

Mechatronika algoritmy, modelování a simulace

Robert Grepl

...pracovní text...

Dobrý večer milé děti.

Ačkoli tento text vzniká s velkými i malými přestávkami již více než deset let, je stále převělice pracovní, buděte prosím nadmíru tolerantní.

Připomínky, nápady, chyby pište na: robert.grepl@mechlab.cz.

Díky.

Text *opravdu není* určen k veřejnému šíření (takže ho na ty internety nedáváte).

Díky.

RG

Release notes:

- *2018-02-07*: První update v r. 2018. Alfahotové kap. [Rovnice 2](#) a [ODE 3](#). Kap. [LR2 5](#), [DC motor 4](#) a [KF 7](#) nějak rozpracovány. 101 str.
- *2018-02-19*: Nová obálka. Přidaná kap. [StateSpace 6](#), ale nedosochaná (je to remake verze z r. 2007). 107 str.
- *2018-05-14*: Příklady FFC anténa ([9.4](#)), Pohon [9.3](#). Zárodeek kap. [Dynam.syst. 1](#), [Ident. syst. 8](#), [FFC 9](#) a [Přehled modelů 10](#). Text v kap. [9](#). Drobnosti v kap. [10](#). Literatura (zárodeek). 125 str.

1	Lineární dynamický systém	5
2	Řešení m rovnic o n neznámých	7
3	Numerické řešení ODE	36
4	Model DC motoru	71
5	Lagrangeovy rovnice 2. druhu (LR2)	74
6	Stavové řízení	83
7	Kalmanův filtr	89
8	Identifikace systémů	109
9	Feedforward control aneb přímovazebné řízení	111
10	Přehled modelů systémů	119
	Použitá a doporučená literatura	123

1	Lineární dynamický systém	5
1.1	Lineární DS se spojitým časem	5
1.2	Systém prvního řádu	5
1.3	Systém druhého řádu	5
2	Řešení m rovnic o n neznámých	7
2.1	Intro: $m = n = 1$	9
2.2	Lineární rovnice ($m = n > 1$)	12
2.3	Lineární rovnice ($m > n$) aneb obyčejné nejmenší čtverce (OLS)	15
2.4	Nelineární rovnice ($m = n > 1$)	19
2.5	Nelineární rovnice ($m > n$) aneb nelineární nejmenší čtverce (NLS)	26
2.6	Málo rovnic aneb $m < n$	32
2.7	Otzázky a neřešené úlohy	35
3	Numerické řešení ODE	36
3.1	Co je vlastně řešením ODE?	37
3.2	Maticová exponenciála – řešení lineární ODE	39
3.3	Diskretizace – řešení lineární ODE	42
3.4	Metody numerické integrace	47
3.5	Řešiče (solvery) ODE v MATLABu a Simulinku	52
3.6	Příklady řešení ODE v MATLABu a Simulinku	62
3.7	Otzázky a neřešené úlohy	70
4	Model DC motoru	71
4.1	Dynamický model DC motoru	71
4.2	Statický model elektrického subsystému	72
4.3	Statická charakteristika (celého) DC motoru	72
5	Lagrangeovy rovnice 2. druhu (LR2)	74
5.1	Úvod	74
5.2	Příklad na LR2: Mostový jeřáb	76
5.3	Příklad na LR2: Dvojité kyvadlo	79
5.4	Sestavení LR2 s pomocí nástrojů pro symbolické výpočty	81
6	Stavové řízení	83
6.1	Úvod	83
6.2	Úvod	84
6.3	Jak získat matici K	86
6.4	Stavový pozorovatel	86
6.5	Otzázky a neřešené úlohy	88
7	Kalmanův filtr	89
7.1	Úvod	90
7.2	Lineární KF	90
7.3	Extended KF (EKF)	91
7.4	Detaily, příklady a odvození KF	92
7.5	Otzázky a neřešené úlohy	108
8	Identifikace systémů	109
8.1	Statické modely	109
8.2	Otzázky a neřešené úlohy	109

9	Feedforward control aneb přímovazebné řízení	111
9.1	Lyrický úvod	111
9.2	Schéma FFC a inverzní dynamika	112
9.3	Příklad: Rychlostní řízení 1dof	114
9.4	Příklad: Stabilizace antény 1dof	115
9.5	Příklad: Řízení mostového jeřábu	117
9.6	Otzázkы a neřešené úlohy	117
10	Přehled modelů systémů	119
10.1	Systém prvního rádu	119
10.2	Hmota s pružinou a tlumičem	119
10.3	Matematické kyvadlo	119
10.4	Mostový jeřáb	119
10.5	DC motor	120
10.6	Dvě hmoty s pružinou a tlumičem (fixované k rámu)	121
10.7	Dvě hmoty s pružinou a tlumičem (volné)	121
10.8	Dvojité kyvadlo	121
	Použitá a doporučená literatura	123

1 Lineární dynamický systém

Obsah kapitoly:

1.1	Lineární DS se spojitým časem	5
1.2	Systém prvního řádu	5
1.3	Systém druhého řádu	5
1.3.1	Standardní tvary přenosové funkce	5
1.3.2	Parametry a charakteristiky	6

1.1 Lineární DS se spojitým časem

1.2 Systém prvního řádu

Mnoho reálných systémů se chová přibližně jako systém prvního řádu.

Popis:

$$G(s) = \frac{1}{\tau s + 1} \quad (1.1)$$

$$\tau \dot{q} + q = ku \quad (1.2)$$

Analytické řešení pro skok na vstupu u :

$$q(t) = k(1 - e^{-\tau t}) \quad (1.3)$$

1.3 Systém druhého řádu

Z mechaniky (dynamiky) známe zápis:

$$m\ddot{x} + b\dot{x} + kx = F(t), \quad (1.4)$$

případně tvar

$$\ddot{x} + 2\delta\dot{x} + \Omega^2x = \frac{1}{m}F(t), \quad (1.5)$$

kde $\Omega = \sqrt{\frac{k}{m}}$ je vlastní frekvence a $\delta = \frac{b}{2m}$ je tlumení.

1.3.1 Standardní tvary přenosové funkce

Pro zápis přenosu systému druhého řádu se používá několik standardních tvarů.
Přímý přepis rovnice 1.4 vede na přenos

$$G(s) = \frac{1}{ms^2 + bs + k}. \quad (1.6)$$

Přepis rovnice 1.5 vede na tvar

$$G(s) = \frac{1/m}{s^2 + 2\delta s + \Omega^2}. \quad (1.7)$$

Pokud zavedeme veličinu *poměrný útlum*

$$\zeta = \frac{b}{2\sqrt{km}} \quad (1.8)$$

a *zesílení*

$$k_{DC} = \frac{1}{k}, \quad (1.9)$$

dostaneme tvar

$$G(s) = \frac{k_{DC}\Omega^2}{s^2 + 2\zeta\Omega s + \Omega^2}. \quad (1.10)$$

Dalším možným tvarem je

$$G(s) = \frac{k_{DC}}{\tau^2 s^2 + 2\zeta\tau s + 1} \quad (1.11)$$

1.3.2 Parametry a charakteristiky

Set time:

$$T_s = \frac{4,6}{\zeta\Omega}. \quad (1.12)$$

2 Řešení m rovnic o n neznámých

Řadu praktických a užitečných úloh můžeme formulovat jako řešení m rovnic o n neznámých. Může to být prokládání přímky naměřenými daty, řešení inverzní úlohy kinematiky průmyslového robotu nebo hledání parametrů systému při identifikaci.

Podle povahy řešeného problému může jít o rovnice *lineární* nebo *nelineární* s různým vztahem počtu rovnic m a počtu neznámých n ($=, >, <$).

Smyslem této kapitoly je *na jednom místě* shrnout, vysvětlit a na příkladech ukázat řešení všech popsaných kombinací. V běžné výuce jsou tato témata rozptýlena mezi různé ročníky a předměty (matematika, numerika, kinematika, ...) a jsou od sebe vzdálena v prostoru i čase. To často ztěžuje pochopení souvislostí.

Pěkným příkladem takové *integrace různých teorií, algoritmů a přístupů* je příkaz \ v MATLABu. Slouží k řešení „archetypální“ rovnice $\mathbf{Ax} = \mathbf{b}$ prostým zápisem $x = A\backslash b$. Podle tvaru matice \mathbf{A} (skalár, čtvercová nebo obdélníková) se pak zvolí vhodný algoritmus řešení¹.

Možné varianty problému m rovnic o n neznámých shrneme v následujících poznámkách a tabulce. Detailně se jim pak budeme věnovat v jednotlivých sekcích této kapitoly.

- $m = n$ – počet neznámých je roven počtu rovnic, existuje nula, jedno nebo několik „bodů“ řešení; grafická představa v rovině $x - y$: průnik dvou křivek; příklad z kinematiky: manipulátor
- $m > n$ – počet rovnic je větší než počet neznámých; matematicky: neexistuje řešení, dále však uvidíme, že i takovou úlohu je užitečné řešit - hledat řešení přibližné; typická aplikace: prokládání přímky nebo jiné funkce množinou naměřených bodů.
- $m < n$ – počet neznámých je větší než počet rovnic, matematicky: nekonečně mnoho řešení.

	soustava lineárních rovnic $\mathbf{Ax} = \mathbf{b}$	soustava nelineárních rovnic $\mathbf{f}(\mathbf{x}) = \mathbf{b}$
$m = n > 1$	$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$	Zobecnění Newtonovy metody, iterační metoda: $\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{J}^{-1}\mathbf{f}(\mathbf{x})$
$m > n$	Lineární nejmenší čtverce (OLS): $\mathbf{x} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b}$	Použití pseudoinverze (nelineární nejm. čtv. = NLS), iterační metoda: $\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{J}^\dagger \mathbf{f}(\mathbf{x})$ Tlumené nejmenší čtverce (DLS): $\mathbf{x}_{k+1} = \mathbf{x}_k - (\mathbf{J}^T \mathbf{J} + \lambda \mathbf{I})^{-1} \mathbf{J}^T \mathbf{f}(\mathbf{x})$
$m < n$	OLS ² : $\mathbf{x} = \mathbf{A}^T (\mathbf{A} \mathbf{A}^T)^{-1} \mathbf{b}$	NLS, DLS: $\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{J}^T (\mathbf{J} \mathbf{J}^T + \lambda \mathbf{I})^{-1} \mathbf{f}(\mathbf{x})$

¹ Pokud je \mathbf{A} skalární hodnota, podělí se touto hodnotou vektor \mathbf{b} . Pokud je matice čtvercová, najde se přesné řešení (pokud existuje). Pokud je obdélníková, najde se řešení podle „nejmenších čtverců“. Jednou formulací tedy řešíme celý prostřední sloupec tabulky. Více v helpu pod heslem `mldivide` (přesnou implementaci se ale nedozvímíme, kód pro `mldivide` není otevřený).

Obsah kapitoly:

2.1	Intro: $m = n = 1$	9
2.1.1	Newtonova metoda (NM)	9
2.1.2	Příklad: Numerické řešení jedné nelineární rovnice	10
2.2	Lineární rovnice ($m = n > 1$)	12
2.2.1	Příklad: průsečík dvou přímek v rovině	12
2.2.2	Příklad: jak nakreslit přímku dvěma body v rovině	13
2.2.3	Příklad: interpolace pomocí polynomu	13
2.3	Lineární rovnice ($m > n$) aneb obyčejné nejmenší čtverce (OLS)	15
2.3.1	Úvodní úvaha	15
2.3.2	Definice problému	15
2.3.3	Pohled č.1: Pseudoinverze	15
2.3.4	Pohled č.2: Řešení minimalizací součtu čtverců odchylky	16
2.3.5	Příklad: Proložení přímky čtyřmi body aneb kalibrace teploměru	17
2.3.6	Příklad: Nelineární teploměr aneb čtyřmi body a parabola	18
2.4	Nelineární rovnice ($m = n > 1$)	19
2.4.1	Úvodní příklad	19
2.4.2	Newtonova metoda pro soustavu nelineárních rovnic	20
2.4.3	Příklad: dvě křivky v rovině	21
2.4.4	Příklad: Inverzní kinematika robotu v rovině	22
2.4.5	Vylepšení Newtonovy metody	24
2.4.6	Příklad: RR manipulátor – srovnání konvergence inverze a LMa	25
2.5	Nelineární rovnice ($m > n$) aneb nelineární nejmenší čtverce (NLS)	26
2.5.1	Úvod a Gauss-Newtonova metoda	26
2.5.2	Příklad: Dvě rovnice o jedné neznámé	26
2.5.3	Tlumené nejmenší čtverce aneb Levenberg-Marquardt algoritmus (LMa)	29
2.5.4	Vážené tlumené nejmenší čtverce	29
2.5.5	Příklad: Inverzní kinematika redundantního manipulátoru v prostředí s překážkami	30
2.6	Málo rovnic aneb $m < n$	32
2.6.1	Příklad pro lineární rovnice: parabola dvěma body	32
2.6.2	Příklad pro nelineární rovnice: redundantní RRR manipulátor	33
2.7	Otzázkы a neřešené úlohy	35

² Všimněme si, že je použita jiná forma OLS než v předchozím řádku $m > n$ – takzvaná *pravá inverze*. Více v sekci 2.3.3.

2.1 Intro: $m = n = 1$

Nejprve se podíváme na příklad jedné rovnice o jedné neznámé. Přeskočíme triviální případ jedné lineární rovnice o jedné neznámé³ a zabývejme se jednou *nelineární* rovnicí o jedné neznámé. Některé speciální případy jako

$$x^2 - 2x + 3 = 0 \quad (2.1)$$

nebo

$$\cos(5x + 1) + 2 = 0 \quad (2.2)$$

umíme řešit tzv. *analyticky* neboli v uzavřeném tvaru.

Obecně ale nelineární rovnici analyticky (přesně) řešit neumíme, musíme se spojit s řešením *numerickým* (přibližným). Numerických metod pro řešení naší rovnice (neboli pro hledání kořene funkce jedné proměnné) existuje celá řada⁴, pro nás bude teď užitečné se seznámit s *Newtonovou metodou*(NM).

2.1.1 Newtonova metoda (NM)

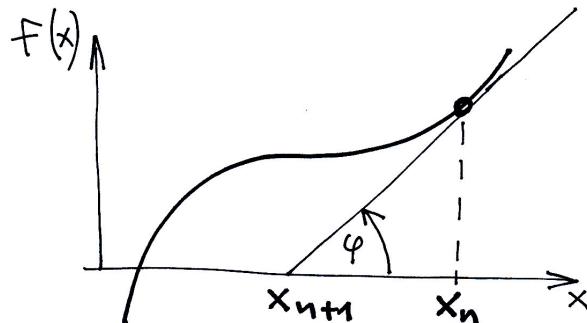
Máme zadánu funkci $f(x)$ a hledáme její kořen = hledáme takové x , pro které platí $f(x) = 0$.

Newtonova metoda funguje tak, že v daném pracovním bodě nahradí funkci $f(x)$ její tečnou a počítá průsečík této tečny s vodorovnou osou (kořen tečny).

Vzhledem k tomu, že $f(x)$ ale není přímka, nemůžeme najít tímto způsobem přesné řešení. Postupujeme proto tak, že začneme v nějakém bodě (počáteční odhad) a postup opakujeme (*iterace*). Pokud řešení *konverguje*, dostaneme po několika iteracích řešení s požadovanou přesností.

Algoritmus NM zapíšeme takto⁵:

$$x_{k+1} = x_k - \frac{f(x)}{f'(x)} \quad (2.3)$$



Obrázek 2.1 Newtonova metoda

³ Příklad: $5x + 3 = 0$

⁴ Metoda půlení intervalu, Regula-falsi, ...

⁵ Odvození: $\tan(\varphi) = f'(x) = \frac{f(x)}{x_k - x_{k+1}}$, vyjádříme $x_{k+1} = \dots$

Kód v MATLABu bude vypadat takto:

```
x=1.5 % počáteční odhad
err = inf
while err > 1e-6
    f = ... % vypočítej f(x) v aktuálním bodě x
    f_der = ... % vypočítej derivaci f(x)
    x = x - f/f_der % jádro NM
end
```

2.1.2 Příklad: Numerické řešení jedné nelineární rovnice

[Ex.: NM01] Uvažujme jednoduchou nelineární rovnici u které nelze najít řešení v uzavřeném tvaru:

$$\frac{-x^2}{2} + \sin(x) + 1 = 0 \quad (2.4)$$

Numerické řešení můžeme napsat takto:

```
clc
clear
x = 0 % počáteční odhad
err = Inf; % ini error
while err > 1e-6 % dokud je err >..., opakuj smyčku
    f = -x^2/2 + sin(x) + 1; % vypočítej f(x) v aktuálním bodě x
    f_der = -x + cos(x); % vypočítej derivaci f(x)
    x = x - f/f_der % jádro NM
    err = f^2; % chyba musí být vždy kladná...
end
% Vypiš řešení:
disp(['Solution is: ',num2str(x)])
```

A nebo použít vestavěnou funkci FSOLVE v MATLABu:

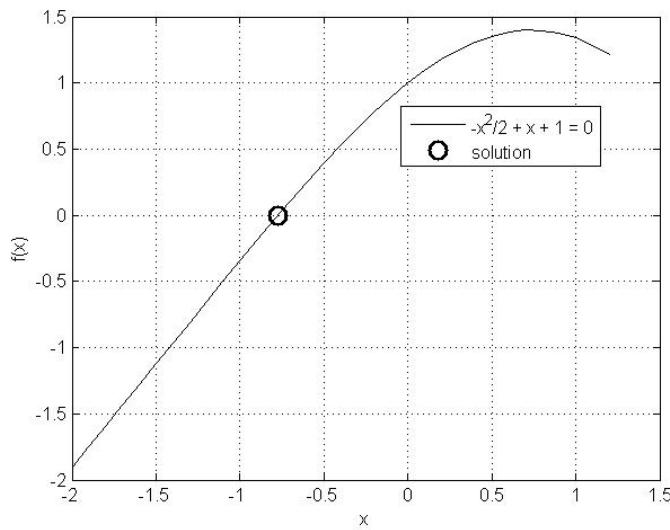
```
x0 = 0; % počáteční odhad řešení
x1_ = fsolve(@(x) -x.^2/2 + sin(x) + 1 ,x0) % použití funkce FSOLVE
```

Když je to možné, vždycky je vhodné si při řešení pomocí vizualizací. Na obr. 2.2 vidíme vykreslení funkce a nalezené řešení.

Otzásky, úvahy a úkoly

- Podrobné studium úlohy hledání kořene nelineární funkce jedné proměnné je dosti důležité, ukazuje totiž mnoho vlastností, problémů a závludností, které se vyskytují při řešení soustav více rovnic. Přitom lze funkci, konvergenci metody i řešení jednoduše vizualizovat. Všem nadšeným a motivovaným studujícím⁶ lze jednoznačně

⁶ Genderově zcela vyvážený termín používaný na doporučení MŠMT místo zcela nevyhovujícího slova „student“.



Obrázek 2.2 Vykreslení průběhu nelineární funkce a jejího kořene

doporučit: naprogramuj si Newtonovu metodu sám a vyzkoušej si řešení různých příkladů. Přitom si vykresluj chování metody. Testuj vlivy různých úprav a variant metody a jejich parametrů.

- Zkus vymyslet funkci $f(x)$ a počáteční odhad x_0 takové, že řešení Newtonovou metodou (2.3) nebude konvergovat nebo dokonce úplně zhavaruje! Co by se s tím dalo udělat?
- Jaké znáš jiné numerické metody pro řešení úlohy hledání kořene nelineární funkce jedné proměnné?

2.2 Lineární rovnice ($m = n > 1$)

Soustavu m lineárních rovnic o n neznámých lze zapsat v maticovém tvaru takto:

$$\mathbf{Ax} = \mathbf{b}, \quad (2.5)$$

kde \mathbf{A} je čtvercová matice. Řešení snadno nalezneme pomocí inverze matice:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}. \quad (2.6)$$

2.2.1 Příklad: průsečík dvou přímek v rovině

Uvažujme dvě přímky v rovině popsané rovnicí

$$y = k_1x + c_1$$

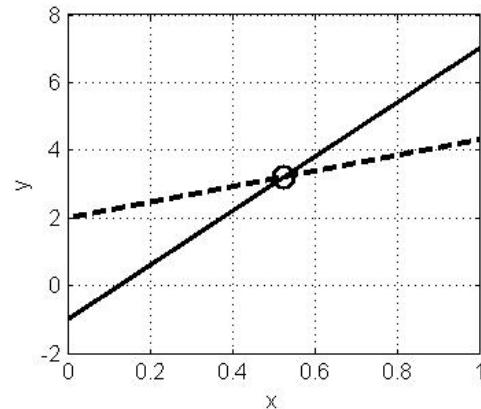
$$y = k_2x + c_2$$

kde k_i a c_i jsou konstanty.

