=pocetSouctu+1;
       else
         break;
       end;
     end;
    end;
    //  Zapíšeme hodnoty pro analyzovaný bod
    ustaleneHodnoty=[ustaleneHodnoty;pocetSouctu,vec(i)];
  end;
 else
   //  Pokud zadaný vektor má 0 bodů tak v ustálených hodnotách bude v každém sluci false
   ustaleneHodnoty=[%f,%f];
 end;

endfunction;
```

Nyní zadáme příslušné hodnoty konstant vyskytujících se v diferenciální rovnici (5.2o) ze vztahů (5.2j-n).

```
K=k1*(L1/L2)^2+k2;
m=m1*L4/L2+m2;
M=1/12*m1*(L/L2)^2+m1*(L4/L2)^2+m2+1/2*m2*(R/L2)^2;
Q0=F0*L3/L2;
B=b;
```

Pro vlastní definici studované diferenciální rovnice (5.2o) použijeme v systému SCILAB následující zápis (viz. vztah 5.2p).

```
function [xdot]=f(t,x)
  xdot(1)=x(2);
  xdot(2)=x(3);
  xdot(3)=( Q0*sin(omg*t)-B*xdot(2)-K*xdot(1)+m*g ) / M;
endfunction;
```

Zadáme časový krok (t_{krok}) pro řešení diferenciální rovnice a délku časového intervalu (t_{skok}), ve kterém se řeší studovaná diferenciální rovnice v jednom cyklu. Dále definujeme počáteční podmínky diferenciální rovnice a počet řešení diferenciální rovnice v časovém intervalu (t_{skok}) při časovém kroku (t_{krok}).

```
tkrok=0.01;
tskok=0.25;
t0=0;
x0=[0;0;0];
pocIndexu=tskok/tkrok;
```

Nastavíme interval ω , minimální počet lokálních maxim a minim (n_{min}), hodnotu maximální odchylky (ξ) pro funkci *ustalenaHodnota* (viz. blok instrukcí na straně 26,27), maximální čas (t_{max}), pro který se může

diferenciální rovnice řešit a minimální čas (t_{min}), pro který se bude diferenciální soustava pokaždé řešit.

```
omgStep=1;
OMG=OMGL:omgStep:OMGR;
//  Nastavím toleranci pro funkci ustalenaHodnota
maxChyba=2d-4;
//  Nastavím minimální počet maxim a minim
minPocExt=30;
//  Nastavím maximální čas pro který se soustava může řešit
maximalniCas=200;
//  MinimalniCas
minimalniCas=1.5;
```

Nyní definujeme matice, do kterých se budou průběžně ukládat výsledky.

```
AMP=[];
AMPMAX=[];
AMPMIN=[];
X1=[];
X2=[];
X3=[];
EXTTREMY=[];
MAXIMA=[];
MINIMA=[];
DEFMAX=[];
MAXIMA=[];
DEFMIN=[];
MINIMA=[];
DEFEXT=[];
EXTREMY=[];
TKONEC=[];
```

Vytvoříme cyklus *for*, kde pro každou ω zjistíme velikost amplitudy.

```
for omg=OMG do
```

Definujeme proměnné, které budeme používat uvnitř každého cyklu.

```
 pocetMinim=0;
 pocetMaxim=0;
 i=0;
 T0=0;
 extremy=[];
 defmax=[];
 maxima=[];
 defmin=[];
 minima=[];
 defext=[];
 extremy=[];
 XX=[x0];
 T0=0;
```

Bude se provádět cyklus, dokud nebude počet maxim a počet minim větší než námi definovaný minimální počet maxim a minim (n_{min}), nebo čas nebude větší než minimální čas (t_{min}).

```
 while (pocetMaxim<minPocExt | pocetMinim<minPocExt | T0<minimalniCas) do
```

Nastavím okrajové podmínky v daném čase a aktuální index pro zapsání výsledků.

```
  T0=i*tskok;
  T=T0:tkrok:(T0+tskok);
  index=i*pocIndexu+1;
  X0=[XX(1,index);XX(2,index);XX(3,index)];
```

Pokud bude překročen maximální čas, tak se zobrazí graf s průběhem kmitání při dané frekvenci a zastaví se

celý skript.

```
  if (T0+tskok)>maximalniCas then
    plot2d(XX(2,:));
    disp("Skript byl zastaven kvůli překročení maximálního časového úseku");
    abort;
  end;
```

Pomocí funkce *ode* vyřešíme diferenciální rovnici.

```
  xx=ode(X0,T0,T,f);
```

Výsledky následně zapíšeme do průběžných výsledků.

```
  XX(:,index:(index+pocIndexu))=xx(1:3,1:(pocIndexu+1));
```

Pomocí funkce *najdiExtremy* najdeme maxima a minima z vyřešené diferenciální rovnice. Pro případ, že by se maximum nebo minimu nalézalo na levé hranici definičního oboru nebo na pravé hranici definičního oboru předchozího řešení, tak zkontrolujeme aktuální řešení diferenciální rovnice i s posledními 2 body z předchozího řešení diferenciální rovnice.

```
  if i>0 then
    pocatecni_index=index-2;
    T=[T(1)-2*tkrok,T(1)-1*tkrok,T];
  else
    pocatecni_index=1;
  end;

  [%extremy,%maxima,%minima]=najdiExtremy(XX(2,pocatecni_index:(index+pocIndexu)),T);
```

Zapíšeme maxima a minima do průběžných výsledků v daném cyklu *for* a zjistíme jejich celkový počet.

```
  defmax=[defmax,%maxima(1,:)];
  maxima=[maxima,%maxima(2,:)];
  [m,pocetMaxim]=size(maxima);
  defmin=[defmin,%minima(1,:)];
  minima=[minima,%minima(2,:)];
  [m,pocetMinim]=size(minima);
  defext=[defext,%extremy(1,:)];
  extremy=[extremy,%extremy(2,:)];
```

Zvýšíme hodnotu indexu *i* o 1 pro další cyklus *while*.

```
  i=i+1;
end;
```

Uložíme výsledky pro aktuální úhlovou frekvenci.

```
X1=[X1;XX(1,1:(minimalniCas/tkrok+1))];
X2=[X2;XX(2,1:(minimalniCas/tkrok+1))];
X3=[X3;XX(3,1:(minimalniCas/tkrok+1))];
MAXIMA=[MAXIMA;maxima(1:minPocExt)];
MINIMA=[MINIMA;minima(1:minPocExt)];
EXTREMY=[EXTREMY;extremy(1:2*minPocExt)];
DEFMAX=[DEFMAX;defmax(1:minPocExt)];
DEFMIN=[DEFMIN;defmin(1:minPocExt)];
TKONEC=[TKONEC;(i-1)*tskok];
```

Pomocí funkce *ustalenaHodnota* najedeme ustálené hodnoty pro aktuální průběh maxim a minim.

```
ustHodMax=ustalenaHodnota(maxima,maxChyba);
ustHodMin=ustalenaHodnota(minima,maxChyba);
```

Teď vypočítáme ustálenou hodnotu křivky maxim a minim z ustálených hodnot jako vážený průměr, kde četnost výskytu bodů (W_i) jsou váhy a jednotlivé hodnoty bodů (A_{pi}) jsou hodnoty k těmto vahám.

$$A_{max}=\frac{\sum_{i=1}^{n} A_{pi}\cdot W_i}{\sum_{i=1}^{n} W_i} \qquad (5.2q)$$

```
amplitudaMax=sum(ustHodMax(:,2).*ustHodMax(:,1))/sum(ustHodMax(:,1));
amplitudaMin=sum(ustHodMin(:,2).*ustHodMin(:,1))/sum(ustHodMin(:,1));
```

Uložíme výsledky do proměnných. Výslednou amplitudu vypočítáme jako $\frac{A_{max}-A_{min}}{2}$.

```
AMP=[AMP,(amplitudaMax-amplitudaMin)/2];
AMPMAX=[AMPMAX,amplitudaMax];
AMPMIN=[AMPMIN,amplitudaMin];

end;
```

Pro zjištění amplitudové frekvenční charakteristiky je také možné použít analytické řešení pohybové rovnice pro ustálené kmity . V tomto případě převedeme pohybovou rovnici naší soustavy (5.2o) na normovaný tvar (5.1a) a aplikujeme vzorec (5.1b), ve kterém $\delta=\frac{B}{2M}$, $\Omega_0=\sqrt{\frac{K}{M}}$, $F_0=Q_0$, kde K,B,M,Q_0 jsou výrazy ze vztahů (5.2j-m) a závislost amplitudy s_0 na frekvenci budící síly je dána vztahem

$$s_0=\frac{Q_0}{M\sqrt{\left(\frac{K}{M}-\omega^2\right)^2+\left(\frac{B\omega}{M}\right)^2}} \qquad (5.2r)$$

Pro účely grafického znázornění tento výraz naprogramujeme pomocí instrukcí Scilabu.

```
#K=k1*(L1/L2)^2+k2;
#M=1/12*m1*(L/L2)^2+m1*(L4/L2)^2+m2+1/2*m2*(R/L2)^2;
#Q0=F0*L3/L2;
#B=b;

s0=#Q0./(#M*((#K/#M-OMG^2)^2+(#B*OMG./(#M ))^2)^0.5);
```

Zobrazíme analytické a numerické řešení amplitudové frekvenční charakteristiky v intervalu ω.

```
plot2d(OMG,s0);
plot2d(OMG,AMP,style=5);
```

Nakonec ještě vypočítáme průměrnou odchylku amplitudy určené přímým numerickým řešením diferenciální rovnice od amplitudy určené analyticky podle vzorce pro průměr :

$$\bar{x}=\frac{\sum_{i=1}^{n} |A_i-X_i|}{n} \qquad (5.2s)$$

kde A_i jsou hodnoty amplitud pro jednotlivé frekvence ω_i vycházející z přímé metody a X_i jsou hodnoty amplitud vypočítané pomocí analytického řešení ustálených kmitů, n je počet hodnot ω_i .

```
prumernaOdchylka=sum(abs(AMP-s0))/((OMGR-OMGL)/omgStep)
```

Na závěr ještě definujeme funkci, která zobrazí průběh kmitání pro danou frekvenci, maxima, minima a ustálenou hodnotu maxim a minim.