Řešením této soustavy 2 rovnic o 2 neznámých je bod $[x, y]$, který je oběma přímkám společný = jejich *průsečík*.

Vektor neznámých \mathbf{x} v rovnici (2.5) je tedy definován v tomto případě takto:

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} \quad (2.7)$$



Obrázek 2.3 Dvě přímky v rovině a jejich průsečík

a matice \mathbf{A} a vektor \mathbf{c} takto:

$$\mathbf{A} = \begin{bmatrix} -k_1 & 1 \\ -k_2 & 1 \end{bmatrix}; \mathbf{b} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \quad (2.8)$$

Otázky, úvahy a úkoly

- Vykresli 2 přímky do obrázku, vypočítej jejich průsečík a přidej ho do obrázku (podobně jako na obr. 2.3).
- Vyzkoušej výpočet průsečíku pro dvě rovnoběžné přímky. Co se stane? Jak se tedy matematicky projeví to, že přímky nemají průsečík?

2.2.2 Příklad: jak nakreslit přímku dvěma body v rovině

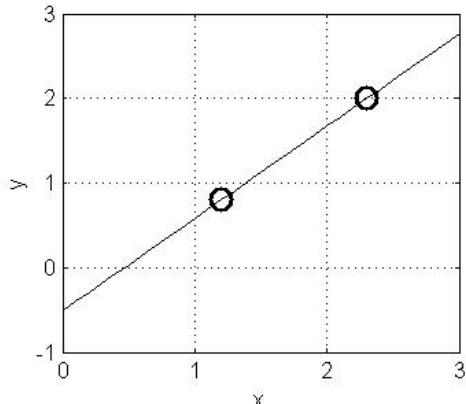
Uvažujme dva zadané body v rovině $[x_1, y_1]$ a $[x_2, y_2]$. Vypočítejte koeficienty přímky vedoucí těmito dvěma body. Rovnice přímky je

$$y = kx + c. \quad (2.9)$$

Řešíme tedy soustavu dvou rovnic

$$y_1 = kx_1 + c$$

$$y_2 = kx_2 + c$$



o dvou neznámých k a c .

Definice problému podle rovnice (2.5) je tato:

$$\mathbf{x} = \begin{bmatrix} k \\ c \end{bmatrix}, \mathbf{A} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \quad (2.10)$$

Otázky, úvahy a úkoly

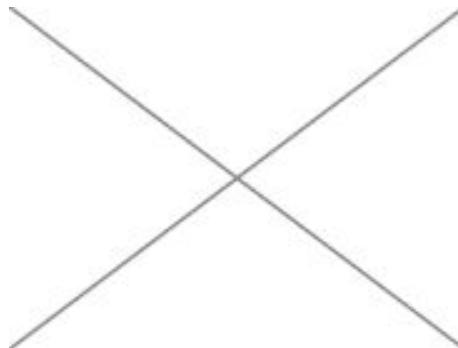
- Zkus vymyslet takové zadání problému se dvěma body, kdy výpočet nebude fungovat. A vysvětli proč!

2.2.3 Příklad: interpolace pomocí polynomu

Uvažujme následující úlohu plánování hladké trajektorie nějakého mechanismu. Máme zadánu polohu q a rychlosť \dot{q} ve dvou časových okamžicích t_0 a t_F . Definujeme tedy čtyři podmínky

$$q(t_0) = q_0, q(t_F) = q_F$$

$$\dot{q}(t_0) = \dot{q}_0, \dot{q}(t_F) = \dot{q}_F$$



Obrázek 2.5 TODO: obrázek definující polohu a její derivaci v bodech t_0 a t_F

Tyto podmínky dokáže zajistit polynom třetího stupně

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

Výše uvedené podmínky pak zapíšeme jako soustavu čtyř algebraických rovnic o čtyřech neznámých

$$\begin{aligned} q_0 &= a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 \\ q_F &= a_0 + a_1 t_F + a_2 t_F^2 + a_3 t_F^3 \\ \dot{q}_0 &= a_1 + 2a_2 t_0 + 3a_3 t_0^2 \\ \dot{q}_F &= a_1 + 2a_2 t_F + 3a_3 t_F^2 \end{aligned}$$

nebo ekvivalentně v maticové podobě

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 \\ 0 & 1 & 2t_0 & 3t_0^2 \\ 1 & t_F & t_F^2 & t_F^3 \\ 0 & 1 & 2t_F & 3t_F^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} q_0 \\ \dot{q}_0 \\ q_F \\ \dot{q}_F \end{bmatrix} \quad (2.11)$$

Opět jsme dostali formulaci podle rovnice (2.5), kterou vyřešíme pro vektor neznámých parametrů a_i .

Otázky, úvahy a úkoly

- Vykresli polynom mezi body t_0 a t_F podle postupu v tomto příkladu.

2.3 Lineární rovnice ($m > n$) aneb obyčejné nejmenší čtverce (OLS)

2.3.1 Úvodní úvaha

V příkladu 2.2.2 jsme prokládali přímku dvěma body v rovině. Co kdyby těch bodů ale bylo více, např. 3, 4, 50, 1000? Pokud jsou body rozmištěny obecně (neleží opravdu přesně na jedné přímce), tak je zřejmé, že to *nejde*. Matematicky jde o úlohu $m = 1000$ rovnic o $n = 2$ neznámých (k a c).

Má ale smysl takovou úlohu *nějak* řešit? Třeba tak, že bychom přímku body proložili *přibližně*?

Ano. V řadě praktických úloh takové zadání smysl má, typicky pokud chceme aproximovat naměřená data nějakou funkcí (přímkou, polynomem, sinusovkou). Slovem „*přibližně*“ pak myslíme to, že *minimalizujeme součet druhé mocniny (kvadrát, čtverec) odchylek všech bodů od funkce*.

V případě, že je tato funkce tzv. *lineární v parametrech* (např. právě naše přímka, ale nejen ta), můžeme použít obyčejné (lineární) nejmenší čtverce (angl. OLS) a výpočet se provede v jednom kroku.

Pokud funkce lineární v parametrech není, musí se použít iterační výpočet pomocí metody *nelineárních nejmenších čtverců* - o tom budeme dále mluvit v další části 2.5.

Dále tedy budeme mluvit o *lineárních nejmenších čtvercích*.

2.3.2 Definice problému

V kap. 2.2 jsme definovali problém pomocí rovnice (2.5). V případě m rovnic o n neznámých zapíšeme problém stejně, pouze budeme mít jiné velikosti matice a vektorů:

$$\mathbf{Ax} = \mathbf{b}, \quad (2.12)$$

kde \mathbf{A} je obdélníková matice $m \times n$, \mathbf{x} je vektor $n \times 1$ a \mathbf{b} je vektor $m \times 1$.

V následujících dvou sekcích ukážeme dva pohledy na věc (dva různé způsoby řešení) a uvidíme, že ve výsledku vedou na stejnou maticovou formulaci.

2.3.3 Pohled č.1: Pseudoinverze

Víme, že inverzi matice lze spočítat pouze pro čtvercovou matici, řešení rovnice (2.12) tedy nelze provést tak jako v případě (2.6), protože matice \mathbf{A} je obdélníková.

Co kdyby ale existovala nějaká zvláštní inverze, taková *pseudoinverze*, která by byla definovaná i pro obdélníkovou matici? Když napíšeme v MATLABu `pinv`, zjistíme, že něco takového opravdu existuje...

Pseudoinverze se označuje \mathbf{A}^\dagger , je definovaná pro obdélníkovou matici a má (velmi zjednodušeně řečeno) podobné vlastnosti jako inverze matice. Její výpočet je snadný pokud má matice \mathbf{A} plnou hodnost, pak je pro matici s lineárně nezávislými sloupcí definována takto (levá inverze):

$$\mathbf{A}^\dagger = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T, \quad (2.13)$$

nebo pro matici s lineárně nezávislými řádky takto (pravá inverze):

$$\mathbf{A}^\dagger = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}, \quad (2.14)$$

kde velikost \mathbf{A} je $m \times n$ a velikost \mathbf{A}^\dagger je $n \times m$.

Pokud matice \mathbf{A} nemá plnou hodnost⁷, je výpočet složitější. Používá se singulární rozklad (SVD)⁸.

Řešení soustavy m rovnic o n neznámých, kde $m > n$ pak zapíšeme jednoduše takto:

$$\mathbf{x} = \mathbf{A}^\dagger \mathbf{b}, \quad (2.15)$$

2.3.4 Pohled č.2: Řešení minimalizací součtu čtverců odchylky

Druhý pohled na otázku řešení rovnice (2.12) směřuje na minimalizaci odchylky (ne-přesnosti) řešení soustavy rovnic. Přesné řešení najít nelze, bude tedy platit, že

$$\mathbf{r} = \mathbf{Ax} - \mathbf{b} \neq \mathbf{0}. \quad (2.16)$$

Vektor \mathbf{r} zde reprezentuje odchylky jednotlivých m rovnic od přesného řešení. Při hledání přibližného řešení budeme chtít tyto odchylky minimalizovat, přesněji budeme minimalizovat součet jejich kvadrátů. Nadefinujeme tedy kritérium J takto:

$$J(\mathbf{x}) = \sum_1^m r_i^2 \quad (2.17)$$

Minimalizace kritéria J znamená, že platí:

$$\frac{\partial J}{\partial x_i} = 0, \text{ pro všechna } i = 1, \dots, n. \quad (2.18)$$

Po kratším odvození, které si zvídavý čtenář jistě sám dohledá, případně i provede, dostaneme maticovou rovnici

$$\mathbf{A}^T \mathbf{Ax} = \mathbf{A}^T \mathbf{b}. \quad (2.19)$$

Řešení pak získáme ve tvaru

$$\mathbf{x} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b} \quad (2.20)$$

Vidíme, že jsme vlastně dostali definici (levé) pseudoinverze. Pohled č. 2 tedy vede na stejný výsledek (2.15) jako pohled č. 1.

⁷ Příklad situace, kdy má matice za určitých podmínek sníženou hodnost (je singulární), je v sekci 2.4.4 (manipulátor v situaci $\mathcal{J}_1 = \mathcal{J}_2 = 0$).

⁸ Možných způsobů výpočtu je více, SVD je použit v MATLABu i jiných prostředích pro numerickou matematiku. Stačí napsat open pinv...

2.3.5 Příklad: Proložení přímky čtyřmi body aneb kalibrace teploměru

[Ex.: NM03]

Uvažujme, že máme teploměr s napěťovým výstupem, který chceme kalibrovat. Provedeme 4 měření a do grafu zaznamenáme napětí u a naměřenou teplotu θ (měříme např. přesným rtuťovým teploměrem). Předpokládáme, že teplotní sensor je lineární a pro kalibraci budeme chtít naměřené hodnoty approximovat přímkou:

$$\theta = ku + c \quad (2.21)$$

Pokud uvážíme provedená 4 měření,
máme $m = 4$ rovnice

$$\theta_1 = ku_1 + c$$

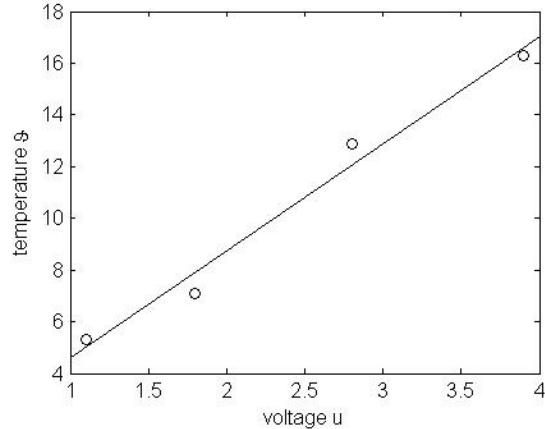
$$\theta_2 = ku_2 + c$$

$$\theta_3 = ku_3 + c$$

$$\theta_4 = ku_4 + c$$

Tyto 4 rovnice můžeme podle (2.12)
zapsat maticově takto:

$$\begin{bmatrix} u_1 & 1 \\ u_2 & 1 \\ u_3 & 1 \\ u_4 & 1 \end{bmatrix} \begin{bmatrix} k \\ c \end{bmatrix} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix}$$



Obrázek 2.6 Linearizace teploměru = proložení přímky čtyřmi body

a řešit podle (2.20). Kód v MATLABu může vypadat takto [example NM03]:

```

u = [1.1 1.8 2.8 3.9]' % namerene napeti
theta = [5.3 7.1 12.9 16.3]' % namerena teplota
A = [u, ones(4,1)] % definice A
b = theta % definice b
x = inv(A'*A)*A' * b % reseni pomocí OLS
% kde x = [k;c]

```

2.3.6 Příklad: Nelineární teploměr aneb čtyřmi body a parabola

[Ex.: NM03]

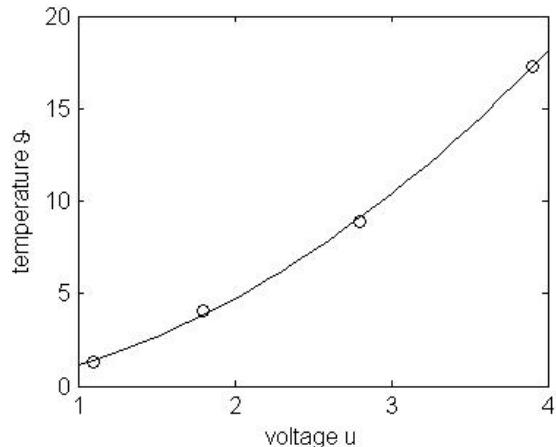
Uvažujme malou modifikaci předchozího příkladu: opět naměříme 4 hodnoty, ale teploměr bude mít kvadratickou charakteristiku – budeme tedy hledat koeficienty funkce

$$\theta = ku^2 + pu + c \quad (2.23)$$

Funkce, kterou chceme použít pro proložení body, je nelineární, z hlediska řešení ale problém dokážeme formulovat ve tvar $\mathbf{Ax} = \mathbf{b}$ – problém je tzv. *lineární v parametrech*.

Matice \mathbf{A} a vektory \mathbf{x} a \mathbf{b} budou mít tento tvar:

$$\mathbf{A} = \begin{bmatrix} u_1^2 & u_1 & 1 \\ u_2^2 & u_2 & 1 \\ u_3^2 & u_3 & 1 \\ u_4^2 & u_4 & 1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} k \\ p \\ c \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix} \quad (2.24)$$



Obrázek 2.7 Teploměr II. = proložení paraboly čtyřmi body

Jádro kódu v MATLABu pak bude vypadat takto:

```
A = [u.^2, u, ones(4,1) ]
b = theta
x = inv(A'*A)*A' * b
```

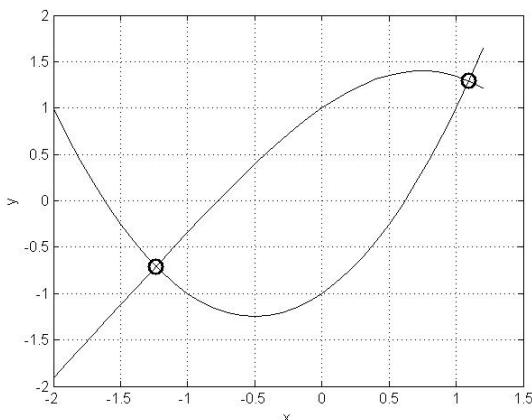
2.4 Nelineární rovnice ($m = n > 1$)

V sekci 2.3 jsme řešili problém m lineárních rovnic o n neznámých. Jak budeme postupovat, pokud jsou rovnice nelineární?

2.4.1 Úvodní příklad

Začneme příkladem dvou nelineárních rovnic

$$y = x^2 + x - 1$$
$$y = -x^2 + \sin(x) + 1$$



Obrázek 2.8 Dvě nelineární funkce a jejich průsečíky

Pokud je vykreslíme do grafu, zjistíme, že se protínají. Souřadnice průsečíku $[x, y]$ jsou řešení soustavy dvou rovnic o dvou neznámých (??). Vidíme také, že řešení (průsečíky) jsou dvě⁹.

Je zřejmé, že postup použitý pro soustavu lineárních rovnic nám nepomůže - problém prostě pomocí matice \mathbf{A} a vektorů \mathbf{x} a \mathbf{b} neformulujeme.

Problém můžeme zapsat do tvaru

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}, \quad (2.25)$$

Na základě zkušenosti s Newtonovou metodou (NM) pro $n = m = 1$ se můžeme domnívat, že:

- Analytické řešení obecně nebude možné získat, budeme muset použít nějakou numerickou metodu.
- Měli bychom tedy najít nějaké zobecnění NM (a tedy derivace) pro více funkcí o více proměnných.

⁹ To je zásadní rozdíl oproti soustavě lineárních rovnic, kde je řešení (průsečík) jen jedno, případně žádné. V případě nelineárních rovnic může být počet řešení (průsečíků) 0,1,2,...

2.4.2 Newtonova metoda pro soustavu nelineárních rovnic

Problém řešení soustavy m rovnic o n neznámých zapíšeme jako vektor funkcí takto¹⁰:

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ f_m(x_1, x_2, \dots, x_n) \end{bmatrix} = \mathbf{0}, \quad (2.26)$$

kde $\mathbf{x} = [x_1, x_2, \dots, x_n]^T$ je vektor neznámých.

Zobecněním pojmu *derivace funkce* pro vektor funkcí je *jakobián* neboli Jacobihho matice¹¹. Je to matice parciálních derivací jednotlivých funkcí podle jednotlivých proměnných:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \quad (2.27)$$

Pokud je $m = n$, jakobián je čtvercová matice.

Jednu iteraci numerického výpočtu pak zapíšeme úplně stejně jako v případě Newtonovy metody pro funkci jedné proměnné:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{J}(\mathbf{x})^{-1} \mathbf{f}(\mathbf{x}) \quad (2.28)$$

Otzázkы, úvahy a úkoly

- Srovnej tvar rovnice (2.28) s NM pro funkci jedné proměnné (2.3). Je tam nějaká podobnost? Vidíš analogii operace $\frac{1}{x}$ ve světě maticových výpočtů?

¹⁰ Vidíme, že místo rovnice $\mathbf{Ax} = \mathbf{b}$ máme $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ - pravá strana je tedy nulová. Případná pravá strana je „schována“ ve funkci \mathbf{f} .

¹¹ Carl Gustav Jacob Jacobi (1804–1851).

2.4.3 Příklad: dvě křivky v rovině

Podívejme se ještě jednou [na příklad 2.4.1](#) dvou funkcí vykreslených v rovině. Zapíšeme je v souladu s definicí [\(2.26\)](#):

$$\begin{aligned} f_1 &= x_1^2 + x_1 - 1 - x_2 = 0 \\ f_2 &= -x_1^2 + \sin(x_1) - 1 - x_2 = 0 \end{aligned}$$

Jakobián pak vypadá takto:

$$\mathbf{J} = \begin{bmatrix} 2x+1 & -1 \\ -x+\cos(x) & -1 \end{bmatrix} \quad (2.29)$$

[Ex.: NM02]

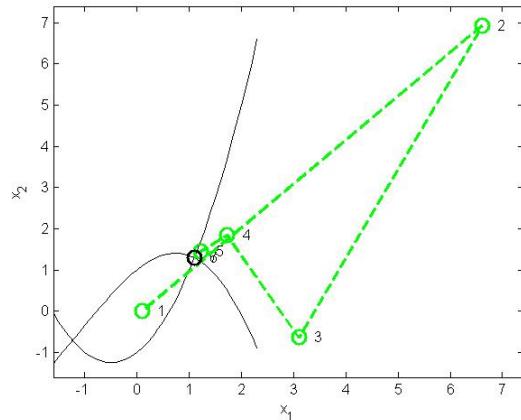
Kód v MATLABu bude vypadat takto:

```
x(1) = 0.1 % initial guess
x(2) = 0 % initial guess
for i = 1:10
    f = [x(1)^2 + x(1) - 1 - x(2) ; ...
           -x(1)^2/2 + sin(x(1)) + 1 - x(2)];
    J = [2*x(1)+1 , -1 ; -x(1)+cos(x(1)) , -1];
    x = x - inv(J) * f;
end
```

Na následujícím obrázku vidíme vývoj řešení po jednotlivých iteracích. Řešení blízko přesnému dosáhneme po 7 iteracích.