```
function Aw(omg)
  t=0:tkrok:minimalniCas;
  plot2d(t,X2(omg-OMGL+1,:),style=2);
  plot2d(DEFMAX(omg-OMGL+1,:),MAXIMA(omg-OMGL+1,:),style=3);
  plot2d(DEFMIN(omg-OMGL+1,:),MINIMA(omg-OMGL+1,:),style=3);
```

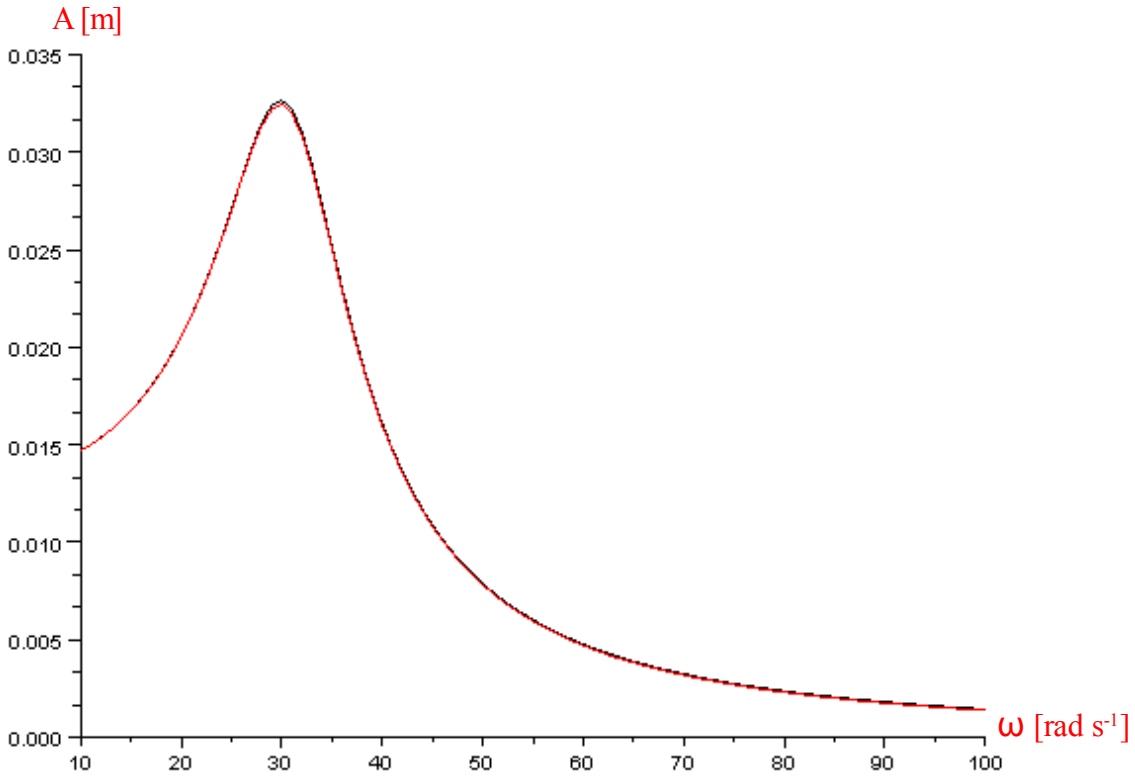
```
  plot2d([0 max(DEFMIN(omg-OMGL+1,:))],[AMPMIN((omg-OMGL)/omgStep+1) AMPMIN((omg-
OMGL)/omgStep+1)],style=5);
  plot2d([0 max(DEFMAX(omg-OMGL+1,:))],[AMPMAX(omg-OMGL+1) AMPMAX(omg-OMGL+1)],style=5);
endfunction;
```

Poznámka: Opět je nutné při vkládání do Scilabu, vložit každou instrukci na jeden řádek viz příklad 1.

Tady skript končí, proto si zobrazíme celkový čas vykonání skriptu.

```
celkovyCas=timer()
```

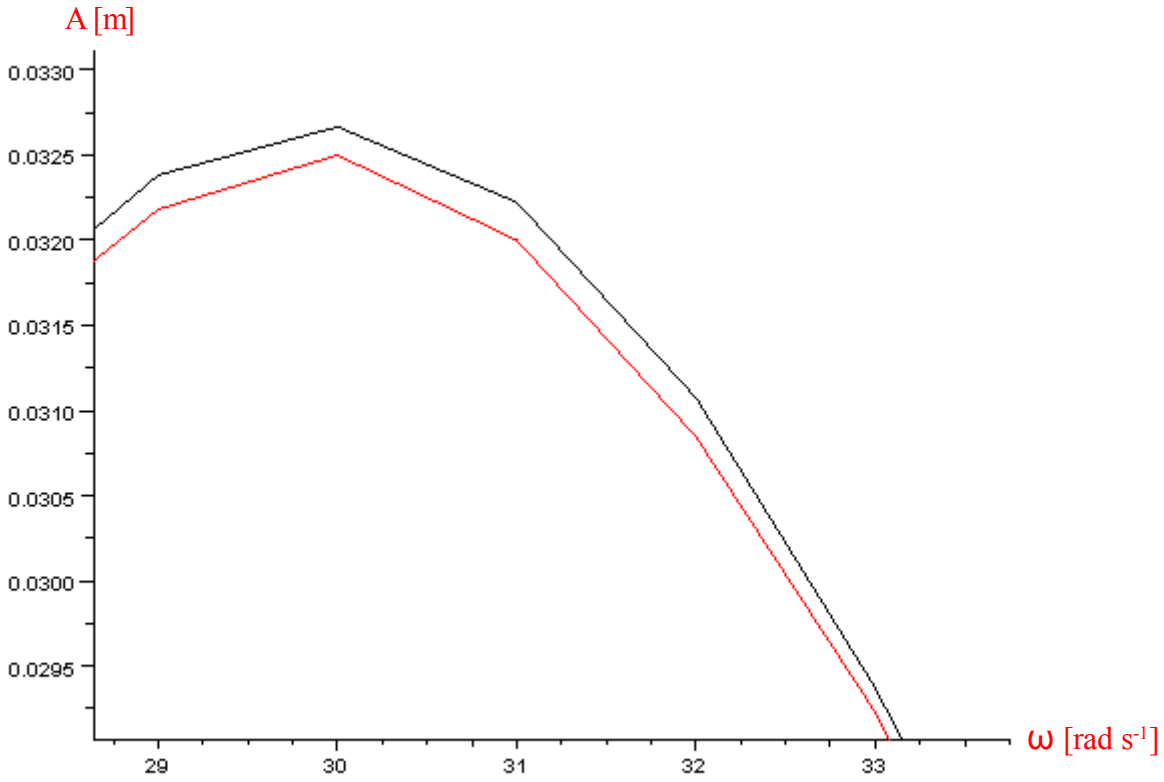
Nyní skript spustíme. Po vypočítání vidíme výsledný graf amplitudové frekvenční charakteristiky (obr. 15) a v příkazovém řádku (pod obr.15) se zobrazí výpočetní čas s průměrnou odchylkou od analytického řešení.



Obr. 15: Amplitudo frekvenční charakteristika

```
prumernaOdchylka  =

   0.0000743
celkovyCas  =

   18.484375
```

Na grafu si můžeme všimnout, že jak analytické řešení, tak i numerické řešení se překrývají. Po větším přiblížení vidíme (obr.16), že je mezi nimi velice malá odchylka, která zřejmě odpovídá numerickým chybám.

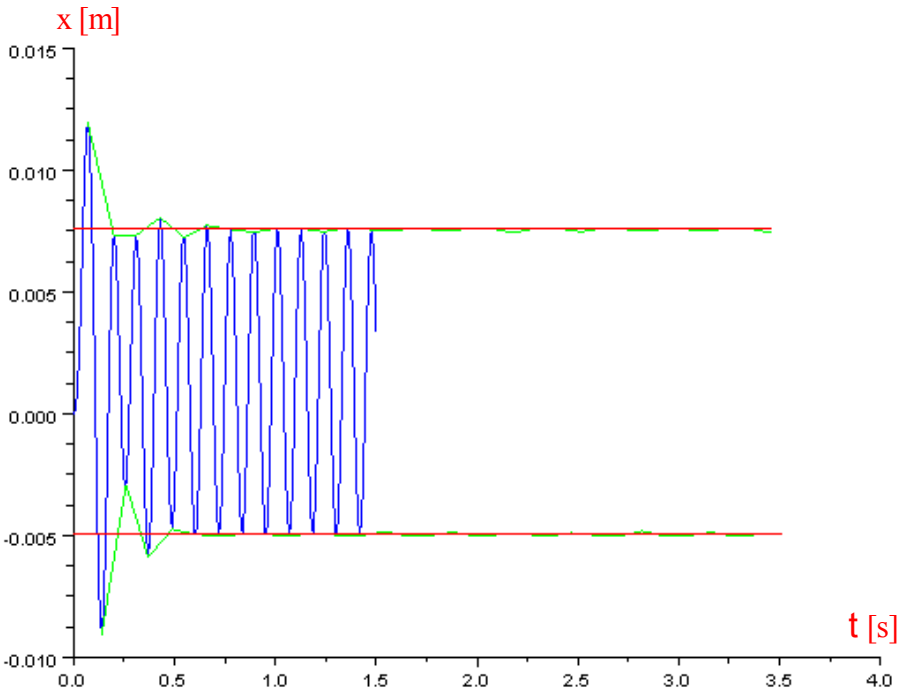


Obr. 16:Detail amplitudové frekvenční charakteristiky vypočítané přímou metodou a analytickým vztahem

Na detailu grafu můžeme vidět, že výsledek se liší v deseti tisících metru, tedy v desetinách milimetru. Po odměření z grafu je vidět, že odchylka v těchto bodech je asi 0,16 mm. Vypočítaná průměrná odchylka od analytického řešení tohoto příkladu je 0,0000743m tedy 0,0743 mm. Pomocí funkce *Aw* si zobrazíme jakýkoliv průběh kmitání, abychom mohli porovnat přesnost tohoto výpočtu. Zobrazíme průběh kmitání při budící frekvenci $\omega=54\ rad\ s^{-1}$.

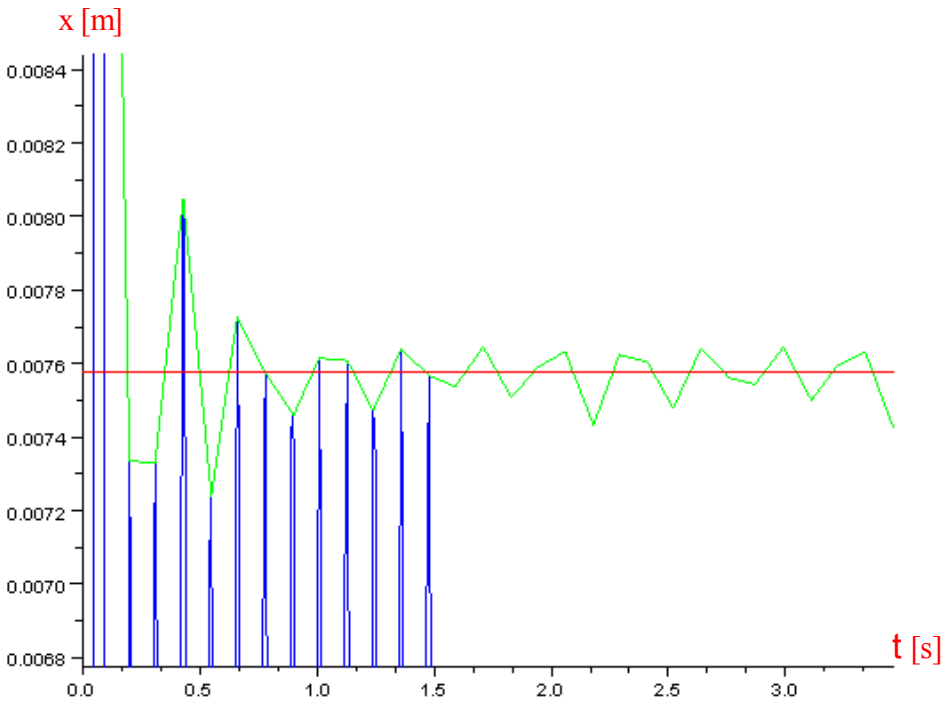
```
- - > Aw(54)
```