Otázky, úvahy a úkoly

- Kdy výše popsaný algoritmus nebude fungovat? (hint: jak se počítá inverze matice?)
- Dokážeš vymyslet alespoň dva body (počáteční odhadů), ze kterých konkrétní příklad ?? nebude konvergovat?



Obrázek 2.9 Dvě nelineární funkce – postup iterací od počátečního odhadu k řešení

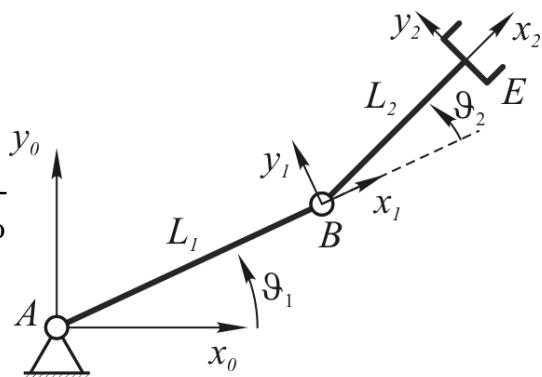
2.4.4 Příklad: Inverzní kinematika robota v rovině

[Ex.: KD12]

Na obrázku 2.10 je znázorněn rovinný manipulátor (robot), který má dva klouby (polohy) ϑ_1 a ϑ_2 . V kinematice robotů rozlišujeme dvě základní úlohy: přímou a inverzní.

V úloze přímé kinematiky máme zadány kloubové souřadnice (ϑ_1, ϑ_2) a počítáme kartézské souřadnice (x, y). Tato úloha je snadno řešitelná:

$$\begin{aligned}x &= L_1 \cos(\vartheta_1) + L_2 \cos(\vartheta_1 + \vartheta_2) \\y &= L_1 \sin(\vartheta_1) + L_2 \sin(\vartheta_1 + \vartheta_2)\end{aligned}$$



Obrázek 2.10 Rovinný manipulátor

Opačná (inverzní) úloha, kdy máme dánou se dvěma stupni volnosti (x, y) a hledáme (ϑ_1, ϑ_2) je obtížnější a obecně není řešitelná u uzavřeném tvaru¹².

Formulujme tedy problém podle (2.26):

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\vartheta_1, \vartheta_2) \\ f_2(\vartheta_1, \vartheta_2) \end{bmatrix} = \begin{bmatrix} L_1 \cos(\vartheta_1) + L_2 \cos(\vartheta_1 + \vartheta_2) - x \\ L_1 \sin(\vartheta_1) + L_2 \sin(\vartheta_1 + \vartheta_2) - y \end{bmatrix} = \mathbf{0}, \quad (2.30)$$

kde $\mathbf{x} = [\vartheta_1, \vartheta_2]^T$ je vektor neznámých.

Jakobián vytvoříme takto:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial \vartheta_1} & \frac{\partial f_1}{\partial \vartheta_2} \\ \frac{\partial f_2}{\partial \vartheta_1} & \frac{\partial f_2}{\partial \vartheta_2} \end{bmatrix} = \begin{bmatrix} -L_1 \sin(\vartheta_1) - L_2 \sin(\vartheta_1 + \vartheta_2) & -L_2 \sin(\vartheta_1 + \vartheta_2) \\ L_1 \cos(\vartheta_1) + L_2 \cos(\vartheta_1 + \vartheta_2) & L_2 \cos(\vartheta_1 + \vartheta_2) \end{bmatrix} \quad (2.31)$$

Algoritmus řešení pak bude odpovídat rovnici (2.28). Na obr. 2.11 pak vidíme vykreslený postup konvergence k zadané hodnotě $x = 1, y = 0.5$.

Viz také:

Část 2.4.6

Je zřejmé, že je potřeba přibližně 7-10 iterací k dosažení přibližně dostatečně přesného řešení, přičemž tyto iterace nesměřují „spořádaně“ k cíli, nýbrž nehezky oscilují. Jistě nás také napadne otázka, zda se použitý algoritmus nemůže stát nestabilním¹³. V příkladu 2.4.6 ukážeme, jak můžeme konvergenci numerické iterační metody vylepšit *tlumením*.

Zajímavá situace nastane pro $\vartheta_1 = 0, \vartheta_2 = 0$. Jakobián bude mít hodnotu

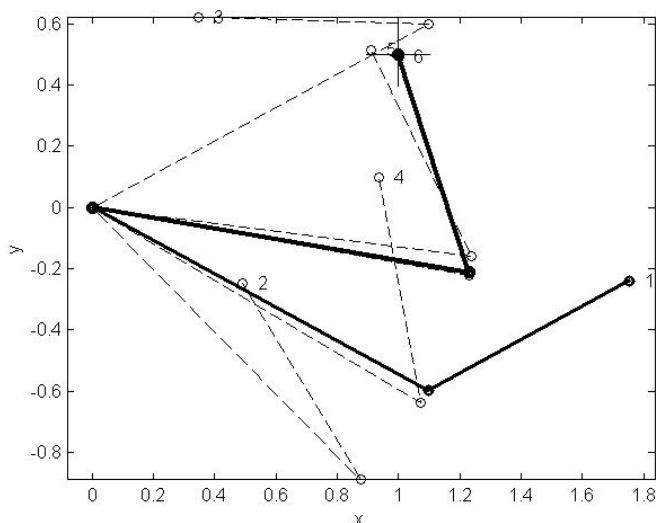
$$\mathbf{J} = \begin{bmatrix} 0 & 0 \\ L_1 + L_2 & L_2 \end{bmatrix}$$

a jeho determinant bude nulový. Nelze tedy určit inverzi matice \mathbf{J} . Říkáme, že taková matice je *singulární*. Znamená to, že inverzní kinematiku nelze v této poloze vyřešit?

Odpověď je: kinematiku (pochopitelně) lze vyřešit, musíme ale vylepšit algoritmus v rovnici (2.28). Více v sekci 2.3.3 a především pak v 2.5.3.

¹² Máme soustavu dvou rovnic o dvou neznámých (ϑ_1, ϑ_2). Obvyklý postup „vyjádříme jednu neznámou z jedné rovnice a dosadíme do druhé“ fungovat nebude. V tomto *konkrétním* případě je přesné (analytické, v uzavřeném tvaru) řešení možné - popsáno je např. v [TODO RKD skripta]. I tak je ale tento příklad pěknou motivační ukázkou problematiky řešení $m = n$ nelineárních rovnic.

¹³ Je zde zřejmá analogie s Newtonovou metodou pro hledání kořene funkce jedné proměnné, která také může snadno divergovat.



Obrázek 2.11 Jednotlivé iterace řešení (kroky 1,2,...,6). Je vidět, že řešení hodně „skáče“ než se ustálí v cílové hodnotě (označeno křížkem) [example KD12].

2.4.5 Vylepšení Newtonovy metody

V příkladech 2.4.3 i 2.4.4 jsme viděli, že ačkoli Newtonova metoda konverguje k řešení, její chování je poněkud „divoké“. A v řadě případů je ještě divočejší a metoda tzv. diverguje, tj. nekonverguje k řešení.

Otázkou tedy je, jak chování Newtonovy metody vylepšit. Řešení jsou tato:

- **Tlumení kroku v Newtonově metodě** - konvergence metody se zpomalí vhodně zvoleným koeficientem α :

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \mathbf{J}(\mathbf{x})^{-1} \mathbf{f}(\mathbf{x}) \quad (2.32)$$

- **Gauss-Newtonova metoda používající pseudoinverzi** - odvození a zdůvodnění této metody je v další sekci 2.5, kdy řešíme soustavu nelineárních rovnic $m > n$. Použití na úlohu $m = n$ může vylepšit konvergenci, více v 2.5.1:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha (\mathbf{J}^T \mathbf{J})^{-1} \mathbf{J}^T \mathbf{f}(\mathbf{x}) \quad (2.33)$$

- **Levenberg-Marquardtova metoda** - vylepšení Gauss-Newtonovy metody o tlumící člen, více v 2.5.3:

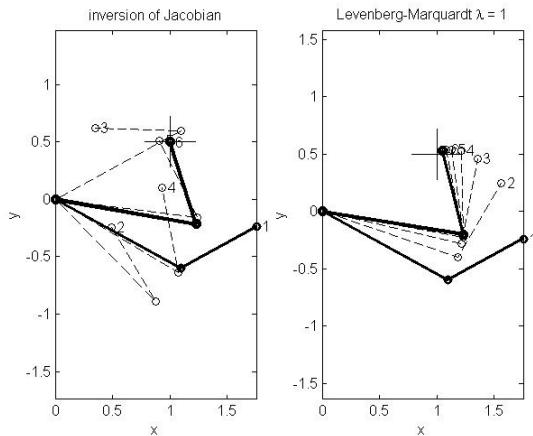
$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha (\mathbf{J}^T \mathbf{J} + \lambda \mathbf{I})^{-1} \mathbf{J}^T \mathbf{f}(\mathbf{x}) \quad (2.34)$$

2.4.6 Příklad: RR manipulátor – srovnání konvergence inverze a LMa

[Ex.: KD15]

V tomto příkladu ukážeme užitečnost LMa oproti prosté inverzi jakobiánu. V příkladu [2.4.4](#) jsme řešili inverzní kinematiku manipulátoru ($m = n = 2$). Zjistili jsme, že při použití inverze (nebo pseudoinverze) dochází k celkem značným oscilacím předtím, než algoritmus dokonverguje k řešení.

Na obrázku [2.12](#) vidíme jak může LMa výrazně utlumit „oscilace“ numerické metody. Cenou, kterou platíme může být mírně vyšší počet kroků potřebných pro nalezení řešení s požadovanou přesností a také nutnost nalezení (ladění) správné hodnoty konstanty λ .



Obrázek 2.12 Porovnání konvergence algoritmu při použití inverze (vlevo) a LM algoritmu (vpravo)

2.5 Nelineární rovnice ($m > n$) aneb nelineární nejmenší čtverce (NLS)

2.5.1 Úvod a Gauss-Newtonova metoda

V sekci 2.3 jsme ukázali, jak řešit problém $m > n$ v případě, že ho lze formulovat ve tvaru soustavy lineárních rovnic $\mathbf{Ax} = \mathbf{b}$. Nyní se budeme věnovat obecnějšímu problému soustavy nelineárních rovnic, kdy $m > n$.

Problém tedy formulujeme jako hledání vektoru \mathbf{x} v soustavě nelin. rovnic:

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \quad (2.35)$$

Řešení problému bude kombinací dvou přístupů vysvětlených v sekci 2.4 (soustava nelineárních rovnic, $m = n$) a sekci 2.3.3 (použití pseudoinverze). Stručně postup shrneme takto:

- řešíme problém formulovaný rovnicí (2.26), ovšem nyní pro $m > n$
- v souladu s postupem v kap. 2.4.2 použijeme iterační metodu inspirovanou Newtonovou metodou (rovnice (2.28))
- vzhledem k tomu, že pro obdélníkový jakobián ($m > n$) nelze spočítat inverzi, použijeme místo toho pseudoinverzi.

Jádro iteračního algoritmu zapíšeme takto (tato metoda se jmenuje Gauss-Newtonova):

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (\mathbf{J}^T \mathbf{J})^{-1} \mathbf{J}^T \mathbf{f} \quad (2.36)$$

Podstatu problému detailně rozebereme v následujícím velmi jednoduchém příkladu.

2.5.2 Příklad: Dvě rovnice o jedné neznámé

Uvažujme jednoduchý příklad dvou nelineárních rovnic o jedné neznámé:

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} x^2 - 1 \\ \sin(x) - 0.4 \end{bmatrix} = \mathbf{0} \quad (2.37)$$

Tato soustava rovnic zjevně nemá jedno přesné řešení x , snadno zjistíme, že každá z rovnic má přesné řešení 1 a 0.4115. Znovu ale zopakujeme: ačkoli přesné řešení problému neexistuje, má smysl hledat řešení přibližné, které je dáno minimalizací kritéria J (součet čtverců odchylek).

Vypočteme jakobián (bude rozměru $m \times n$):

$$\mathbf{J} = \begin{bmatrix} 2x \\ \cos(x) \end{bmatrix} \quad (2.38)$$

a napíšeme iterační algoritmus kolem rovnice:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (\mathbf{J}^T \mathbf{J})^{-1} \mathbf{J}^T \mathbf{f} \quad (2.39)$$

Zvolíme počáteční odhad x_0 a spustíme iterační výpočet.

Kód v MATLABu vypadá takto:

```
x = 0
for i = 1:10
    f = [x^2 - 1 ; sin(x) - 0.4]; % vector function f
    J = [2*x ; cos(x)]; % jacobian
    x = x - inv(J'*J)*J'*f; % left pseudoinversion
end
```

Alternativně můžeme algoritmus psát s použitím označení pseudoinverze:

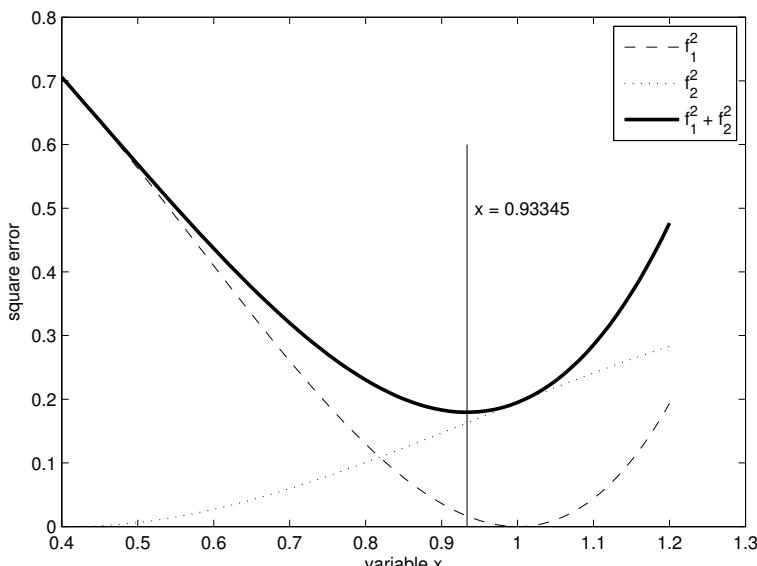
$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{J}^\dagger \mathbf{f} \quad (2.40)$$

a v MATLABu použít příslušný příkaz:

```
...  
x = x - pinv(J)*f; % left pseudoinversion  
...
```

MATLAB přitom za nás správně zvolí levou nebo pravou inverzi (pro případ $m < n$ – kap. 2.6).

Na obrázku 2.13 vidíme polohu řešení $x = 0.93345$ vzhledem k kritériu součtu nejmenších čverců $J = f_1^2 + f_2^2$.



[Ex.: NM04]

Obrázek 2.13 Závislost čtverce chyby první (f_1) a druhé (f_2) rovnice a jejich součtu na proměnné x (připomeňme znovu, že první rovnice má řešení $x = 1$ a druhá $x = 0.4115$)

Otázky, úvahy a úkoly

- Výsledek je $x = 0.9334$. Intuitivně bychom možná čekali hodnotu někde mezi 1 a 0.4 (což jsou přesná řešení obou rovnic nezávisle). Proč vyšlo řešení tak výrazně blíž k 1?

2.5.3 Tlumené nejmenší čtverce aneb Levenberg-Marquardt algoritmus (LMa)

Vlastnosti Gauss-Newtonovy metody lze významně vylepšit přidáním tlumicího člena do matice, kterou invertujeme. Vzniknout tak *tlumené nejmenší čtverce* neboli *Levenberg-Marquardtův algoritmus*:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (\mathbf{J}^T \mathbf{J} + \lambda \mathbf{I})^{-1} \mathbf{J}^T \mathbf{f} \quad (2.41)$$

Zjednodušeně lze říci, že v podstatě nemá smyslu zabývat se rovnicí (2.36), téměř vždy je lepší použít tlumení = LMa (rovnice (2.41)).

Tlumící člen $\lambda \mathbf{I}$ vlastně „kazí“ matici $\mathbf{J} \mathbf{J}^T$ tak, že k její diagonále přidává vhodně zvolenou konstantu λ . Toto tlumení pak také dokáže zajistit fungování algoritmu pokud je \mathbf{J} singulární¹⁴.

Ačkoli je Gauss-Newtonova metoda (použití pseudoinverze) i LMa nutné v případě problémů $m > n$, v příkladu 2.4.6 ukazujeme názorně, že je velmi vhodné i v případech $m = n$.

Bez velkého přehánění lze říci, že LMa je jeden z nejužitečnějších algoritmů vůbec.

2.5.4 Vážené tlumené nejmenší čtverce

V předchozím textu jsme si řekli, že člen $\lambda \mathbf{I}$ v rovnici (2.41) vlastně „kazí“ matici $\mathbf{J} \mathbf{J}^T$ tím, že k diagonále přidává λ . Přirozeným rozšířením této myšlenky je „kazit“ každý prvek diagonály jinou hodnotou λ . Ovlivňujeme tím *váhu* jednotlivých funkcí f_1, f_2, \dots ve vektoru \mathbf{f} . Algoritmus tedy upravíme do tohoto tvaru:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (\mathbf{J}^T \mathbf{J} + \lambda \mathbf{I})^{-1} \mathbf{J}^T \mathbf{f} \quad (2.42)$$

kde λ je diagonální matice. Platí přitom, že *čím větší hodnota λ_i , tím méně nám na výsledku f_i záleží*.

Zcela jasné by to mohlo být z následujícího příkladu 2.5.5.

¹⁴ Vyzkoušej si sám jak bude vypadat výpočet inverze, pseudoinverze příkazem pinv, pseudoinverze pomocí vzorců (2.13) a (2.14) a členu $\mathbf{J}^T(\mathbf{J} \mathbf{J}^T + \lambda \mathbf{I})^{-1}$.

2.5.5 Příklad: Inverzní kinematika redundantního manipulátoru v prostředí s překážkami

[Ex.: KD17]

Uvažujme rovinný manipulátor se čtyřmi klouby ($n = 4$), v kartézském souřadničovém systému nás zajímají dvě souřadnice (x, y) ($m = 2$). Manipulátor je tedy redundantní. Zadáním úlohy je při zadaných (x, y) vypočítat kloubové souřadnice a to tak, abychom minimalizovali vzdálenost všech kloubů 2,3,4 od definovaných bodových překážek.

Úlohu vyřešíme tak, že do vektoru \mathbf{f} přidáme další funkce, které budou „vhodně kódovat“ požadavek na maximalizaci vzdálenosti kloubů od bodových překážek. Toto můžeme realizovat např. tak, že použijeme převrácenou hodnotu vzdálenosti d :

$$f_{i+2} = \frac{1}{d_i + \epsilon}, \quad \text{kde} \\ d_i = (x_i - x_P)^2 + (y_i - y_P)^2, \quad i = 1, \dots, 3$$

Hodnota ϵ řeší problém dělení nulou.

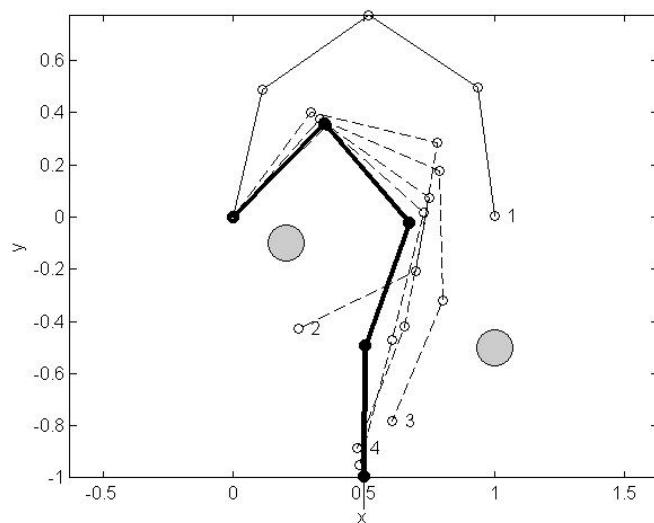
Pokud použijeme dvě bodové překážky, budeme mít $2 \times 3 = 6$ vzdáleností, bude platit $m = 2 + 6 = 8$ a jádro algoritmu bude vypadat takto¹⁵:

```
while err > 1e-6
    [x1,y1,x2,y2,x3,y3,x4,y4] = fkine(q)
    d1 = (x1-xP)^2 + (y1-yP)^2 % vzdálenost bodu P od kloubu
    1
    ...
    d4 = (x1-xR)^2 + (y1-yR)^2 % vzdálenost bodu R od kloubu
    1
    ...
    d6 = (x3-xR)^2 + (y3-yR)^2 % vzdálenost bodu R od kloubu
    3

    f = [x4 - x ; ... % poloha efektoru v ose x
          y4 - y ; ... % ... v ose y
          1/(d1+0.1); ... % prevrácena vzdalenost 1
          ...
          1/(d6+0.1)];
    J = jacobian(q1,q2,q3,q4) % numerický výpočet J
    Lambda = diag([0.1 0.1 100 100 100 100 100 100])
    q = q - J'*inv(J*J' + Lambda) * f
end
```

Konkrétní výsledek řešení inverzní kinematiky se zahrnutím překážek vidíme na obr. 2.14. Je vidět, že koncový efektor dosáhl přesně žádané polohy (x, y) a manipulátor se „hezky propletl“ mezi překážkami.