Na následujícím grafu (obr. 17) vidíme průběh kmitání, který je definovaný pro námi zvolený minimální čas (t_{min}). Jsou zde také vidět křivky, které spojují maxima (křivka maxim) a minima (křivka minim) a konečně taky dvě přímky, které představují ustálenou hodnotu maxim (A_{max}) a ustálenou hodnotu minim (A_{min}).



Obr. 17: Průběh kmitání při budící frekvenci ω=54 rad s^{-1}

Při zobrazení detailního průběhu kmitání (viz. Obr.18) je zřejmé, že ustálená hodnota maxim (A_{max}) se shoduje s maximální výchylkou, kolem které křivka maxim mírně osciluje po ustálení.



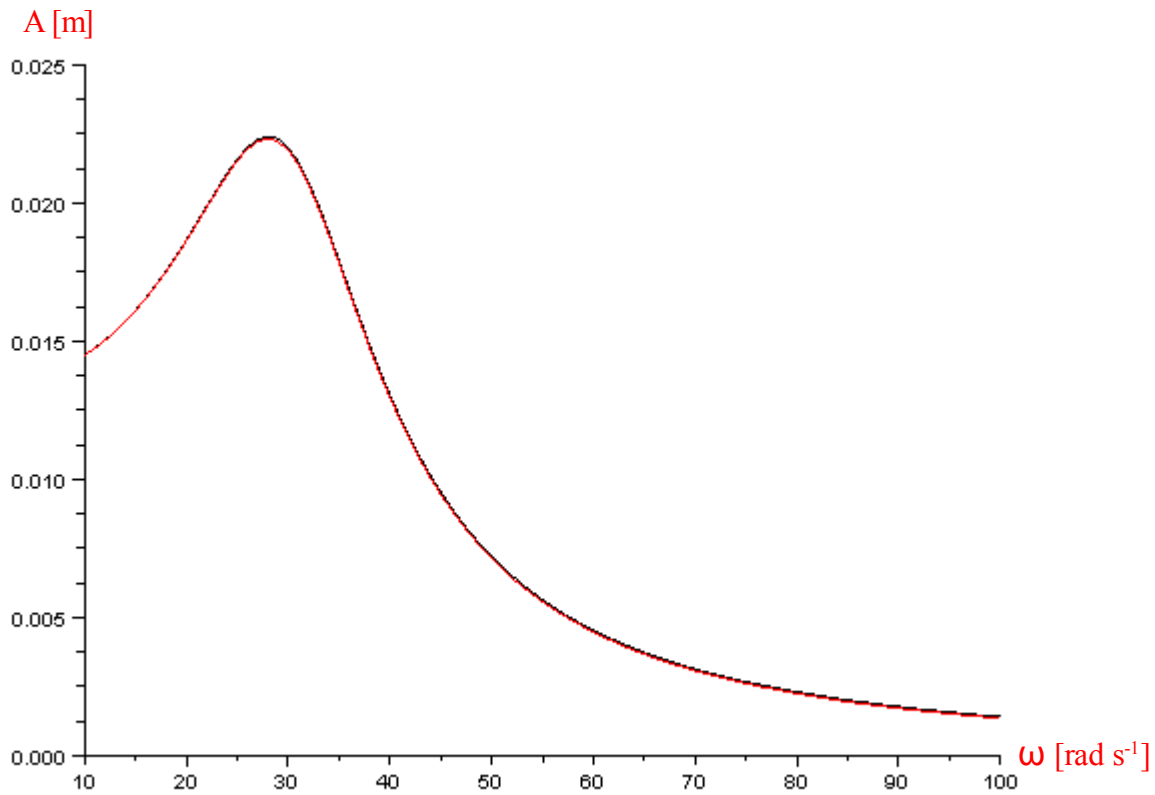
Obr. 18: Detail průběhu kmitání při budící frekvenci ω=54rad s^{-1}

Provedené testovací výpočty závislosti kmitů na čase pro jednotlivé frekvence ukázaly, že vždy dochází k ustálení výchylek kolem středních hodnot A_{max} a A_{min}. Záleží samozřejmě na vstupních parametrech. Tedy jaký zvolíme časový krok (t_{krok}), minimální počet maxim a minim (n_{min}) a maximální odchylku (ξ).

Hodnotu maximálního vychýlení dostaneme pomocí příkazu

```
- - >max(AMPMAX)
 ans  =

    0.0337903
```

Výchylka nám vyšla 3,38 cm , takže vymezení vůle nastane. Aby vůle vymezena nebyla, tak je nutné změnit některý z parametrů soustavy, např. zvýšit hodnotu tlumení. Výsledná amplitudová frekvenční charakteristika, po zvýšení hodnoty tlumení ze b= 100 Ns^{-1} na b_1 =150 Ns^{-1} je na obr. 19.



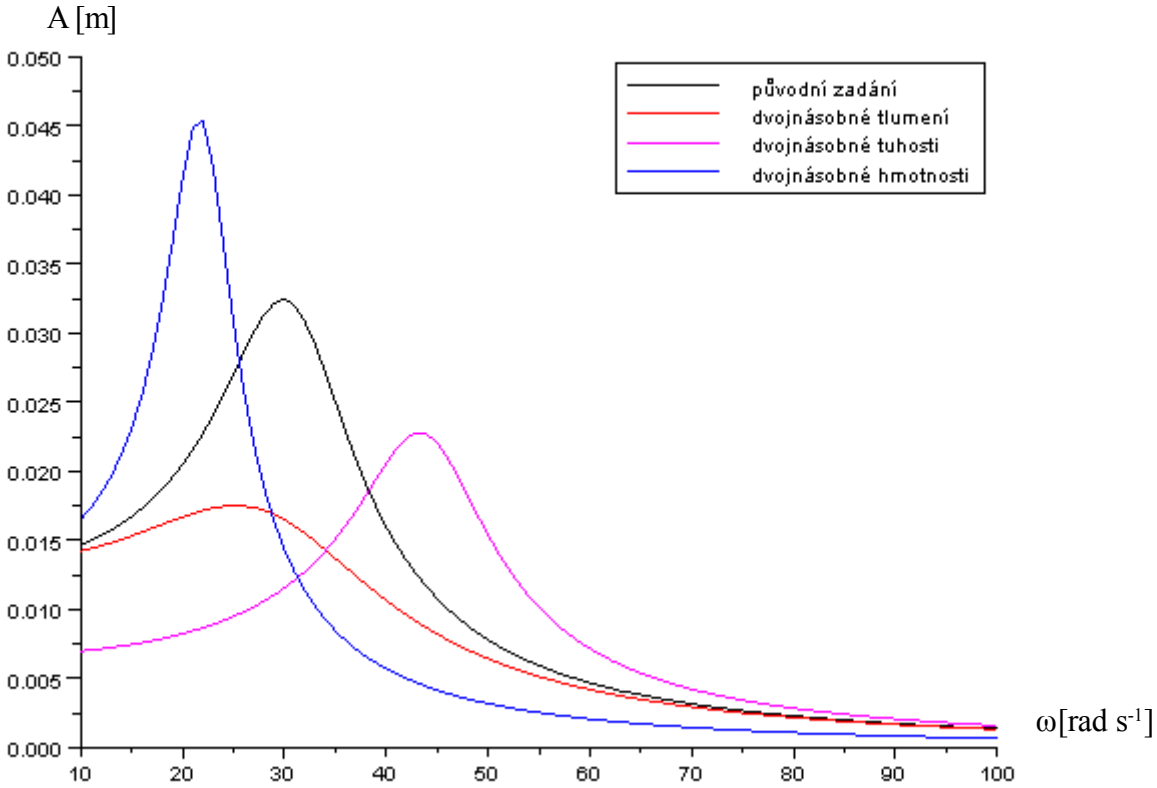
Obr. 19: Amplitudová frekvenční charakteristika při zvýšeném tlumení

Nyní opět určíme maximální výchylku.

```
- - >max(AMPMAX)
 ans  =

    0.0236503
```

Při zvýšeném tlumení ze 100 Ns^{-1} na 150 Ns^{-1} tedy vychází maximální výchylka 2,37 cm , takže k vymezení vůle by v tomto případě nedošlo.

Pro analýzu chování studované kmitající soustavy provedeme parametrické výpočty při různých hodnotách tlumení b, tuhosti k_i pružin a hmotnosti m_i nastaveních. Jednou zvolíme dvojnásobně velké hmotnosti členů, podruhé dvojnásobně velké tuhosti pružin, potřetí dvojnásobně velké tlumení a výsledek porovnáme s amplitudovou frekvenční charakteristikou dle původního zadaní. Výsledky těchto parametrických výpočtů jsou na obr. 20.



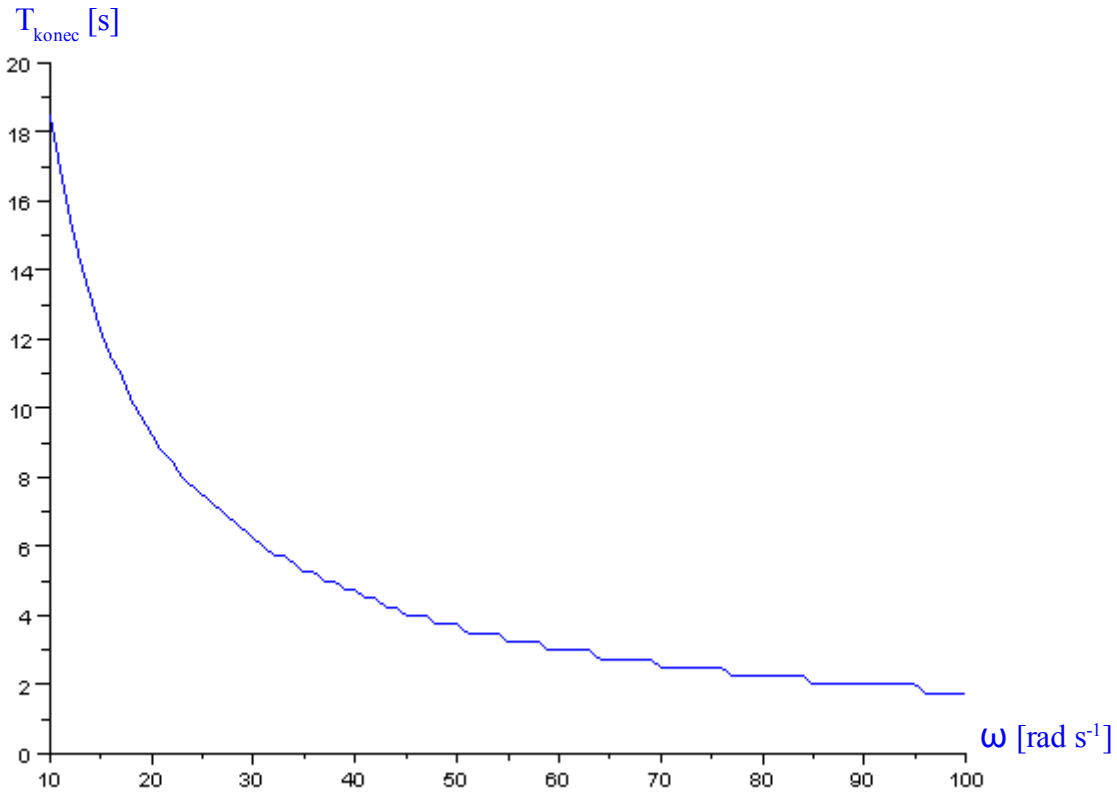
Obr. 20: Vliv změny parametrů na amplitudovou charakteristiku

Jedna z nevýhod tohoto algoritmu je, že pro výpočet amplitudové frekvenční charakteristiky při nízké budící frekvenci je třeba nastavit velký maximální čas (t_{max}), pro který se řeší diferenciální rovnice. Na druhou stranu