¹⁵ Kód by jistě šlo napsat elegantněji, nám jde nyní pouze o názornost a jednoduchost.



Obrázek 2.14 RRRR manipulátor s překážkou

2.6 Málo rovnic aneb $m < n$

V této sekci se stručně zamyslíme nad řešením problému soustavy lineárních nebo nelineárních rovnic pro $m < n$. Je zjevné, že problém má nekonečně mnoho řešení, v praxi však může být užitečné nalézt alespoň nějaké z nich.

Kouzlo myšlenky nejmenších čtverců je v tom, že ji lze použít na řadu různých úloh. Metoda hledá minimum součtu čtverců jednotlivých řádků vektoru funkcí $\mathbf{f}(\mathbf{x})$.

- Pokud existuje (jedno) přesné řešení $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, tak ho najde - to je případ $m = n$.
- Pokud neexistuje řešení, pak najde minimum - to je případ $m > n$.
- Pokud existuje nekonečně mnoho řešení ($m < n$), pak najde nějaké z nich (protože tím jednoduše minimalizuje kritérium nejmenších čtverců).

V případě $m < n$ použijeme formálně stejný přístup jako při řešení úloh $m > n$ s tím rozdílem, že budeme používat *PRAVOU pseudoinverzi*.

2.6.1 Příklad pro lineární rovnice: parabola dvěma body

[Ex.: NM05] Uvažujme velmi jednoduchou úlohu: jsou zadány dva body v rovině a cílem je najít parametry a , b a c paraboly $y = ax^2 + bx + c$, která těmito dvěma body prochází.

Zjevně má tato úloha nekonečně mnoho řešení (speciálním případem je přímka, tedy $a = 0$, jinak obecně různě „zahnuté“ paraboly).

Řešení je velmi podobné [příkladu 2.2.2](#). Řešíme soustavu dvou rovnic

$$\begin{aligned}y_1 &= ax_1^2 + bx_1 + c \\y_2 &= ax_2^2 + bx_2 + c\end{aligned}$$

o třech neznámých a , b a c .

Definice problému podle rovnice (2.5) je tato:

$$\mathbf{x} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \quad (2.43)$$

Řešení pomocí pravé pseudoinverze vypadá takto:

$$\mathbf{x} = \mathbf{A}^T (\mathbf{A} \mathbf{A}^T)^{-1} \mathbf{b} \quad (2.44)$$

Kód v MATLABu bude vypadat takto:

```
x = [0.3;1]
y = [0.7 ; 1.2]
A = [x(1)^2, x(1), 1;...
      x(2)^2, x(2), 1];
b = [y(1); y(2)];

parametry = A'*inv(A*A') * b
```

Každý zvídavý čtenář si může sám snadno vyzkoušet, že pro některé příklady zadaných bodů najde MATLAB řešení v podobě přímky ($a = 0$), pro jiné ale paraboly.

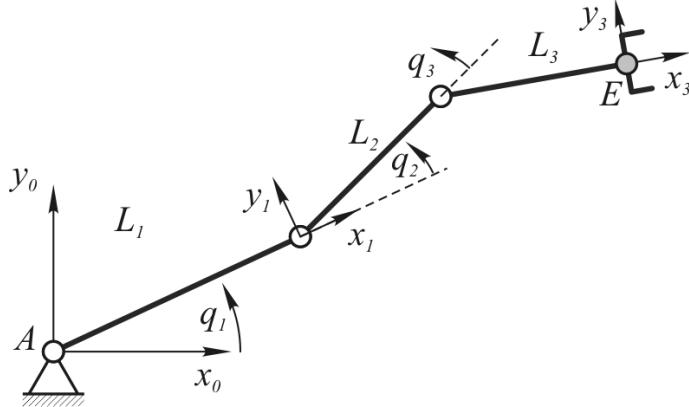
2.6.2 Příklad pro nelineární rovnice: redundantní RRR manipulátor

[Ex.: KD16]

Na obr. 2.15 je schéma rovinného manipulátoru se třemi klouby ($n = 3$). Z hlediska inverzní kinematiky nás ale v tomto příkladu budou zajímat pouze souřadnice x, y :

$$x = L_1 \cos(\vartheta_1) + L_2 \cos(\vartheta_1 + \vartheta_2) + L_3 \cos(\vartheta_1 + \vartheta_2 + \vartheta_3)$$

$$y = L_1 \sin(\vartheta_1) + L_2 \sin(\vartheta_1 + \vartheta_2) + L_3 \sin(\vartheta_1 + \vartheta_2 + \vartheta_3)$$



Obrázek 2.15 RRR manipulátor

Počet rovnic je $m = 2$, platí tedy, že $m < n$. Říkáme, že takový manipulátor je *redundantní*, má prostě manipulátor jeden kloub navíc (nadbytečný, redundantní), který by mít nemusel a přesto by bylo možné souřadnice x, y dosáhnout. Prakticky je jistě zajímavé a užitečné takový manipulátor provozovat, např. pokud má pracovat ve složitějším prostředí a vyhýbat se překážkám¹⁶.

Podobně jako v příkladu 2.4.4 v rovnici (2.30) přeformulujeme problém podle rovnice (2.26):

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\vartheta_1, \vartheta_2, \vartheta_3) \\ f_2(\vartheta_1, \vartheta_2, \vartheta_3) \end{bmatrix} = \begin{bmatrix} L_1 \cos(\vartheta_1) + \dots - x \\ L_1 \sin(\vartheta_1) + \dots - y \end{bmatrix} = \mathbf{0}, \quad (2.45)$$

kde $\mathbf{x} = [\vartheta_1, \vartheta_2, \vartheta_3]^T$ je vektor neznámých.

Vidíme tedy, že $m = 2$ a $n = 3$. Matematicky můžeme najít nekonečně mnoho řešení, prakticky nás zajímá *nějaké z nich*.

Jakobián bude obdélníkový:

$$\mathbf{J} = \begin{bmatrix} -L_1 s_1 - L_2 s_{12} - L_3 s_{123} & -L_2 s_{12} - L_3 s_{123} & -L_3 s_{123} \\ L_1 c_1 + L_2 c_{12} + L_3 c_{123} & L_2 c_{12} + L_3 c_{123} & +L_3 c_{123} \end{bmatrix} \quad (2.46)$$

kde např. c_{123} je zkrácený zápis pro $\cos(\varphi_1 + \varphi_2 + \varphi_3)$.

Pokud formulujeme problém takto, je při výpočtu nutné použít pravou inverzi (2.14) nebo rovnou LMa (kap. 2.5.3).

Jádro výpočtu bude vypadat následovně:

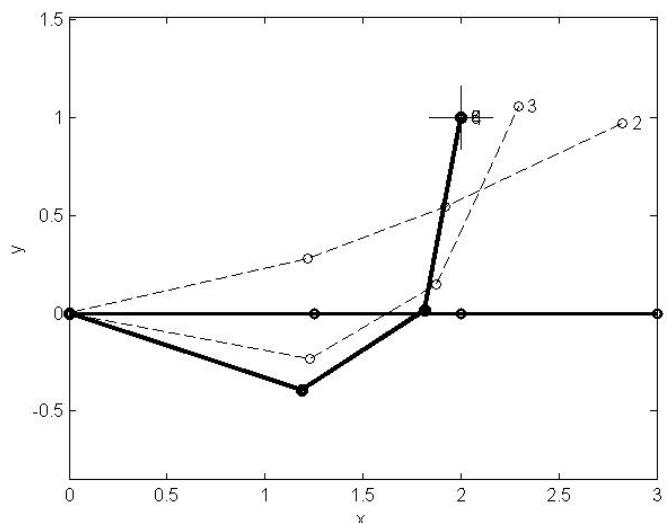
¹⁶ Lidská ruka (paže) také oplývá notnou dávkou redundancy – kdo někdy vyměňoval alternátor v autě nebo jinou součástku na nějakém nepřístupném místě o tom ví své.

```

while err < 1e-6
    f = [L1*cos(q1) + ... ];
    J = [-L1*sin(q1)- ...];
    q = [q1;q2;q3] - J'*inv(J*J' + 0.1*eye(2)) * f; % LMa
    q1 = q(1), q2 = q(2), q3 = q(3);
    err = norm(f);
end

```

Na obr. 2.16 vidíme konvergenci algoritmu k cílové poloze. Záměrně jsme použili na počátku singulární polohu manipulátoru ($\vartheta_1 = \vartheta_2 = \vartheta_3 = 0$, abychom ukázali, že si s tím LMa poradí.



Obrázek 2.16 Ukázka konvergence
LMa pro RRR manipulátor

2.7 Otázky a neřešené úlohy

Otázky

1. Odvodte Newtonovu metodu hledání kořene funkce jedné proměnné $y = f(x)$. Zakerďte obrázek, zakótujte a zapište rovnici pro jeden krok metody.
- 2.

Úlohy

3. Naprogramujte řešení hledání parametru a v rovnici $y = \sin(at)$:
 - a) Definujte časový vektor t a vypočtěte hodnoty funkce y .
 - b) K vypočteným hodnotám přidejte šum s normálním rozdělením, nulovou střední hodnotou a směr. odchylk. σ . Takto vytvořená data budeme považovat za data měření.
 - c) Na základě znalosti naměřených dat a tvaru funkce $y = \sin(at)$ naprogramujte výpočet odhadu parametru a pomocí NLS (bez využití funkce lsqnonlin nebo jiných hotových funkcí MATLABu).
 - d) Otestujte konvergenci metody při různém počátečním odhadu a_0 . V jakém přibližně rozmezí metoda konverguje ke správné hodnotě a ?

3 Numerické řešení ODE

V této kapitole se

- zamyslíme nad podstatou řešení ODE a na příkladu jedné ODE si ukážeme proč je řešením exponenciála,
 - toto zjištění zobecníme pro soustavu několika ODE vyjádřených pomocí maticy **A** a vyřešíme (homogenní soustavu) pomocí maticové exponenciály,
 - seznámíme se s řešením nelineárních ODE pomocí numerické integrace,
 - řekneme si několik (důležitých) detailů a
 - vše budeme demonstrovat na příkladech.
-

Obsah kapitoly:

3.1	Co je vlastně řešením ODE?	37
3.2	Maticová exponenciála – řešení lineární ODE	39
3.2.1	Řešení homogenní soustavy ODE pomocí mat. exponenciály	39
3.2.2	Výpočet maticové exponenciály (ME)	39
3.2.3	Příklad: Volné kmitání mechanického oscilátoru	39
3.2.4	Řešení soustavy ODE s pravou stranou	40
3.2.5	Příklad: Oscilátor se vstupem	40
3.3	Diskretizace – řešení lineární ODE	42
3.3.1	Diskretizace spojitého LTI systému	42
3.3.2	Příklady diskretizace	43
3.3.3	Lineární simulace lsim v MATLAB/Control Toolbox	45
3.4	Metody numerické integrace	47
3.4.1	Co to je numerická integrace?	47
3.4.2	Formulace úlohy	47
3.4.3	Euler, lichoběžník a pánové Carl David Tolmé Runge a Martin Wilhelm Kutta	48
3.4.4	Explicitní vs. implicitní metody	49
3.5	Řešiče (solvery) ODE v MATLABu a Simulinku	52
3.5.1	Variabilní délka kroku	52
3.5.2	Stiff systém	54
3.5.3	Tvary soustavy ODE	59
3.5.4	Přehled řešičů (solverů) v MATLABu a Simulinku	59
3.6	Příklady řešení ODE v MATLABu a Simulinku	62
3.6.1	Mechanický oscilátor 1dof	62
3.6.2	Příklad: Mechanický oscilátor 2dof	64
3.6.3	Příklad: Mostový jeřáb (2dof)	67
3.7	Otázky a neřešené úlohy	70

3.1 Co je vlastně řešením ODE?

Jednoduchou a přitom velice prakticou ODE je:

$$\tau \dot{x} + x = ku, \quad (3.1)$$

kde x je stavová proměnná a τ, k jsou konstanty. Popisuje dynamiku tzv. *systému prvního řádu* a velmi dobře modeluje chování řady reálných systémů.

V této sekci se zamyslíme nad podstatou *analytického* řešení této rovnice. Někdy se tato rovnice zapisuje ve tvaru

$$\dot{x} = ax + bu, \quad (3.2)$$

kde a, b jsou konstanty. Uvažujme nyní $a = -1$ a nulový vstup, dostaneme rovnici ve tvaru:

$$\dot{x}(t) + x(t) = 0 \quad (3.3)$$

neboli

$$\dot{x}(t) = -x(t). \quad (3.4)$$

Abychom zdůraznili to, že x a \dot{x} se mění v čase, zapsali jsme je jako funkce času.

Otzázkou: Co znamená úloha *nalézení řešení této rovnice (3.4)?*

Co je vlastně řešením ODE?

Odpověď: Odpověď na tuto hádanku je tato: musíme nalézt takovou funkci $x(t)$, pro kterou v každém bodě t platí, že její derivace $\dot{x}(t)$ je rovna záporné funkční hodnotě, tedy $-x(t)$ (přesně tuto informaci vyjadřuje rovnice (3.4)).

To není tak těžké, pokud si vzpomeneme na základní pravidla derivování a funkci e^y ¹⁷.

Nyní už řešení odhalíme snadno:

$$x(t) = ce^{-t}, \quad (3.5)$$

kde c je nějaká konstanta. Po chvíli přemýšlení budeme souhlasit s tím, že v tomto případě $c = x_0$ (počáteční hodnota proměnné x). Dostáváme tedy výsledné řešení ve tvaru:

$$x(t) = x_0 e^{-t}, \quad (3.6)$$

Funkce e^y je taková funkce, jejíž derivace v bodě y je rovna funkční hodnotě v tomto bodě.

Podobně můžeme hádat dál. Rovnici

$$\tau \dot{x}(t) + x(t) = 0 \quad (3.7)$$

převedeme na tvar

$$\dot{x}(t) = -\frac{1}{\tau}x(t) = ax(t) \quad (3.8)$$

a řešení pak snadno uhodneme ve tvaru:

¹⁷ Záměrně zde píšeme e^y a nikoli e^x jak je obvyklé. Chceme tím zabránit záměně s proměnnou x , která se vyskytuje v okolních rovnicích.

$$x(t) = x_0 e^{-\frac{1}{\tau}t} \quad \text{nebo} \quad x(t) = x_0 e^{at} \quad (3.9)$$

Pokud zahrneme konstantní vstup do systému (tj. konstantní pravou stranu), má rovnice tuto podobu:

$$\tau \dot{x}(t) + x(t) = k, \quad (3.10)$$

kde k je konstantní vstup do systému. Rovnici převedeme na tvar:

$$\dot{x}(t) = -\frac{1}{\tau}x(t) + \frac{k}{\tau} \quad (3.11)$$

a uhádneme řešení ve tvaru:

$$x(t) = (x_0 - k)e^{-\frac{1}{\tau}t} + k \quad (3.12)$$

Řešení ODE jsme odhalili „hádáním“, což se v případě složitějších rovnic už moc dařit nebude.

Každý si samozřejmě může dohledat v příslušných partiích matematiky ty správné postupy řešení lineárních ODE, smyslem těchto dvou sekcí bylo ukázat, že ...se postupu řešení dá v případě jednoduché rovnice i snadno rozumět.

Otzázkы, úvahy a úkoly

- Nakresli do grafu funkci (3.6) pro několik různých počátečních podmínek x_0 .
- Dokaž správnost odvození řešení (3.12) a nakresli graf funkce pro hodnoty $k > x_0$ i $k < x_0$.

3.2 Maticová exponenciála – řešení lineární ODE

3.2.1 Řešení homogenní soustavy ODE pomocí mat. exponenciály

Zústaneme ještě chvíli u řešení soustavy lineárních ODE.

V předchozím textu jsme „uhodli“ řešení rovnice (3.8)

$$\dot{x}(t) = ax(t) \quad (3.13)$$

ve tvaru

$$x(t) = x_0 e^{at} \quad (3.14)$$

Toto zjištění lze pro soustavu ODE zobecnit. Soustavu ODE zapíšeme takto:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) \quad (3.15)$$

Jejím řešením je pak

$$\mathbf{x}(t) = e^{\mathbf{At}}\mathbf{x}_0, \quad (3.16)$$

kde $e^{(\cdot)}$ je tzv. *maticová exponenciála*.

3.2.2 Výpočet maticové exponenciály (ME)

Základní vzorec pro výpočet ME je:

$$e^{\mathbf{M}} = \sum_{k=0}^{\infty} \frac{1}{k!} \mathbf{M}^k \quad (3.17)$$

Pro tvar rovnice (3.16) můžeme výpočet ME zapsat takto:

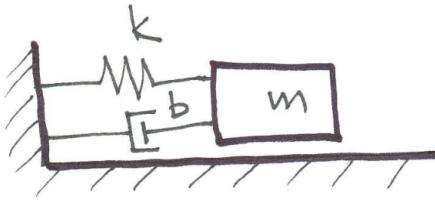
$$e^{\mathbf{At}} = \mathbf{E} + t\mathbf{A} + \frac{t^2\mathbf{A}^2}{2!} + \dots, \quad (3.18)$$

kde \mathbf{E} je jednotková matice. C. Moler a Ch. van Loan popisují ve velice zajímavém článku [[Moler:2003:NDW](#)] detailně problematiku výpočtu ME a implementaci funkce `expm` v MATLABu¹⁸.

3.2.3 Příklad: Volné kmitání mechanického oscilátoru

Řešení homogenní ODE (s nulovou pravou stranou, tedy bez vstupu) si ukážeme na příkladu hmoty s pružinou a tlumičem (obr. 3.1).

¹⁸ Cleve Moler je jedním ze tří zakladatelů firmy Mathworks. V článku [[Moler:2003:NDW](#)] je na 46 stranách rozebráno 19 způsobů výpočtu ME. Předtím je na názorném příkladu ukázáno, proč je tento výpočet obecně výzvou. Na konci článku na str. 40 je pak uvedeno, že funkce `expm` byla mezi 80 funkcemi, které byly implementovány v první verzi MATLABu na konci 70 let. Moler zde také uvádí, že právě ME mohla výrazným způsobem přispět k rozšíření a oblíbenosti MATLABu v komunitě inženýrů a výzkumníků zabývajících se teorií (a praxí) řízení systémů – tedy mimo oblast numerické lineární algebry.



Obrázek 3.1 Schéma mechanického oscilátoru s tlumením bez vstupu

Známou rovnici

$$m\ddot{x} + b\dot{x} + kx = 0 \quad (3.19)$$

přepíšeme do maticového tvaru = převedeme ji na dvě rovnice prvního rádu:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k/m & -b/m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad (3.20)$$

kde $x_1 = x$, $x_2 = \dot{x}$.

Řešení pomocí maticové exponenciály v MATLABu bude vypadat takto:

```
k = 100, b = 2, m = 3;
A = [0, 1; -k/m, -b/m];
t = 0:0.01:10;
x0 = [1;0];
for k = 1:length(t)
    x(:,k) = expm(A*t(k))*x0;
end
plot(t,x(1,:),'r-')
```

3.2.4 Řešení soustavy ODE s pravou stranou

Pokud má rovnice nenulovou pravou stranu (nehomogenní rovnice) znamená to z hlediska teorie řízení nebo systémů, že má systém nějaký vstup. Tento vstup je obecně nějaký časově proměnný vektor $\mathbf{u}(t)$, jeho vliv na derivace jednotlivých stavových proměnných pak vyjadřuje matice \mathbf{B} .