při nízkých frekvencích je výpočet natolik přesný, že i po mnohonásobném zvětšení grafu amplitudové frekvenční charakteristiky se přímé řešení překrývá s analytickým řešením. Při vyšších budících frekvencích je výpočet méně přesný, ale za to probíhá daleko rychleji. Tento efekt je způsoben především nastavením časového kroku (t_{krok}) a minimálního počtu maxim a minim (n_{min}). Tuto nevýhodu je možné odstranit tím, že rozsah budící frekvence rozdělíme na několik částí a pro každou část nastavíme v algoritmu jiné časové kroky a minimální počet maxim a minim. Přesnost výpočtu je možné ověřit tím, že příklad vypočítáme opětovně s různými parametry výpočtu. Jednou nastavíme určitý časový krok, minimální počet maxim a minim a podruhé tyto hodnoty vhodně změníme a porovnáme obě křivky výsledné amplitudové frekvenční charakteristiky. Pokud by se výrazně nelišily s ohledem na nastavení zmíněných parametrů výpočtu, tak můžeme říct, že výpočet je přesný. V našem případě jsem zkusil výpočet provést při časovém kroku desetkrát menším 0,001s oproti původním 0,01s. Průměrná odchylka od analytického řešení se změnila z 0,0743mm na 0,0043mm a čas výpočtu prodloužil z 18.5s na 31,7s (konfigurace počítač : *AMD Athlon(tm) 64 Processor 3000+ 1,86 Ghz, 896 MB RAM, Systém MS Windows XP*). Obě křivky se vzájemně překrývaly.

Průběh potřebného času (t_{konec}) pro nalezení minimálního počtu maxim a minim pro jednotlivé frekvence je

na obr. 21.