Soustavu ODE pak zapíšeme ve tvaru stavového modelu:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \quad (3.21)$$

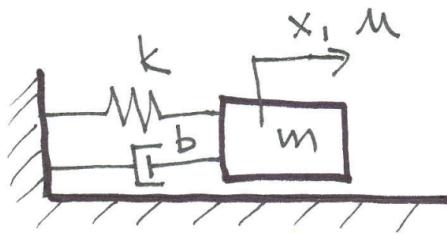
Bez důkazu a odvození uvedeme řešení (metoda variace parametrů):

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}_0 + \int_0^t e^{\mathbf{A}(t-w)}\mathbf{B}\mathbf{u}(w)dw \quad (3.22)$$

3.2.5 Příklad: Oscilátor se vstupem

Stavový model vylepšíme o vstup u (jeden vstup):

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k/m & -b/m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t). \quad (3.23)$$



Obrázek 3.2 Schéma mechanického oscilátoru se vstupem

Pomocí rovnice (3.22) vypočítáme odezvu.

- **Konstantní vstup** – pokud na soustavu působí konstantní vstup c , můžeme zapsat že $\mathbf{Bu} = \begin{bmatrix} 0 \\ c \end{bmatrix}$. Integrál v rovnici (3.22) pak umíme spočítat analyticky takto:

$$\begin{aligned}
 & \int_0^t e^{(t-w)\mathbf{A}} \mathbf{Bu}(w) dw = \\
 &= \int_0^t e^{(t-w)\mathbf{A}} \begin{bmatrix} 0 \\ c \end{bmatrix} dw = \int_0^t e^{(w)\mathbf{A}} \begin{bmatrix} 0 \\ c \end{bmatrix} dw = \\
 &= \mathbf{A}^{-1} (e^{t\mathbf{A}} - \mathbf{E}) \begin{bmatrix} 0 \\ c \end{bmatrix} \quad (3.24)
 \end{aligned}$$

Obecný vstup – v tomto případě nelze určitě řešení v uzavřeném tvaru, musíme numericky integrovat výraz

$$\int_0^t e^{\mathbf{A}(t-w)} \mathbf{Bu}(w) dw.$$

V příkladu MD02 je integrace implementována příkazem trapz:

```

for k = 1:length(t)
    for j = 1:k
        p(:,j) = expm(A*(t(k) - t(j)))*B*u(j);
    end
    x(:,k) = expm(A*t(k))*x0 + trapz(p,2)*Ts;
end

```

Otázky, úvahy a úkoly

- Zkontroluj platnost rovnice (3.22) třeba na příkladu mechanického oscilátoru, pro srovnání použij funkci lsim(Control toolbox) nebo numerickou integraci.

3.3 Diskretizace – řešení lineární ODE

V předchozí části 3.2 jsme ukázali, že soustavu lineárních ODE lze řešit pomocí maticové exponenciály. V této části si ukážeme, že lze řešení provést i tak, že soustavu ODE (dynamika systému je vyjádřena ve spojitém čase) nejprve převedeme do diskrétního tvaru (dynamika systému je vyjádřena v jednotlivých časových okamžicích) a pak už jen vyčíslíme výsledek.

3.3.1 Diskretizace spojitého LTI systému

Uvažujme lineární časově invariantní (LTI) systém ve tvaru stavového modelu:

$$\dot{\mathbf{x}}(t) = \mathbf{Ax}(t) + \mathbf{Bu}(t)$$

$$\mathbf{y}(t) = \mathbf{Cx}(t)$$

Tento model lze vždy *přibližně* nahradit diskrétním tvarem [[Valasek1995](#)](str. 95):

$$\mathbf{x}_{k+1} = \Phi \mathbf{x}_k + \Gamma \mathbf{u}_k$$

$$\mathbf{y}_{k+1} = \mathbf{C} \mathbf{x}_{k+1}$$

kde T_s je perioda vzorkování, \mathbf{E} je jednotková matice a

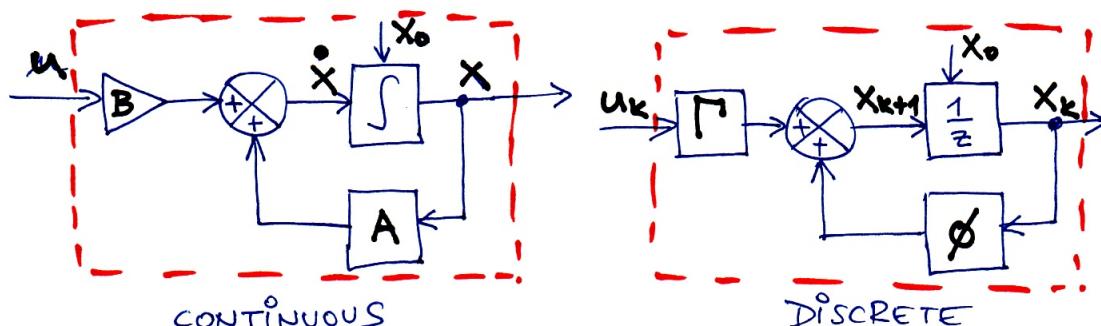
$$\Phi = \mathbf{E} + \mathbf{AT}_s + \frac{\mathbf{A}^2 T_s^2}{2!} \quad (3.25)$$

$$\Gamma = (\mathbf{ET}_s + \frac{\mathbf{AT}_s^2}{2!}) \mathbf{B} \quad (3.26)$$

V MATLABu lze tuto transformaci provést pomocí funkce `c2d`.

Pokud bychom chtěli použít jednodušší tvar, můžeme se vrátit k panu Eulerovi:

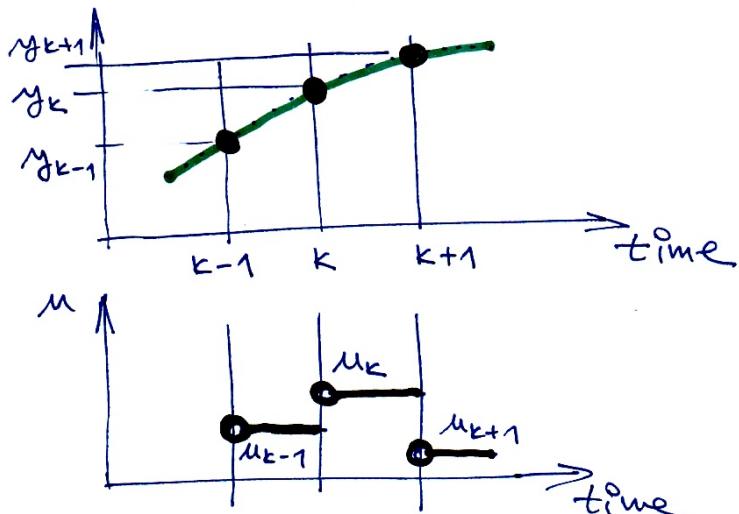
$$\begin{aligned} \Phi &= \mathbf{E} + \mathbf{AT}_s \\ \Gamma &= T_s \mathbf{B} \end{aligned} \quad (3.27)$$



Obrázek 3.3 Spojitý a diskrétní stavový model

Pro převod spojitého na diskrétní popis existuje několik různých metod, pro případ schodovitého vstupního signálu u je nevhodnější metoda ZOH (Zero-Order Hold) [[matlab_help](#)]¹⁹.

¹⁹ Systémy, kde uvažujeme schodovitý vstupní signál, jsou pro nás nejzajímavější, protože při praktické realizaci řízení počítačem/kontrolérem je skutečně akční signál po dobu vzorkování T_s konstantní.



Obrázek 3.4 Označení veličin
a ZOH reprezentace vstupu u

3.3.2 Příklady diskretizace

3.3.2.1 Systém prvního řádu

Jako nejjednodušší příklad opět použijeme dynamický systém prvního řádu popsaný rovnicí

$$\tau \dot{x} + x = u, \quad (3.28)$$

kterou upravíme na tvar

$$\dot{x} = -\frac{1}{\tau}x + \frac{1}{\tau}u. \quad (3.29)$$

Vidíme tedy, že matice \mathbf{A} a \mathbf{B} LTI modelu mají tvar skalárů:

$$\mathbf{A} = -\frac{1}{\tau}, \quad \mathbf{B} = \frac{1}{\tau}. \quad (3.30)$$

[Ex.: NM16] **Diskretizace 1.řád (Euler)** – nejprve ukážeme tvar diskretizovaného modelu podle rovnic 3.27:

$$\Phi = 1 - \frac{T_s}{\tau}, \quad \Gamma = \frac{T_s}{\tau}. \quad (3.31)$$

- **Diskretizace 2. řád** – pokud přidáme další člen Taylorova rozvoje (rovnice ??) dostaneme tvar:

$$\Phi = 1 - \frac{T_s}{\tau} + \frac{T_s^2}{2\tau^2}, \quad \Gamma = \frac{T_s}{\tau} - \frac{T_s^2}{2\tau^2}. \quad (3.32)$$

- **Diskretizace podle Eulera (přímé odvození)** – pro srovnání se ještě podíváme na diskretizaci z druhé strany – přes náhradu derivace konečnou diferencí:

$$\dot{x} = \frac{x_{k+1} - x_k}{T_s}. \quad (3.33)$$

Dosadíme tedy do 3.28 a dostáváme:

$$\frac{x_{k+1} - x_k}{T_s} = -\frac{1}{\tau}x_k + \frac{1}{\tau}u_k. \quad (3.34)$$

Po úpravě:

$$x_{k+1} = (1 - \frac{T_s}{\tau})x_k + \frac{T_s}{\tau}u_k. \quad (3.35)$$

Vidíme tedy, že odvozený tvar 3.35 odpovídá tvaru 3.31.

3.3.2.2 Mechanický oscilátor a linearizované kyvadlo

[Ex.: NM17] Hmota s pružinou a tlumičem i matematické kyvadlo, u kterého nahradíme $\sin \varphi = \varphi$ mají formálně stejnou ODE²⁰:

$$\ddot{q} = -a_1 q - a_2 \dot{q} + cu \quad (3.36)$$

Tuto rovnici přepíšeme do tvaru spojitého lineárního stavového modelu:

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 1 \\ -a_1 & -a_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{c} \end{bmatrix} u \quad (3.37)$$

Pro diskretizaci použijeme rovnic (3.25)–(3.26) a dostaneme:

$$\Phi = \begin{bmatrix} 1 - \frac{T_s^2 a_1}{2} & T_s - \frac{T_s^2 a_1}{2} \\ \frac{a_1 a_2 T_s^2}{2} - a_1 T_s & \frac{-a_2^2 + a_1}{2} T_s^2 - T_s a_2 \end{bmatrix} \quad (3.38)$$

$$\Gamma = \begin{bmatrix} \frac{c T_s^2}{2} \\ c(T_s - \frac{T_s^2 * a_2}{2}) \end{bmatrix} \quad (3.39)$$

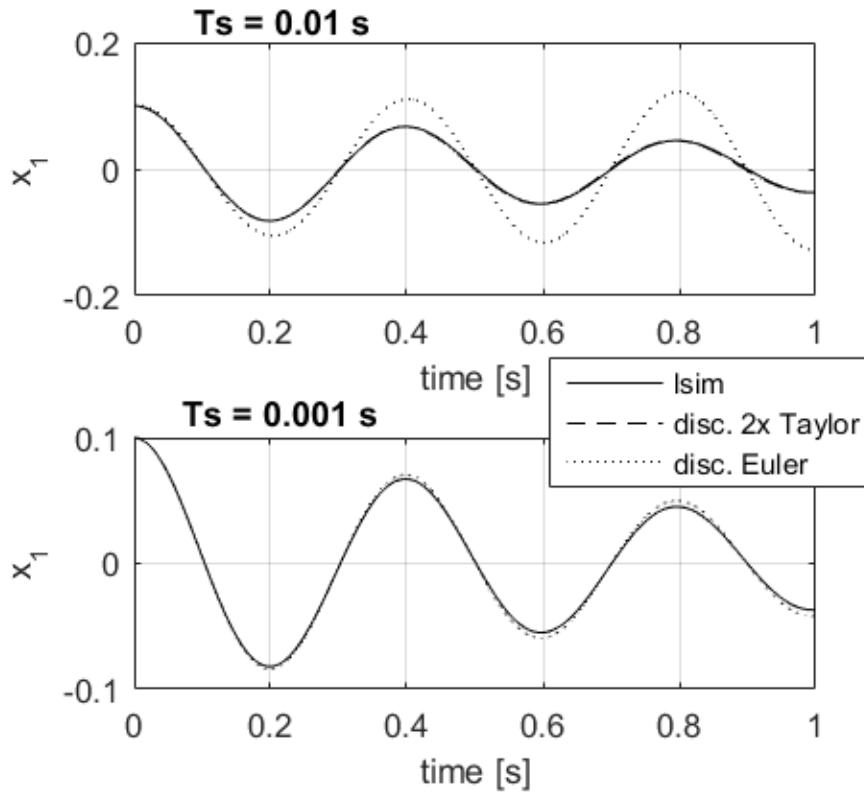
Pokud použijeme pouze Eulerovu approximaci, pak:

$$\Phi = \begin{bmatrix} 1 & T_s \\ -a_1 T_s & 1 - a_2 T_s \end{bmatrix} \quad (3.40)$$

$$\Gamma = \begin{bmatrix} 0 \\ c T_s \end{bmatrix} \quad (3.41)$$

Na obr. 3.5 vidíme, že diskretizace podle Eulera moc přesná není. V případě lineárního systému není žádný důvod ji používat, naproti tomu použití dvou členů Taylorova rozvoje (rovn. (3.38)–(3.39)) je dostačující.

²⁰ Pružina s tlumičem: $m\ddot{x} = -kx - bx + u$
a tedy: $a_1 = k/m$; $a_2 = b/m$; $c = 1/m$. Kyvadlo: $mL^2\ddot{\varphi} = -mgL\varphi - bx + u$
a tedy: $a_1 = g/L$; $a_2 = b/(mL^2)$; $c = 1/(mL^2)$.



Obrázek 3.5 Porovnání diskretizace s použitím dvou členů Taylorova rozvoje (rovn. (3.38)–(3.39)) a pouze jednoho člena, tedy Eulerovy metody (rovn. (3.40)–(3.41)). Horní a dolní výsledek se liší použitým vzorkováním T_s .

3.3.3 Lineární simulace `lsim` v MATLAB/Control Toolbox

Celá kapitola 3 se věnuje otázce hledání řešení ODE. První 4 podkapitoly (včetně této) se omezují na lineární ODE.

Nejstručnější možná odpověď na tuto otázku „jak najít řešení lineární ODE“ přitom je: `lsim`.

`lsim` je příkaz v Control Toolboxu MATLABu, který LTI spojitý stavový model řeší pomocí diskretizace²¹.

Použití (systém 1. řádu) vypadá takto:

```
tau = 2; x0 = 0;
u_const = 4;
Ts = 0.1; % perioda vzorkovani
t = 0:Ts:10;
u = t*x0 + u_const;
A = -1/tau; B = 1/tau; C = 1; D = [] ; % matice systemu
sys = ss(A,B,C,D); % spojity cas
```

²¹ Citace z [`<matlab_help>`]: Continuous-time systems are discretized with `c2d` using either the '`zoh`' or '`foh`' method ('`foh`' is used for smooth input signals and '`zoh`' for discontinuous signals such as pulses or square waves). The sample time is set to the spacing `dt` between the user-supplied time samples `t`.

```
[y,t,x] = lsim(sys, u, t, x0)
plot(t,y,'k-')
```

[Ex.: NM16]

V příkladu NM16 může nadšený čtenář najít detailní porovnání všech metod diskretizace s přesným analytickým řešením.

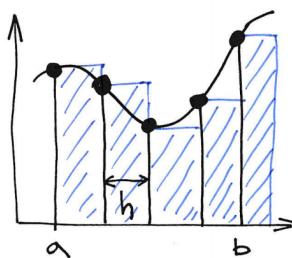
3.4 Metody numerické integrace

V předchozím textu jsme se zabývali analytickým řešením *lineární ODE*, kdy získáme přesné řešení. Většina praktických problémů je ale popsána nelineárními ODE a přesné řešení nelze najít²². Musíme se spokojit s řešením přibližným, které vypočteme metodami numerické integrace.

3.4.1 Co to je numerická integrace?

Už z mateřské školky víme, že *integrace je výpočet plochy pod grafem funkce*. Pokud má integrovaná funkce nějaký hezký tvar (což v praktickém životě nikdy nemá²³), můžeme ji zintegrovat analyticky.

Numericky lze zintegrovat jakoukoli funkci, musíme se ale spokojit s přibližným řešením. Numerická integrace funguje tak, že hodnotu funkce vyčíslíme pouze v určitých diskrétních hodnotách a funkční hodnotu mezi body *nějak odhadneme*. Zcela nejjednodušší odhad je, že předpokládáme hodnotu mezi body za konstantní. Takto funguje Eulerova metoda na obr. 3.6



Obrázek 3.6 Eulerova metoda integrace – určitý integrál mezi hodnotami a a b .

Je zřejmé, že Eulerovy obdélníčky na obrázku velice nepřesně approximují původní funkci. Nabízí se dvě cesty jak to celé vylepšit: a) zmenšit krok h (použít pro approximaci více bodů); b) nahradit obdélník něčím lepším (třeba aspoň lichoběžníkem = spojit body).

V dalším textu se budeme zabývat tím, jak i tato vylepšení použít pro řešení ODE.

3.4.2 Formulace úlohy

Problém je definován tak, že máme zadánu jednu nelineární ODE a počáteční podmínky takto:

$$\dot{x}(t) = f(t, x)$$

$$t_0 = 0$$

$$x(t_0) = x_0.$$

Konkrétním příkladem může být třeba rovnice:

²² V dnešní době řešíme numericky i rovnice, které by analyticky vyřešit šly – jen to neumíme, nemáme na to čas nebo se nám jednoduše nechce.

²³ Analytické řešení integrálů obskurně speciálního tvaru považuji za jednu z nejméně šťastných partií matematiky na VŠ. Nepřináší nic: ani užitek, ani rozvoj představivosti, ani intelektuální zábavu.

$$\dot{x}(t) = 2x^3 + \sin(5t), \quad x_0 = 0, 7. \quad (3.42)$$

Místo spojitého času budeme ODE řešit v diskrétních časových okamžicích $0, 1, \dots, k, k+1, \dots$. Výpočetní algoritmus pak bude vypadat takto:

$$x_{k+1} = x_k + \Delta x, \quad (3.43)$$

kde k je současný krok výpočtu a Δx je odhad přírůstku stavu x .

3.4.3 Euler, lichoběžník a pánové Carl David Tolmé Runge a Martin Wilhelm Kutta

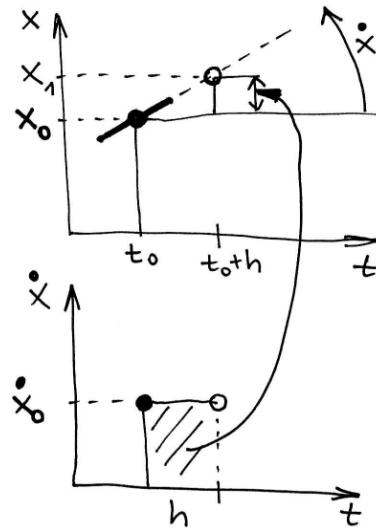
Nejjednodušší diskretizace rovnice ?? je nahrazení derivace v časovém kroku k diferencí takto:

$$\dot{x} = \frac{x_{k+1} - x_k}{h} \quad (3.44)$$

Po úpravě dostaneme *Eulerovu metodu integrace* ODE:

$$x_{k+1} = x_k + h\dot{x}_k \quad (3.45)$$

Vidíme, že člen $\dot{x}_k h$ je opět oním nepřesným obdélníčkem.



Obrázek 3.7 Eulerova metoda integrace

Jasnou výhodou Eulerovy metody je jednoduchost a rychlosť výpočtu²⁴. Pro přesné simulace a analýzy na PC budeme hledat přesnější approximaci.

Lichoběžníková metoda je příkladem vícekrokové metody:

$$x_{k+1} = x_k + h \frac{\dot{x}_k + \dot{x}_{k-1}}{2} \quad (3.46)$$

²⁴ Eulerova metoda se používá např. pro výpočty na platformách s omezeným výpočetním výkonem, dnes především na embedded systémech.

Hodnotu derivace počítá jako průměr z dvou přechozích kroků, neboli ji approximuje lineárně (lichoběžník). Na začátku vyžaduje jednokrokový startér, což bývá většinou Euler.