```
- - >plot2d(OMG,TKONEC,style=2)
```



Obr. 21: Vliv frekvence na dobu výpočtu (t_{konec}) potřebnou k nalezení minimálního počtu maxim a minim

Z obrázku jde vidět jak je křivka při počátečních frekvencích hladká a postupně se začíná měnit skokově. Je to způsobeno nastavením časového intervalu t_{skok} , který slouží k urychlení výpočtu v reálném čase. Kdyby byla hodnota časového intervalu t_{skok} menší, křivka by byla i při vyšších frekvencích hladká. Tato křivka může nabývat minimální hodnotu t_{min}

6. Závěr

Tato práce byla zaměřena na možnosti využití matematických softwarů pro výpočet příkladů z kinematiky a dynamiky. Pro tyto účely byl popsán program Scilab, který jsem zvolil z mnoha důvodů. Jednak je zdarma a taky se mi líbí uživatelské prostředí, které je jednoduché a přehledné. S žádným jiným matematickým softwarem nemám takové zkušenosti právě jako ze Scilabem. Jeho velkou předností je, že je neustále vyvíjen. Toto tvrzení potvrzuje i dotační program z Evropské Unie, která má zřejmě zájem na dalším vývoji tohoto matematického systému. Jelikož se samostatně věnuji programování internetových stránek v PHP a absolvoval jsem předměty jako základy informatiky nebo numerické metody (které byly zaměřeny na systém Maple), tak přechod na Scilab mi nedělal žádné větší potíže. Hlavně v začátcích mne práce se Scilabem hodně bavila. V průběhu semestru, kdy jsem pracoval na této práci, jsem prodiskutoval s ostatními kamarády i z jiných fakult VUT rozdíly mezi jednotlivými programy pro numerické výpočty. Zněl názor, že Matlab je

kvalitnější než Scilab. O tomto tvrzení nelze pochybovat. Na druhou stranu Scilab je zdarma a to je jeho největší výhoda. Navíc je možnost propojit Scilab s jinými profesionálními komerčními nástroji jako LabVIEW a řada jiných.

Pro demonstraci byli vybrány 3 ukázkové příklady. První 2 příklady (čtyřkloubový mechanismus, nelineární převod otevírání okna) byli z kinematiky. V obou dvou případech bylo známo analytické řešení, i když u čtyřkloubého mechanismu se k němu docházelo poměrně složitým způsobem. Na čtyřkloubovém mechanismu byla aplikována vektorová metoda, která je universální metodou pro řešení kinematických mechanismů. Z důvodu složitosti analytického řešení, vycházejícího z vektorové metody, jsem předvedl řešení pomocí soustavy nelineárních rovnic, která byla vyřešena prostou iterací. Je to universální metoda, která lze aplikovat na složité kinematické mechanismy. U příkladu, kde se otevíralo okno, jsem provedl jednoduchý výpočet, který vycházel z analytického řešení.

Třetí úloha byla z dynamiky a zabývala se analýzou kmitů soustavy s jedním stupněm volnosti. Pro tuto úlohu byla vypočítána a znázorněna amplitudová frekvenční charakteristika jak použitím vztahů pro analytické řešení příslušné pohybové rovnice tak i na základě přímého numerického řešení diferenciální rovnice kmitu. Numerické řešení bylo založeno na sledování časového průběhu kmitů pro různé budící frekvence, amplitudová charakteristika byla určována z početně vyhodnocené ustálené výchylky kmitu. Pro analyzovanou mechanickou soustavu oba způsoby určení amplitudové charakteristiky se prakticky nelišily s ohledem na nastavené parametry. Výhodou předloženého algoritmu přímé metody jsou zřejmě následující: 1) Umožňuje sledovat chování výchylky kmitů i v oblastech, kdy kmity ještě nejsou ustáleny ; 2) Umožňuje určit čas, kdy se hodnota kmitů ustálí, což může mít význam pro experimentální měření; 3) Je pravděpodobné že uvedené numerické řešení by mohlo být používáno i pro nelineární soustavy (nelineární pružiny, tlumiče s hodnotou tlumení závisející na poloze apod.). Tento problém však již značně přesahoval rámec zadání bakalářské práce, rád bych se k němu možná vrátil v rámci práce diplomové.

7. Seznam použitých zdrojů

[1] *GNU Octave - Wikipedia* [online]. 5.9.2009 [cit. 2009-04-22]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/GNU_Octave>.

[2] *Octave* [online]. [cit. 22-4-2009].Dostupný z WWW : <http://www.gnu.org/software/octave/>.

[3] *MATLAB - Wikipedia* [online]. [cit. 2009-04-22]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/MATLAB>.

[4] *Mathematica - Wikipedia* [online]. [cit. 2009-04-22]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Mathematica>.

[5] *Oficiální stránky programu wxMaxima* [online]. [cit. 22-4-2009]. Dostupný z WWW: <http://wxmaxima.sourceforge.net/wiki/index.php/Main_Page>

[6] *Maxima - Wikipedia* [online]. [cit. 2009-04-22]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Maxima>.

[7] *Oficiální stránky programu Yacas* [online]. [cit. 22-4-2009]. Dostupný z WWW: <http://yacas.sourceforge.net/homepage.html>.

[18] Yacas - *Wikipedia* [online]. [cit. 2009-04-22]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Yacas>.

[19] DŘÍMALA, Josef. *Aplikace systému maple při řešení úloh kinematiky a dynamiky*. [s.l.], 2008. 30 s. Vedoucí bakalářské práce doc. RNDr. Karel Pellant, CSc.

[10] Scilab Home Page [online]. Dostupný z WWW: <http://www.scilab.org/> [cit. 2009-4-12].

[11] VACULÍKOVÁ, Martina. *Elektronický manuál pro program SCILAB*. [s.l.], 2006. 75 s. Univerzita Tomáše Bati ve Zlíně, Fakulta aplikované informatiky . Bakalářská práce.

[12] *Modeling and Simulation in Scilab/Scicos*. CAMPBELL Stephen, CHANCELIER Jean-Philippe, NIKOUKHAH Ramine. 2006, ISBN: 978-0-387-27802-5

[13] *Kinematika*. Autor: PŘIKRYL Karel, ISBN 80-214-2931-8, červen 2005

[14] *Technická mechanika II* [online]. [cit. 2009-04-22]. Autor: PELANT Karel.Dostupný z WWW : <http://www.umt.fme.vutbr.cz/~kpellant/>.

8. Seznam obrázků

9. Seznam tabulek

10.Seznam použitých zkratek a symbolů

$\varphi \quad [rad]$	Úhlové natočení
$\dot{\varphi}$, $\omega \quad [rad\,s^{-1}]$	Úhlová rychlost
$\ddot{\varphi}$, $\alpha \quad [rad\,s^{-2}]$	Úhlové zrychlení
$s \quad [m]$	Dráha
$v \quad [m\,s^{-1}]$	Rychlost
$a \quad [m\,s^{-2}]$	Zrychlení
$t \quad [s]$	Čas
L , $l \quad [m]$	Délka
$F \quad [N]$	Síla
$E_k \quad [J]$	Kinetická energie
$E_p \quad [J]$	Potenciální energie
$E_b \quad [J]$	Energie zatlumené funkce
$m \quad [kg]$	Hmotnost
$g \quad [m\,s^{-2}]$	Gravitační zrychlení
K , $k \quad [N\,m^{-1}]$	Tuhost
B , $b \quad \left[\frac{N}{m\,s^{-1}}\right]$	Součinitel tlumení
$Q_0 \quad [N]$	Amplituda zobecněné síly
$A \quad [m]$	Amplituda výchylky
$x \quad [m]$	Výchylka
x_p	Partikulární řešení pohybové rovnice
x_h	Homogenní řešení pohybové rovnice
s_0	Amplituda vynucených kmitů
δ	Součinitel doznívání
Ω_0	Vlastní frekvence netlumených kmitů
Ω	Vlastní frekvence tlumených kmitů
ψ	Zdvihová závislost
μ	Převodová funkce
ν	Derivace převodu

11. Seznam příloh

Přílohy jsou na přiloženém CD obsahující soubory :

- ctyrkloubovy_mechanismus.sci *(skript pro výpočet čtyřkloubového mechanismu - příklad 1)*
- okno.sci *(skript pro výpočet otevírání okna – příklad 2)*
- kmitani.sci *(skript pro výpočet kmitající soustavy - příklad 3)*
- scilab-5.1.1.exe *(instalační soubor Scilabu 5.1.1 pro Win32)*
- BakalarskaPrace.pdf *(vlastní soubor vypracované bakalářské práce)*