*Metoda Runge-Kutta*²⁵ je jednokroková metoda, která vyčísluje funkční hodnotu derivace $f(t, x)$ celkem ve čtyřech bodech. Tyto hodnoty následně váženě zprůměruje:

$$x_{k+1} = x_k + h \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} \quad (3.47)$$

$$k_1 = f(t_k, x_k)$$

$$k_2 = f\left(t_k + \frac{h}{2}, x_k + k_1 \frac{h}{2}\right)$$

$$k_3 = f\left(t_k + \frac{h}{2}, x_k + k_2 \frac{h}{2}\right)$$

$$k_4 = f(t_k + h, x_k + k_3 h)$$

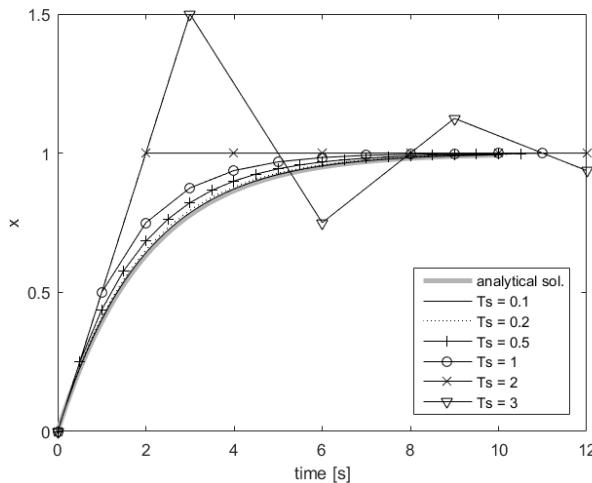
3.4.4 Explicitní vs. implicitní metody

Metody uvedené v předchozím textu (Euler 3.45 i RK 3.47 patří mezi tzv. *explicitní metody integrace* – hodnotu následujícího kroku vypočteme z hodnot stavů a vstupů v minulém kroku (jednokroková metoda) nebo minulých krocích (vícekroková metoda).

Na obr. 3.8 vidíme výsledek jednoduchého experimentu, kdy integrujeme Eulero-vou metodou rovnici

$$2\dot{x} + x = 1, \quad (3.48)$$

a testujeme přitom rozdílnou velikost integračního kroku h . Je zřejmé, že přesnost integrace klesá s velikostí kroku (což lze očekávat) a také, že od určité²⁶ velikosti kroku začne řešení oscilovat (což už bychom čekat nemuseli). Eulerova metoda se tedy stane nestabilní.



Obrázek 3.8 Explicitní Eulerova metoda – výsledek testu různé velikosti kroku integrace

²⁵ Konkrétně zde uvádíme nejznámější variantu RK4 z rodiny RK metod.

²⁶ Můžeme si všimnout, že do hodnoty cca $h = 0,2$ je výsledek uspokojivý. Lze tedy konstatovat, že pro maximální velikost kroku integrace můžeme použít podobnou poučku jako pro vzorkování při měření atd.: měla by být alespoň 10x menší než časová konstanta systému.

Připomeňme, že problém je při použití (explicitní) Eulerovy metody formulován takto:

$$\dot{x}_k = \frac{1}{2}(1 - x_k)$$

$$x_{k+1} = x_k + h\dot{x}_k,$$

takže celý problém se dá přepsat jako:

$$x_{k+1} = x_k + h \frac{1}{2}(1 - x_k). \quad (3.49)$$

Implicitní Eulerova metoda počítá následující krok z hodnoty derivace v následujícím kroku:

$$x_{k+1} = x_k + h\dot{x}_{k+1} \quad (3.50)$$

Když ale opět přepíšeme celý problém do jedné rovnice, dostaneme

$$x_{k+1} = x_k + h \frac{1}{2}(1 - x_{k+1}). \quad (3.51)$$

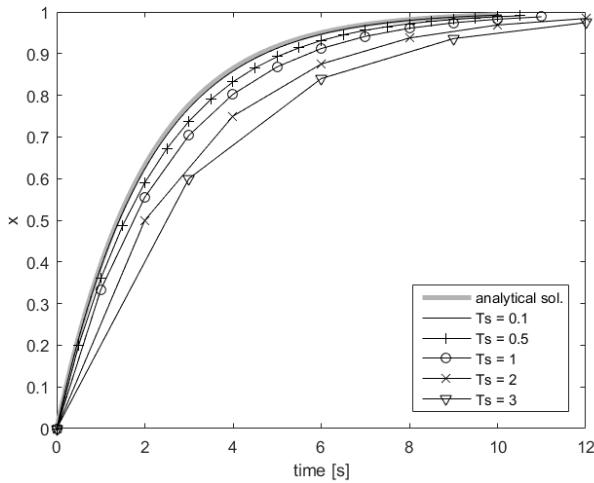
Hodnota x_{k+1} je tedy v této rovnici implicitně vyjádřená. To ale obecně²⁷ znamená, že musíme použít nějakou numerickou iterační metodu pro její výpočet!

Příklad řešení této úlohy v MATLABu následuje:

```
function ImplicitEulerExample
clc, clear
tau = 2; Ts = 0.1;
x(1) = 0;
t(1) = 0;
for i = 2:100
    x_prev = x(i-1);
    x(i) = fzero(@FZeroEquation, x_prev);
    t(i) = t(i-1) + Ts;
end
plot(t,x)
function F = FZeroEquation(x)
    F = x - x_prev - Ts*derivative(x);
end
function dx = derivative(x)
    dx = 1/tau*(-x + 1)
end
end
```

Na obr. 3.9 pak vidíme, že použití implicitního vyjádření integrační metody může mít smysl. Metoda zůstává stabilní i při velkém kroku integrace.

²⁷ V tomto konkrétním případě můžeme jistě snadno z rovnice ručně vyjádřit x_{k+1} , obecně ale používáme komplikovanější metody integrace než Eulerovu, rovnice je složitější a často i nelineární, takže by vyjádření x_{k+1} bylo buď velmi pracné nebo je zcela nemožné.



Obrázek 3.9 Explicitní Eulerova metoda – výsledek testu různé velikosti kroku integrace

V popsaném příkladu integrace *jedné* rovnice 3.48 jsme hledali řešení jedné algebraické rovnice (v MATLABu implementováno v příkazu fzero). V obecném případě, kdy integrujeme soustavu ODE, musíme tedy hledat řešení soustavy algebraických rovnic, což je v MATLABu implementováno v příkazu fsolve (Optimization Toolbox).

3.5 Řešiče (solvery) ODE v MATLABu a Simulinku

V předchozí sekci jsme popsali základní numerické metody integrace ODE (Euler, RK4), zmínili jsme se o stabilitě řešení a popsali jsme implicitní metody integrace. Pro praktické použití jsou tyto metody doplněny dalšími vlastnostmi, algoritmy a vylepšeními a jsou takto v MATLABu a Simulinku implementovány jako *řešiče (solvery)* ODE.

Jistě není náhoda ani rozmar programátorů v TMW, že jich je několik – kdyby existoval jeden správný nejlepší ideální optimální řešič, nepotřebovali bychom ty ostatní. Realita je taková, že na různé problémy fungují nejlépe různé řešiče. Přitom neexistuje nějaký obecný návod, který z nich pro daný problém zvolit.

Při provádění simulací (tedy řešení ODE) je zásadně důležité umět kriticky posoudit výsledek řešení, zvolit vhodný řešič a rozumně nastavování jeho parametrů.

K tomu je nutné vědět proč používáme *variabilní krok integrace*, co to je *Zero Crossing Detection* a *algebraická smyčka*, co to je jak se projevuje *stiff systém* a jaké máme možné *tvary soustavy ODE*.

Na konci této sekce si pak shrneme dostupné řešiče v MATLABu a Simulinku.

3.5.1 Variabilní délka kroku

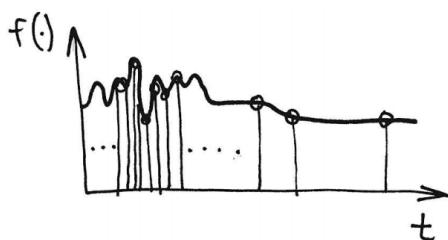
V předchozím oddíle jsme popsali Eulerovu, lichoběžníkovou a RK metodu numerické integrace a vždy jsme uvažovali, že krok integrace h bude konstantní. Existují ale nejméně tři dobré důvody, proč je při integraci ODE vhodné volit různou (variabilní) délku kroku h :

- přizpůsobení délky kroku dynamice soustavy
- detekce průchodu stavové proměnné x nulou (Zero Crossing Detection)
- algebraická smyčka.

Těm se nyní budeme stručně věnovat.

3.5.1.1 Rozdílná dynamika chování v různém čase

Když se podíváme na příklad funkce na obr. 3.10 vidíme, že může být výhodné délku kroku měnit podle dynamiky systému (podle toho, jak moc je integrovaná funkce „zvlněná“).



Obrázek 3.10 Ukázka funkce, když je výhodné použít variabilní krok integrace

Při zachování stejné přesnosti vypočteme integrál nižším počtem kroků.

3.5.1.2 Detekce přesného průchodu nulou (Zero Crossing Detection)

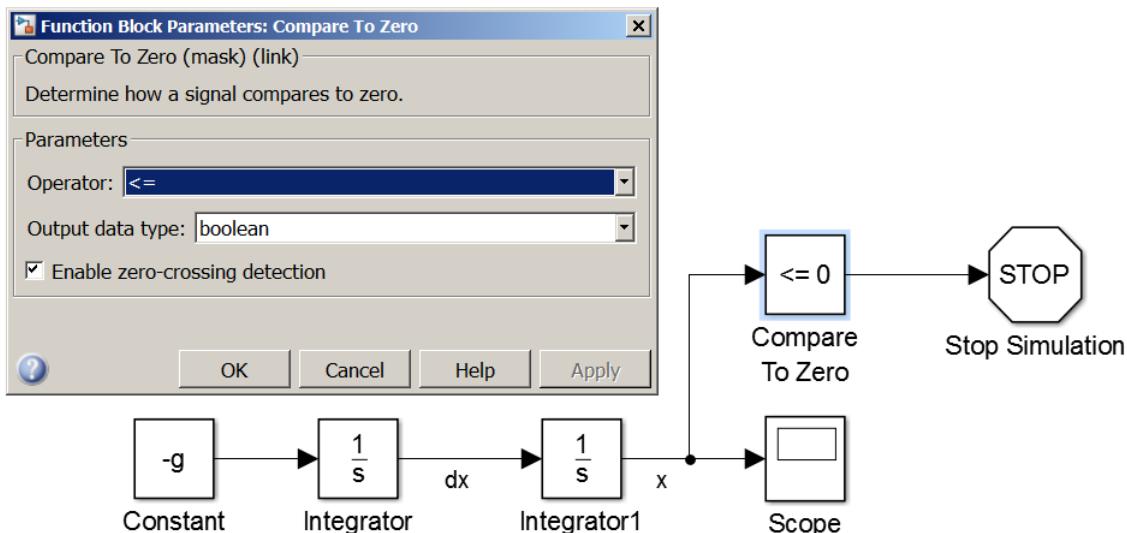
V některých simulačních úlohách je důležité zachytit *přesně*²⁸ průchod stavové proměnné x nějakou hodnotou.

Tato funkcionalita se v Simulinku jmenuje „Zero Crossing Detection“ (ZCD) a v MATLABu „Events“.

Nejjednodušší příkladem je simulace pádu míčku na zem. Rovnice popisující dynamiku je jednoduchá:

$$m\ddot{x} = -mg \quad (3.52)$$

[Ex.: NM12] Rovnici budeme řešit numericky v Simulinku, chceme přitom zjistit přesně okamžik pádu míčku na zem. Na obrázku 3.11 je použitý model, ve kterém je blokem Compare To Zero zajištěno ukončení simulace při podmínce $x \leq 0$. Právě v tomto bloku je možnost zapnout ZCD.



Obrázek 3.11 Simulinkovský model odpovídající rovnici 3.52 s podmínkou ukončení simulace

Na obr. 3.12 je vidět rozdíl mezi výsledkem integrace při vypnutém a zapnutém ZCD. V případě, že se nestaráme o přesný okamžik dopadu (ZCD vypnuto) solver automaticky zvolí velký krok výpočtu (dynamika řešeného problému je hladká) a míček „propadne“ pod hodnotu $x = 0$. Pokud zapneme ZCD, algoritmus zvolí velikost kroku tak, že simulace přesně skončí v hodnotě $x = 0$.

3.5.1.3 Algebraická smyčka

Na obrázku 3.13 vlevo vidíme jednoduchý příklad algebraické smyčky: výstup z bloku Add je přímo přiveden na jeho vstup.

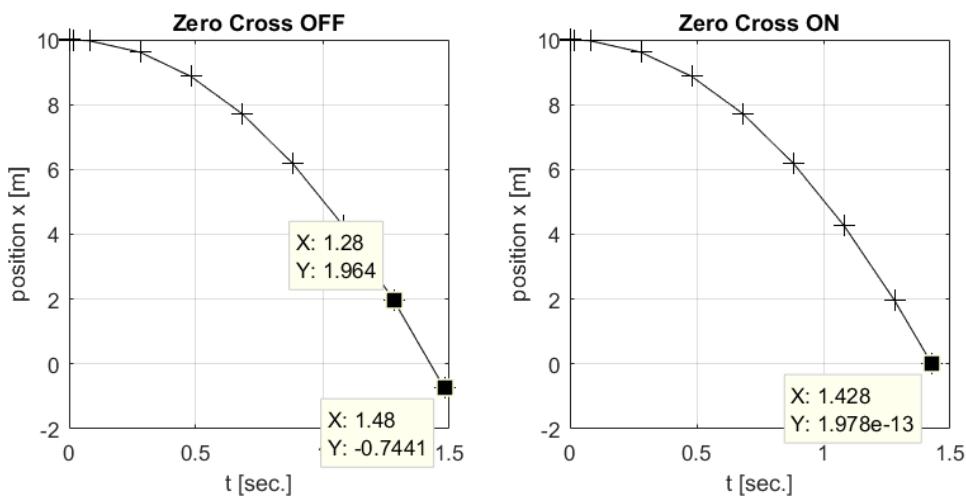
Otázka: Kolik je tedy výstupní hodnota zobrazená v Display?

Odpověď: Zkuste nad tím chvíliku přemýšlet, ...

Jedná se vlastně o řešení rovnice:

$$y = 1 - y, \quad (3.53)$$

²⁸ Pohybujeme se ve světě numerických výpočtů, kde přesně znamená s určitou danou tolerancí.



Obrázek 3.12 Simulace rovnice 3.52 s vypnutým a zapnutým Zero Crossing Detection



Obrázek 3.13 Jednoduchý model s (vlevo) a bez (vpravo) algebraické smyčky

[Ex.: SL13] Jejíž řešení je jasné. Přesto je takto formulovaná úloha pro myšlení člověka „obtížná“ a stejně tak je obtížná pro solver Simulinku. Pokud je Simulinkem detekovaná algebr. smyčka, musí ji v každém kroku řešit. Numerickou iteracní metodou ***TODO*** R EFERENCE tedy hledáme řešení rovnice 3.53, což pochopitelně zpomalí simulaci.

Model na obrázku 3.13 vpravo také obsahuje smyčku, která ale *není* algebraická. Reprezentuje totiž rovnici $\dot{x} = -2x$, tedy ODE, se kterou si Simulink snadno poradí.

Pokud je to možné, snažíme se výskytu algebr. smyčky v modelu vyhnout. Mimo to, že dojde ke zpomalení simulace totiž *není* možné model s algebr. smyčkou použít pro generování C kódu (např. pro RCP nebo HIL ***TODO*** R EFERENCE).

3.5.2 Stiff systém

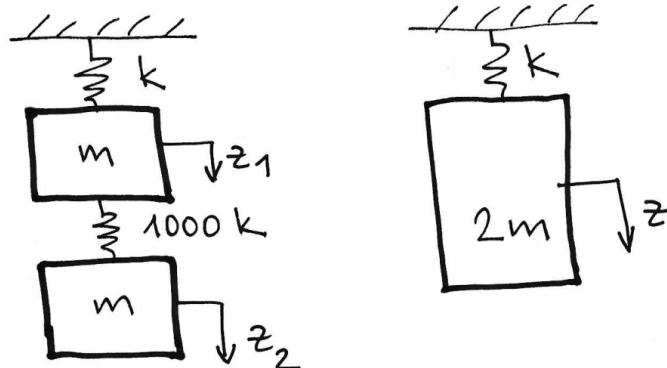
Výklad pojmu *numericky tuhý (stiff) systém* bychom mohli začít pohledem do Wikipedie²⁹, raději si ale ukážeme dva konkrétní příklady, které následně zobecníme.

3.5.2.1 Mechanický oscilátor se dvěma hmotami a nestejně tuhými pružinami

Na obr. 3.14 vlevo je mechanický systém se dvěma stupni volnosti. Hmotnosti obou těles jsou shodné, spodní pružina má *výrazně větší tuhost* než horní. Na obrázku vpravo

²⁹ Citace z Wikipedie: In mathematics, a stiff equation is a differential equation for which certain numerical methods for solving the equation are numerically unstable, unless the step size is taken to be extremely small. It has proven difficult to formulate a precise definition of stiffness, but the main idea is that the equation includes some terms that can lead to rapid variation in the solution.

je pak zjednodušující model, který ukazuje situaci, kdy spodní pružinu považujeme za nekonečně tuhou. Smyslem tohoto příkladu je promyslet chování levého modelu a porovnat rozdíl mezi levým a pravým modelem.



Obrázek 3.14 Schéma mechanického oscilátoru se dvěma pružinami (vlevo) a jeho zjednodušení (vpravo)

Pohybové rovnice snadno odvodíme ve tvaru:

$$\begin{aligned}\ddot{z}_1 &= \frac{-k_1 - k_2}{m} z_1 + \frac{k_2}{m} z_2 \\ \ddot{z}_2 &= \frac{k_2}{m} z_1 + \frac{-k_2}{m} z_2\end{aligned}$$

Tyto dvě ODE druhého řádu převedeme na 4 ODE 1. řádu ($x_1 = z_1$, $x_2 = z_2$, $x_3 = \dot{z}_1$, $x_4 = \dot{z}_2$):

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{-k_1 - k_2}{m} & \frac{k_2}{m} & 0 & 0 \\ \frac{k_2}{m} & \frac{-k_2}{m} & 0 & 0 \end{bmatrix} \mathbf{x} = \mathbf{Ax} \quad (3.54)$$

Z teorie dynamických systémů víme, že dynamika systému $\dot{\mathbf{x}} = \mathbf{Ax}$ je určena vlastními čísly matice \mathbf{A} . Vlastní čísla vypočteme takto:

```
k1 = 100
k2 = 20*k1; % pruzina k2 je radev tuzsi nez k1
m1 = 3/2; m2 = 3/2; % hmotnosti jsou rozdelene na poloviny
% z = [z1; z2] - polohy dvou hmot m1 a m2
K = [(k1+k2)/m1, -k2/m1; -k2/m2, k2/m2];
V = zeros(2,2);
% Prevod 2x ODE 2. radu na 4x ODE 1.radu
% x = [z1 ; z2 ; dz1 ; dz2]
A = [zeros(2,2), eye(2,2); -K, -V]
eigenValues_2dof = eig(A)
```

Pro zadané hodnoty dostaneme výsledek:

```
eigenValues_2dof =
-0.0000 +51.9655i
-0.0000 -51.9655i
-0.0000 + 5.7373i
-0.0000 - 5.7373i
```

Pro srovnání ještě provedeme výpočet pro zjednodušený model (obr. 3.14 vpravo):

```
%% MODEL 1 DOF
k = 100, m = 3;
A = [0 , 1; -k/m, 0];
% x = [z , dz], kde z je poloha
eigenValues_1dof = eig(A)
```

Výsledek je tento:

```
eigenValues_1dof =
0.0000 + 5.7735i
0.0000 - 5.7735i
```

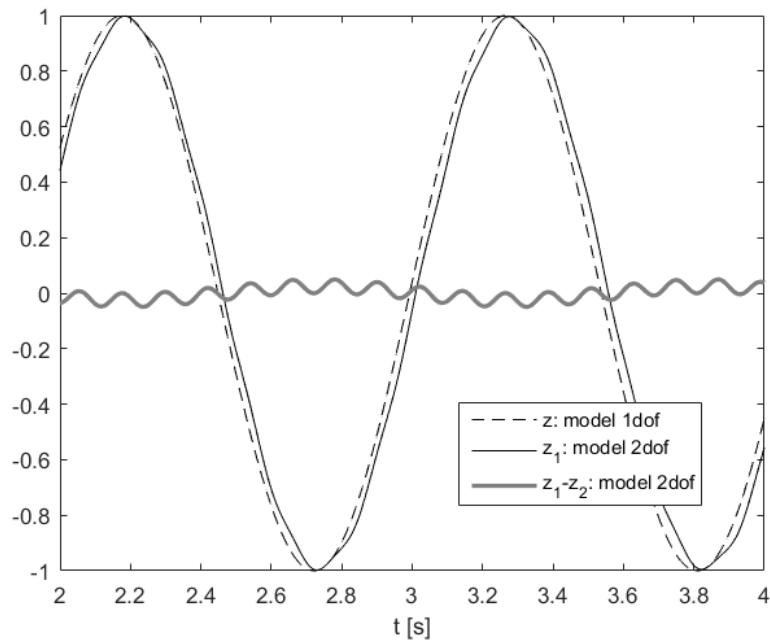
Vidíme, že:

- vlastní čísla jsou komplexně sdružená a leží na imaginární ose (reálná složka je nulová) – znamená to tedy, že systém bude kmitat netlumeně,
- v případě modelu 2dof jsou v systému přítomny „dvě dynamiky“: rychlejší vlastní číslo je cca 10x větší než pomalejší a
- zjednodušený model 1dof má vlastní číslo přibližně stejně velké jako pomalejší modelu 2dof.

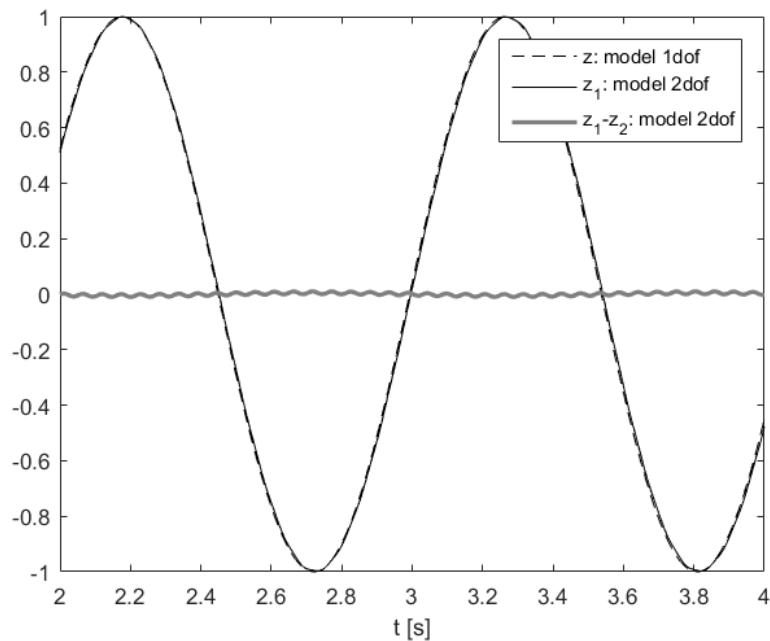
Na obr. 3.15 vidíme porovnání lineární simulace (`lsim`, kap. ??). Je vidět, že 2dof model je mírně „pomalejší“ a trajektorie z_1 je oproti z (model 1 dof) mírně ovlivněná, ale celkově lze říci, že trajektorie z a z_1 jsou srovnatelné.

Pokud budeme ale model 2dof simulovat pomocí řešiče (solveru), tedy pomocí metod numerické integrace, zjistíme, že se jedná o numericky tuhý (stiff) systém. Více v kap. 3.6.2.

Pokud zvolíme hodnotu $k_2 = 100k_1$, dostaneme výsledek na obr. 3.16. Amplituda odchylky modelů 1dof a 2dof $z_1 - z_2$ je 0,005 – můžeme tedy říci, že je to 0,5% z rozsahu výstupního signálu.



Obrázek 3.15 Simulace (lsim) mechanického oscilátoru 2dof (obr. 3.14) pro $k_2 = 20k_1$



Obrázek 3.16 Výsledek simulace pro $k_2 = 100k_1$, výběr čas. intervalu 2-4 sec. (srovnej s obr. 3.15)

3.5.2.2 DC motor jako ukázkový příklad stiff systému se všemi důsledky z toho plynoucími

Vyjdeme z rovnic odvozených v kapitole ?? ve tvaru lineárního stavového modelu $\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{Bu}$:

$$\begin{bmatrix} \frac{di}{dt} \\ \frac{d\omega}{dt} \end{bmatrix} = \begin{bmatrix} -\frac{R}{L} & -\frac{k_M}{L} \\ \frac{k_M}{J} & -\frac{b}{J} \end{bmatrix} \begin{bmatrix} i \\ \omega \end{bmatrix} + \begin{bmatrix} \frac{1}{L} & 0 \\ 0 & \frac{1}{J} \end{bmatrix} \begin{bmatrix} u \\ \tau \end{bmatrix} \quad (3.55)$$

Z teorie dynamických systémů *****TODO***** EF víme, že dynamika systému je určená vlastními čísly matice **A**.

Abychom dostali konkrétní hodnoty, dosadíme do matice parametry vybraného konkrétního motoru Maxon RE36 (malý motorek o výkonu 70W): $k_M = 0,0354$; $J = 67,7 \cdot 10^{-7}$; $R=1,11$; $L=0,2 \cdot 10^{-3}$; $b=0$ (zanedbáme viskózní tření).

Vlastní čísla vypočteme takto:

```
k_M = 0.0354;
J = 67.7e-7;
R = 1.11 ;
L = 0.2e-3;
A = [-R/L, -k_M/L; k_M/J 0];
lambda = eig(A)
```

Po dosazení hodnot vychází $\lambda_1 = -5378$ a $\lambda_2 = -172$. Vidíme, že vlastní čísla jsou záporná (stabilní systém) a nejsou komplexně sdružená (systém nebude kmitat).

Můžeme zhruba říci, že mechanická časová konstanta je $\tau_m = 1/172 = 0,0058$ sec. a elektrická čas. konst. $\tau_e = 1/5378 = 0,0002$ s. Jinak řečeno, elektrický subsystém dosáhne 63% ustáleného stavu 31x rychleji³⁰.

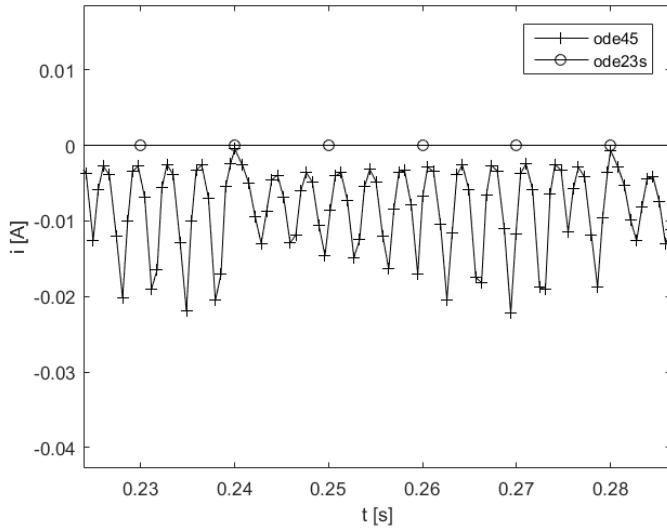
Na obr. 3.17 vidíme detail výsledku numerické integrace plného modelu DC motoru pomocí solverů ode45 a ode23s. Vidíme, že defaultní solver ode45 funguje velice špatně (osciluje pod korektní hodnotou ustáleného proudu) zatímco solver ode23s (stiff) funguje bezvadně.

3.5.2.3 Závěr úvah k stiff systémům

Na základě studia obou příkladů (mechanický systém 2dof 2.5 i DC motor 2.5) můžeme poznamenat toto:

- Numericky tuhý (stiff) systém se vyznačuje tím, že vlastní čísla jeho matice **A** jsou „velmi rozdílná“.
- Z praktického hlediska to znamená, že základní řešič ode45 bude mít problémy s volbou kroku a bude jednak oscilovat a jednak bude potřebovat relativně více kroků integrace.
- Pro řešení je žádoucí použít řešič, který je přímo určen pro stiff systémy, např. ode15s nebo ode23s.
- Také (a někdy především) je ale dobré se zamyslet, zda pracujeme s *rozumným* modelem. Příklad DC motoru je zcela vypovídající – pro naprostou většinu běžných

³⁰ Popsaná úvaha samozřejmě není matematicky korektní – obě rovnice elektrická a mechanická nejsou nezávislé rovnice prvního řádu, jsou provázané. Z hlediska chápání chování tohoto elektromagnetického systému to ale takto můžeme zjednodušit.



Obrázek 3.17 Detail simulace dynamického modelu DC motoru se zahrnutím indukčnosti (solver ode45 vs. ode23s)

použití modelu DC motoru jako komponenty ve složitějších modelech můžeme indukčnost L zanedbat. Neztratíme tím téměř nic na přesnosti simulované rychlosti a proudu a získáme rychlejší výpočet.

3.5.3 Tvary soustavy ODE

Soustava ODE popisuje nějaký reálný fyzikální nebo technický problém. Rovnice se odvozují různým způsobem (např. pomocí Lagrangeových rovnic druhého druhu – kap. ??) a mohou být vytvořeny v různém tvaru/struktuře. Z hlediska volby řešiče, nastavení a vlastností numerického řešení je vhodné rozlišovat následující tři tvary:

- **explicitní tvar ODE (přímo vyjádřený)**

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, t) \quad (3.56)$$

Tento tvar je pro integraci ODE nejvhodnější, ne vždy ale lze soustavu do tohoto tvaru přepsat.

- **lineárně implicitní tvar ODE (nepřímo vyjádřený)**

$$\mathbf{M}(\mathbf{y}, t)\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, t) \quad (3.57)$$

Tento tvar je typický např. pro mechanické systémy (viz např. rovnice XXXXXX XXXX).

- **plně implicitní tvar ODE**

$$\mathbf{f}(\dot{\mathbf{y}}, \mathbf{y}, t) = \mathbf{0} \quad (3.58)$$

Tento tvar soustavy ODE je nejobecnější a není z našeho hlediska příliš důležitý. V technických problémech se nevyskytuje a lze jej řešit pouze řešičem ode15i.

3.5.4 Přehled řešičů (solverů) v MATLABu a Simulinku

V předchozí sekci 3.4 jsme uvedli základy metod numerické integrace a v této sekci 3.5 jsme se věnovali otázkám variabilní délky kroku, stiff systémům a tvarům soustavy

ODE. Všechny tyto záležitosti jsou zohledněny při praktické implementaci v jednotlivých řešičích v MATLABu a Simulinku.

Základní sada řešičů s variabilním krokem (ode45, ode15s, ...) je v MATLABu a Simulinku shodná³¹, celkově jsou ale v možnostech obou prostředí významné praktické rozdíly. Uvedeme bodově ty nejdůležitější:

- Řešiče s konstantním krokem (Fixed-step) jsou dostupné pouze v Simulinku – mají totiž opodstatnění pouze při aplikaci v real-time systémech, které vždy fungují s konstantní dopředu zadáným vzorkováním.
- MATLAB umí řešit všechny tvary soustavy ODE (viz 3.5.3) a umí řešit soustavy DAE indexu 1.
- Obě prostředí umí detekovat průchod nulou (viz 3.5.1.2), v MATLABU se tato funkcionality jmenuje Events, v Simulinku Zero Crossing Detection.

V následující tabulce jsou shrnutý různé aspekty řešení ODE v obou prostředích.

	MATLAB	Simulink
řešič pro diskrétní systémy	nemá význam ³²	ano
řešič s konst. krokem (Fixed-step solver)	ne	ode1, ode2, ode3, ode4, ode5, ode8, ode14x
řešič s proměnným krokem (Variable-step solver)	ode45, ode15s, ode23, ode23t, ode113	ode45, ode15s, ode23, ode23t, ode23s, ode23tb, ode113
Tvar ODE	explicitní lineárně implicitní plně implicitní: ode15i	pouze explicitní
DAE solver (index=1)	ode15i, ode15s, 23t	ne
Detekce průchodu nulou	Events	Zero Crossing Detection
Řešení algebraické smyčky	nemá význam	ano pro řešiče s variabilním krokem

Velmi stručně shrneme vlastnosti (některých) řešičů:

- ode45 – řešič první volby, defaultní řešič Simulinku, založen na explicitní metodě Runge-Kutta (řád 4,5), pro non-stiff problémy. Viz také [`<matlab_help>`]³³.
- ode23 – RK (řád 2,3), může být na některé problémy efektivnější (rychlejší výpočet při stejně přesnosti) než ode45. Non-stiff.³⁴

³¹Ačkoli jsou tyto základní řešiče v MATLABu i Simulinku stejné, konkrétní implementace zcela shodná není. Snadno lze ukázat, že např. automatická volba kroku funguje mírně odlišně (viz také obrázek 3.18).

³²Řešení dynamiky diskrétního systému je pouhé vyříšení iterační rovnice $x_{k+1} = f(x_k, u_k)$. V MATLABu toto řešení provedeme např. ve smyčce for. V Simulinku má smysl zavádět řešič, i když to řeši v pravém slova smyslu není (k žádné numerické integraci nedochází).

³³citace [`<matlab_help>`]: ode45 is based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince pair. It is a one-step solver – in computing $y(t_n)$, it needs only the solution at the immediately preceding time point, $y(t_{n-1})$. In general, ode45 is the best function to apply as a first try for most problems.

³⁴citace z [`<matlab_help>`]: ode23 is an implementation of an explicit Runge-Kutta (2,3) pair of Bogacki and Shampine. It may be more efficient than ode45 at crude tolerances and in the presence of moderate stiffness. Like ode45, ode23 is a one-step solver.

- `ode113` – vícekrokový solver, proměnný řád, metoda Adams-Bashforth-Moulton. Non-stiff.³⁵
- `ode15s`³⁶, `ode23s`³⁷, `ode23t`³⁸, `ode23tb`³⁹ – solvery vhodné pro stiff problémy.
- `ode15i` – solver pro plně implicitní tvar ODE (viz kap. 3.5.3).

³⁵ citace z [<matlab_help>]: `ode113` is a variable order Adams-Bashforth-Moulton PECE solver. It may be more efficient than `ode45` at stringent tolerances and when the ODE file function is particularly expensive to evaluate. `ode113` is a multistep solver — it normally needs the solutions at several preceding time points to compute the current solution.

³⁶ citace z [<matlab_help>]: `ode15s` is a variable order solver based on the numerical differentiation formulas (NDFs). Optionally, it uses the backward differentiation formulas (BDFs, also known as Gear's method) that are usually less efficient. Like `ode113`, `ode15s` is a multistep solver. Try `ode15s` when `ode45` fails, or is very inefficient, and you suspect that the problem is stiff, or when solving a differential-algebraic problem.

³⁷ citace z [<matlab_help>]: `ode23s` is based on a modified Rosenbrock formula of order 2. Because it is a one-step solver, it may be more efficient than `ode15s` at crude tolerances. It can solve some kinds of stiff problems for which `ode15s` is not effective.

³⁸ citace z [<matlab_help>]: `ode23t` is an implementation of the trapezoidal rule using a "free" interpolant. Use this solver if the problem is only moderately stiff and you need a solution without numerical damping. `ode23t` can solve DAEs.

³⁹ citace z [<matlab_help>]: `ode23tb` is an implementation of TR-BDF2, an implicit Runge-Kutta formula with a first stage that is a trapezoidal rule step and a second stage that is a backward differentiation formula of order two. By construction, the same iteration matrix is used in evaluating both stages. Like `ode23s`, this solver may be more efficient than `ode15s` at crude tolerances.

3.6 Příklady řešení ODE v MATLABu a Simulinku

3.6.1 Mechanický oscilátor 1dof

Začneme příkladem explicitní ODE, kdy je situace nejjednodušší. Uvažujme jednoduchý příklad rovnice 3.20 volného kmitání mech. oscilátoru:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k/m & -b/m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad (3.59)$$

[Ex.: NM14]

Nejprve na definujeme tuto ODE v samostatném m-souboru (pořadí argumentů $t, y, options$ se musí dodržet, pak následují volitelné parametry):

```
function dy = my_odefun(t,y,options,k,b,m)
A = [0, 1; -k/m, -b/m];
dy = A*y; % vypocet vektoru derivaci
```

Následně tuto funkci použijeme jako argument při volání zvoleného solveru:

```
clc,clear;
k = 100, b = 2, m=1;
y0 = [0.1;0]; % pocatecni stav
options = '';
tspan = [0 10]; % cas simulace
[T,Y] = ode45('my_odefun',tspan,y0,options,k,b,m);
plot(T,Y(:,1)) % vykresli polohu
```

Solver ode45 si pak volá funkci `my_odefun` *vždy když potřebuje* vypočítat hodnotu \dot{y} .

Solver potřebuje pro výpočet jednoho kroku provést několik volání funkce v různých bodech $f(y, t)$, což je to zřejmě např. z popisu Runge-Kuttovy metody (kap. 3.4.3). Pro zajímavost si vyzkoušíme jak je na tom z tohoto hlediska solver `ode45` v MATLABu i v Simulinku. V případě solveru v MATLABu musíme ještě doplnit nastavení hodnoty `Refine`,⁴⁰ které řídí počet kroků, které se exportují z výpočtu jako jeho výstup:

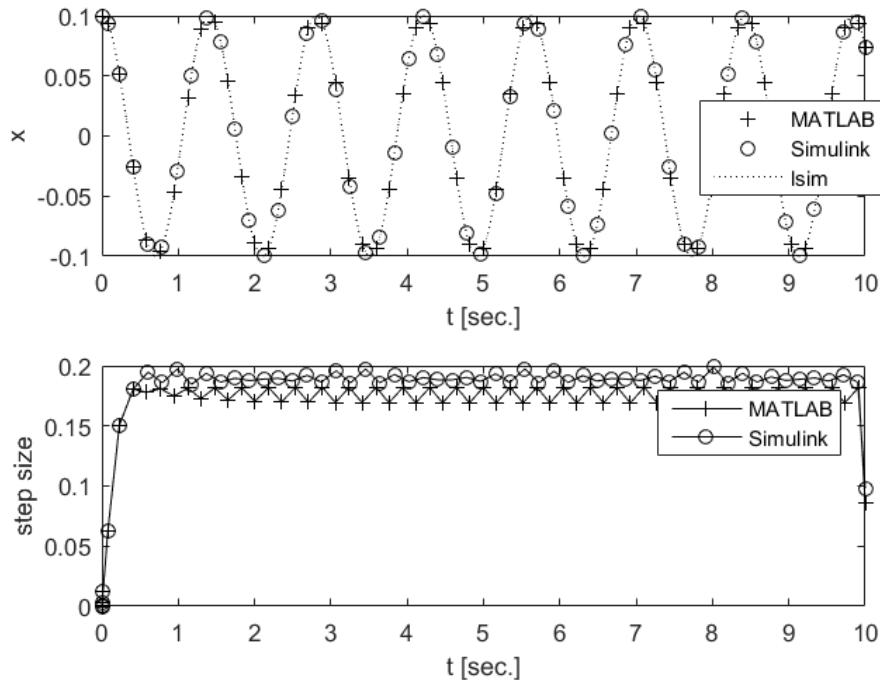
```
options = odeset('Stats', 'on', 'Refine',1);
```

Pak už si můžeme směle prohlédnout výsledky:

MATLAB: počet volání funkce	373
MATLAB: počet kroků simulace	62
Simulink: počet volání funkce	408
Simulink: počet kroků simulace	59

⁴⁰ Citace z [<matlab_help>]: In all the solvers, the default value of `Refine` is 1. Within `ode45`, however, the default is 4 to compensate for the solver's large step sizes. To override this and see only the time steps chosen by `ode45`, set `Refine` to 1.

Vidíme, že v MATLAB potřebuje pro jeden simulační krok 6 volání funkce. Dále si můžeme všimnout, že výstupem ze Simulinku je mírně jiný počet kroků a i počet volání funkce je jiný.

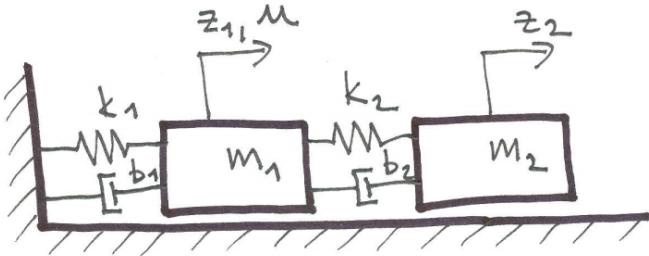


Obrázek 3.18 Porovnání přesného řešení (`lsim`), řešení v MATLABu pomocí funkce `ode45` a řešení v Simulinku solverem `ode45` (nahoře); Vývoj velikosti automaticky zvoleného kroku solveru `ode45` v MATLABu a Simulinku (dole)

Na obr. 3.18 si pak můžeme všimnout, že automatická volba délky kroku je nastavena u MATLABu i Simulinku mírně odlišně. První tři kroky je u obou shodná, pak ale Simulink volí o něco větší délku kroku.

3.6.2 Příklad: Mechanický oscilátor 2dof

3.6.2.1 Sestavení rovnic



Obrázek 3.19 Schéma mechanického systému s 2dof a jedním vstupem (síla působící na hmotu 1)

$$\begin{aligned} m_1 \ddot{z}_1 &= -k_1 z_1 + k_2(z_2 - z_1) - b_1 \dot{z}_1 + b_2(\dot{z}_2 - \dot{z}_1) + u \\ m_2 \ddot{z}_2 &= -k_2(z_2 - z_1) - b_2(\dot{z}_2 - \dot{z}_1) \end{aligned}$$

$$\begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix} \begin{bmatrix} \ddot{z}_1 \\ \ddot{z}_2 \end{bmatrix} = - \begin{bmatrix} k_1 + k_2 & -k_2 \\ -k_2 & k_2 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} - \begin{bmatrix} b_1 + b_2 & -b_2 \\ -b_2 & b_2 \end{bmatrix} \begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \end{bmatrix} + \begin{bmatrix} u \\ 0 \end{bmatrix} \quad (3.60)$$

$$\mathbf{M}\ddot{\mathbf{z}} = -\mathbf{K}\mathbf{z} - \mathbf{V}\dot{\mathbf{z}} + \mathbf{f} \quad (3.61)$$

Převedeme na 4 ODE 1. řádu ($\dot{\mathbf{x}} = [z_1, z_2, \dot{z}_1, \dot{z}_2]^T$):

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{0} & \mathbf{E} \\ -\mathbf{M}^{-1}\mathbf{K} & -\mathbf{M}^{-1}\mathbf{V} \end{bmatrix} \mathbf{x} + \mathbf{M}^{-1} \begin{bmatrix} \mathbf{0} \\ \mathbf{f} \end{bmatrix} \quad (3.62)$$

V tomto případě může snadno rovnice přepsat i ručně:

$$\begin{aligned} \ddot{z}_1 &= \frac{-k_1 - k_2}{m_1} z_1 + \frac{k_2}{m_1} z_2 + \frac{-b_1 - b_2}{m_1} \dot{z}_1 + \frac{b_2}{m_1} \dot{z}_2 + \frac{1}{m_1} u \\ \ddot{z}_2 &= \frac{k_2}{m_2} z_1 + \frac{-k_2}{m_2} z_2 + \frac{b_2}{m_2} \dot{z}_1 + \frac{-b_2}{m_2} \dot{z}_2 \end{aligned}$$

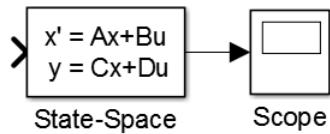
$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{-k_1 - k_2}{m_1} & \frac{k_2}{m_1} & \frac{-b_1 - b_2}{m_1} & \frac{b_2}{m_1} \\ \frac{k_2}{m_2} & \frac{-k_2}{m_2} & \frac{b_2}{m_2} & \frac{-b_2}{m_2} \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{m_1} \\ 0 \end{bmatrix} u = \mathbf{A}\mathbf{x} + \mathbf{B}u \quad (3.63)$$

3.6.2.2 Řešení ODE v MATLABu

Vzhledem k tomu, že jsme rovnici 3.60 převedli z lineárně implicitního do explicitního tvaru, je řešení shodné s předcházejícím příkladem 3.6.1.

3.6.2.3 Řešení ODE v Simulinku

Pokud máme vytvořené matice \mathbf{A} , \mathbf{B} , \mathbf{C} a \mathbf{D} , můžeme model snadno sestavit s využitím bloku State-Space.



Obrázek 3.20 Schéma modelu v Simulinku s využitím bloku State-Space

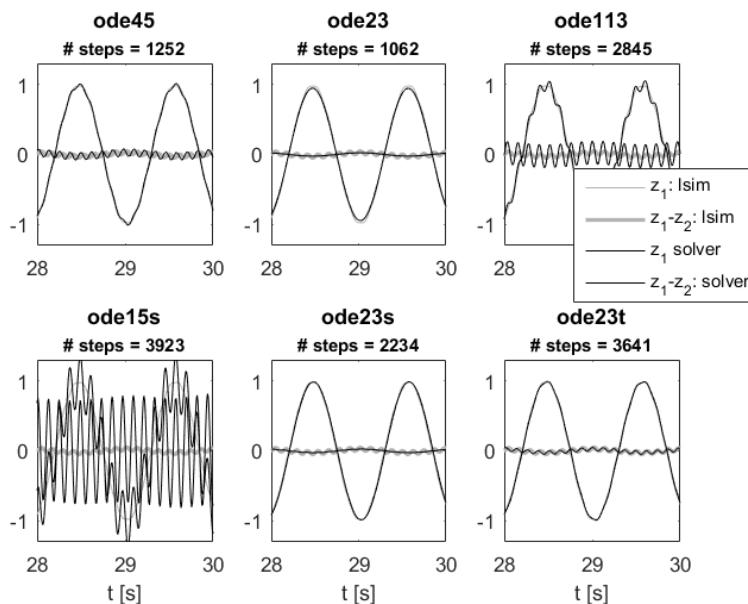
Pomocí příkazů `simset` a `sim` můžeme automaticky spustit simulaci s různými parametry (např. solverem) a vykreslit v MATLABu výsledky:

```
tspan = [0 30];
options = simset('Solver','ode15s');
sim('NM13_model_mech2dof_ss.slx',tspan, options);
t = ScopeData.time;
z1 = ScopeData.signals.values(:,1);
z2 = ScopeData.signals.values(:,2);
plot(t,z1-z2)
```

3.6.2.4 Simulace pro výrazně rozdílné tuhosti pružin (stiff systém)

Abychom ukázali jak můžou být výsledky simulace citlivé na volbu řešiče, budeme simulovat rovnice 3.63 s výrazně rozdílnými tuhostmi k_1 a k_2 a s nulovým tlumením $b_1 = b_2 = 0$:

```
k1 = 100, m1 = 1.5, b1=0 ;
k2 = 20*k1, m2 = 1.5, b2=0 ;
```

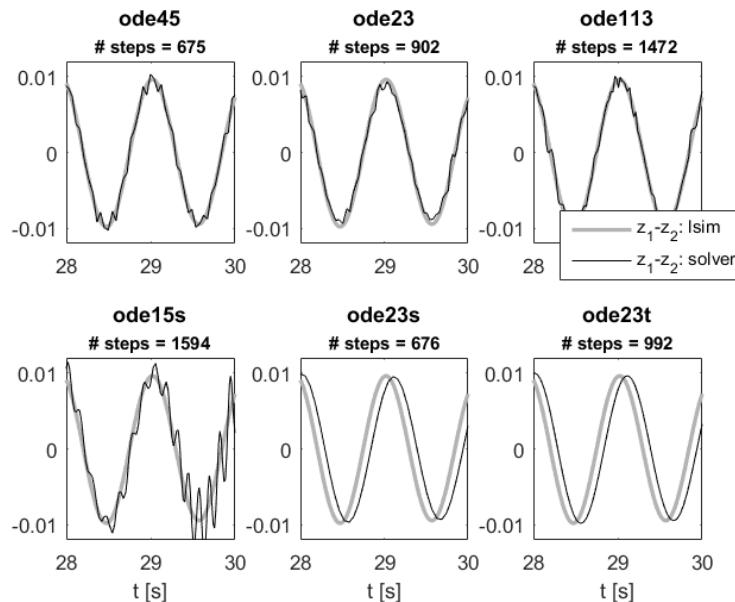


Obrázek 3.21 Srovnání numerické integrace pomocí různých solverů ODE s přesným řešením (`lsim`) (nulové tlumení $b_1 = b_2 = 0$)

[Ex.: NM13]

Na obr. 3.21 vidíme velice rozdílné chování jednotlivých testovaných solverů. Nejhůře je na tom (možná překvapivě) `ode15s`, který by měl být vhodný pro stiff problémy. Nejlepší výsledek dává patrně `ode23t`, i když vyžaduje druhý nejvyšší počet kroků k řešení.

Podstatně lepší numerické stability dosáhneme pokud bude v soustavě nenulové tlumení. Na obrázku 3.22 je zobrazení rozdílu souřadnic $z_1 - z_2$ (při zobrazení z_1 jako na předchozím obr. 3.21 bychom neviděli rozdíly mezi chováním jednotlivých solverů). Vidíme, že `ode23s` a `ode23t` dávají asi nejlepší výsledky, i když je patrné jisté fázové posunutí.



Obrázek 3.22 Srovnání numerické integrace pomocí různých solverů ODE s přesným řešením (`lsim`) (nulové tlumení $b_1 = 0, 2; b_2 = 4$)

3.6.3 Příklad: Mostový jeřáb (2dof)

V kap. 5.1 jsme odvodili rovnice mostového jeřábu ve tvaru:

$$\begin{bmatrix} \frac{m_1 + m_2}{m_2 L \cos(q_2)} & \frac{m_2 L \cos(q_2)}{m_2 L^2} \\ \frac{m_2 L \cos(q_2)}{m_2 L^2} & \frac{-m_2 L \dot{q}_2^2 \sin(q_2)}{m_2 g L \sin(q_2)} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (3.64)$$

Vidíme, že jde o soustavu dvou nelineárních ODE druhého řádu, obecně zapsanou takto:

$$\mathbf{W}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{0} \quad (3.65)$$

Nejprve převedeme soustavu na 4 rovnice prvního řádu, ale ponecháme je v lineárně implicitním tvaru (tvar, který má rám MATLAB: 3.57). Stavový vektor má tvar: $\mathbf{x} = [q_1, q_2, \dot{q}_1, \dot{q}_2]^T$.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & m_1 + m_2 & m_2 L \cos(x_2) \\ 0 & 0 & m_2 L \cos(x_2) & m_2 L^2 \end{bmatrix} \dot{\mathbf{x}} = \begin{bmatrix} x_3 \\ x_4 \\ m_2 L x_4^2 \sin(x_2) \\ -m_2 g L \sin(x_2) \end{bmatrix} \quad (3.66)$$

Stručně tedy:

$$\mathbf{M}(x_2)\dot{\mathbf{x}} = \mathbf{f}(x_2, x_3, x_4) \quad (3.67)$$

Matice \mathbf{M} je závislá na stavu a před provedením inverze musíme ověřit, že není (ne-může nikdy být) singulární. To snadno ukážeme výpočtem determinantu pomocí Symbolic Math toolboxu:

```
clc, clear
syms m1 m2 L x2 x3 x4 g
M = [1 0 0 0; ...
      0 1 0 0; ...
      0 0 m1+m2 m2*L*cos(x2); ...
      0 0 m2*L*cos(x2) m2*L^2]
detM = simplify( det(M) )
```

Determinant má hodnotu

$$L^2 m_2 (m_1 + m_2 - m_2 \cos^2(x_2)) \neq 0. \quad (3.68)$$

S tímto důkazem tedy můžeme snadno převést rovnici 3.67 na explicitní tvar a vyřešit stejně jako v předchozích příkladech:

$$\dot{\mathbf{x}} = \mathbf{M}^{-1}(x_2)\mathbf{f}(x_2, x_3, x_4) \quad (3.69)$$

Řešiče MATLABu ovšem nabízí efektivnější způsob jak s lineárně implicitním tvarem pracovat a to především tehdy, když matice \mathbf{M} je nebo může být singulární (soustava ODE se mění na soustavu DAE). Problém necháme zapsán v lin. impl. tvaru 3.67 a

pomocí vlastnosti Mass solveru sdělíme⁴¹, jak matice vypadá (jak se jmenuje funkce pro její výpočet).

Ukážeme si kompletní kód napsaný v jednom souboru pomocí „nested functions“⁴².

```

function CraneSimulation
clc,clear;
m1 = 10; m2 = 2; L = 0.5; g = 9.81;
x0 = [0; pi/2 ;0;0]; % pocatecni podminky [x, phi, dx, dphi]
options = odeset('Stats', 'on', 'Refine',1, 'Mass',@ode_M);
tspan = [0 1000]; % cas simulace
[T,Y] = ode45(@ode_f,tspan,x0,options);
figure(1), subplot(2,1,1)
    plot(T,Y(:,1),'k-+') % vykresli polohu x
    xlabel('t [sec.]'), ylabel('x')
subplot(2,1,2)
    plot(T,Y(:,2),'k-+') % vykresli natoceni phi
    xlabel('t [sec.]'), ylabel('phi')
    grid on
shg

function f = ode_f(t,x)
    f = [x(3);
          x(4);
          m2*L*x(4)^2*sin(x(2));
          -m2*g*L*sin(x(2))];
end
function M = ode_M(t,x)
    M = [1 0 0 0; ...
          0 1 0 0; ...
          0 0 m1+m2 m2*L*cos(x(2));...
          0 0 m2*L*cos(x(2)) m2*L^2];
end
end

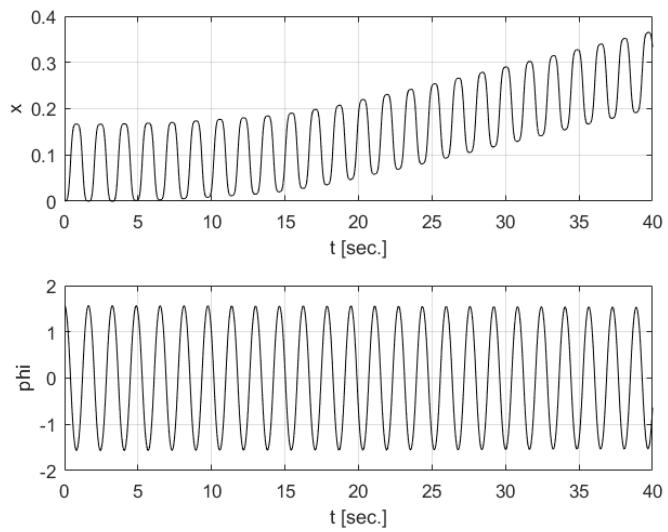
```

V tomto konkrétním ilustrativním příkladu nezjistíme pro solver ode45 téměř žádný rozdíl ani v přesnosti ani v rychlosti výpočtu (zrychlení o 10%). Podstatně zajímavější může být zrychlení u stiff solverů a/nebo u komplexnějších problémů. Hlavní aplikace je ale – jak už bylo řečeno – u řešení DAE soustav.

Na obr. 3.23 pak opět vidíme, jak snadno lze ukázat, že numerická integrace není vždy přesná. Stačí nechat kyvadlo jeřábu kývat 40 sekund a vozíček nám pomalu začne odjíždět kladným směrem osy x – což by tedy neměl...

⁴¹ Podle charakteru matice \mathbf{M} , která může být konstantní, nesingulární nebo singulární, solver řeší problém inverze. Přímá inverze je nahrazena LU rozkladem. Více také vlastnosti solveru MStateDependence a MassSingular.

⁴² Vnořené (nested) funkce vidí parametry hlavní funkce, není je tedy nutné předávat v hlavičce funkcí.



Obrázek 3.23 Ukázka integrace rovnice 3.66 solverem ode45
 (očekávali bychom rovnoměrné oscilace hodnoty x mezi 0 a cca
 0,17)

3.7 Otázky a neřešené úlohy

- Co to je Refine factor u ODE solveru v MATLABu a Simulinku?
- Otevři si v MATLAB Editoru funkci `ode45` (příkaz `open ode45`) a prozkoumej jak vypadá! Kde je jádro Runge-Kuttovy metody? Kolikrát a kde se spouští Tvoje funkce `my_odefun` (viz ??)?

4 Model DC motoru

Elektrický motor s komutátorem a budicími magnety na statoru napájený stejnosměrným napětím je nejstarším, nejjednodušším a stále používaným strojem⁴³.

V této kapitole se podíváme podrobněji na model DC motoru, který je užitečný prakticky (např. pro dimenzování pohonu nebo návrh řízení) i pedagogicky (lze na něm demostrovat řadu důležitých problémů a otázek).



Obrázek 4.1 Malý DC motor

4.1 Dynamický model DC motoru

Základní rovnice bývá uváděna ve tvaru:

$$u = Ri + L \frac{di}{dt} + k_M \omega \quad (4.1)$$

$$J \dot{\omega} = k_M i - b \omega + \tau, \quad (4.2)$$

kde u je napájecí napětí, R odpor vinutí, i proud, L indukčnost, k_M konstanta, ω úhlová rychlosť, J moment setrvačnosti rotoru, b viskózní tlumení a τ vnější zátěžný moment.

První rovnice 4.1 je elektrická (a obsahuje mechanickou veličinu ω), druhá rovnice 4.2 je mechanická (a obsahuje i). Rovnice jsou tedy vzájemně závislé - tj. nelze při uvažování o dynamice systému oddělit elektrickou a mechanickou část.

Výraz $k_M \omega$ představuje indukované napětí na motoru, výraz $k_M i$ je elektrický moment. První rovnice je tedy součtem napětí, druhá momentů.

Z hlediska matematiky jde o dvě závislé ODE prvního řádu. Přepíšeme je do standardního tvaru:

$$\frac{di}{dt} = \frac{1}{L}(u - Ri - k_M \omega) \quad (4.3)$$

$$\frac{d\omega}{dt} = \frac{1}{J}(k_M i - b \omega + \tau). \quad (4.4)$$

Tyto dvě rovnice lze snadno přepsat do tvaru stavového modelu ***TODO*** R EF:

$$\begin{bmatrix} \frac{di}{dt} \\ \frac{d\omega}{dt} \end{bmatrix} = \begin{bmatrix} -\frac{R}{L} & -\frac{k_M}{L} \\ \frac{k_M}{J} & -\frac{b}{J} \end{bmatrix} \begin{bmatrix} i \\ \omega \end{bmatrix} + \begin{bmatrix} \frac{1}{L} & 0 \\ 0 & \frac{1}{J} \end{bmatrix} \begin{bmatrix} u \\ \tau \end{bmatrix} \quad (4.5)$$

⁴³ Mohli bychom dlouze polemizovat o významu slova „nejjjednodušší“. Pokud jde o samotnou konstrukci a např. životnost, pak je jednodušší bezkartáčový motor (BLDC). U něj je ale podstatně komplikovanější řídicí elektronika, zatímco kartáčový motorek lze napájet třeba přímo z baterie. DC motory (či spíše motorky) se stále používají v různých automobilových aktuátozech (vysouvání oken, nastavování zrcátek, ...), v menších zařízeních, u kterých je potřeba řídit polohu nebo rychlosť pohybu.

Dostali jsme standardní tvar $\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{Bu}$.

Tento tvar můžeme použít např. pro lineární simulaci příkazem `lsim` a využijeme ho také při studiu *stiff* systémů v kap. 3.5.2.2.

Dynamický model DC motoru je hezkou ukázkou tzv. **stiff systému** (numerický tuhého systému). Takový systém obsahuje dvě (nebo několik) částí s velmi rozdílnou dynamikou.

4.2 Statický model elektrického subsystému

Na základě rozboru v kap. 3.5.2 se tedy nabízí otázka, zda by nebylo možné a rozumné dynamiku elektrické části zanedbat, když je natolik odlišná od dynamiky mechanické části.

To vede na formulaci modelu, u kterého je první (elektrická) rovnice statická:

$$u = Ri + k_M \omega \quad (4.6)$$

$$J\dot{\omega} = k_M i - b\omega + \tau, \quad (4.7)$$

Z první (statické) rovnice můžeme vyjádřit proud, dosadit do druhé a získat výsledný model DC motoru:

$$J\dot{\omega} = - \left(\frac{k_M^2}{R} + b \right) \omega + \frac{k_M}{R} u + \tau. \quad (4.8)$$

Dostali jsme už pouze mechanickou rovnici (ODE prvního řádu) s časovou konstantou $\tau = \frac{J}{k_M^2/R+b} = 0.006$ sec.

Na obrázku 4.2 vidíme porovnání simulací dynamického a statického modelu pro konkrétní parametry motoru Maxon RE36.

Je zřejmé, že pokud porovnáváme rychlosť, rozdíl mezi oběma modely je neznačitelný (horní obrázek). Pokud se podíváme na proud, vidíme u dynamického modelu krátký (podívejte se na časovou osu!) přechodový děj, zatímco u statického modelu proud skokově změní svoji hodnotu.

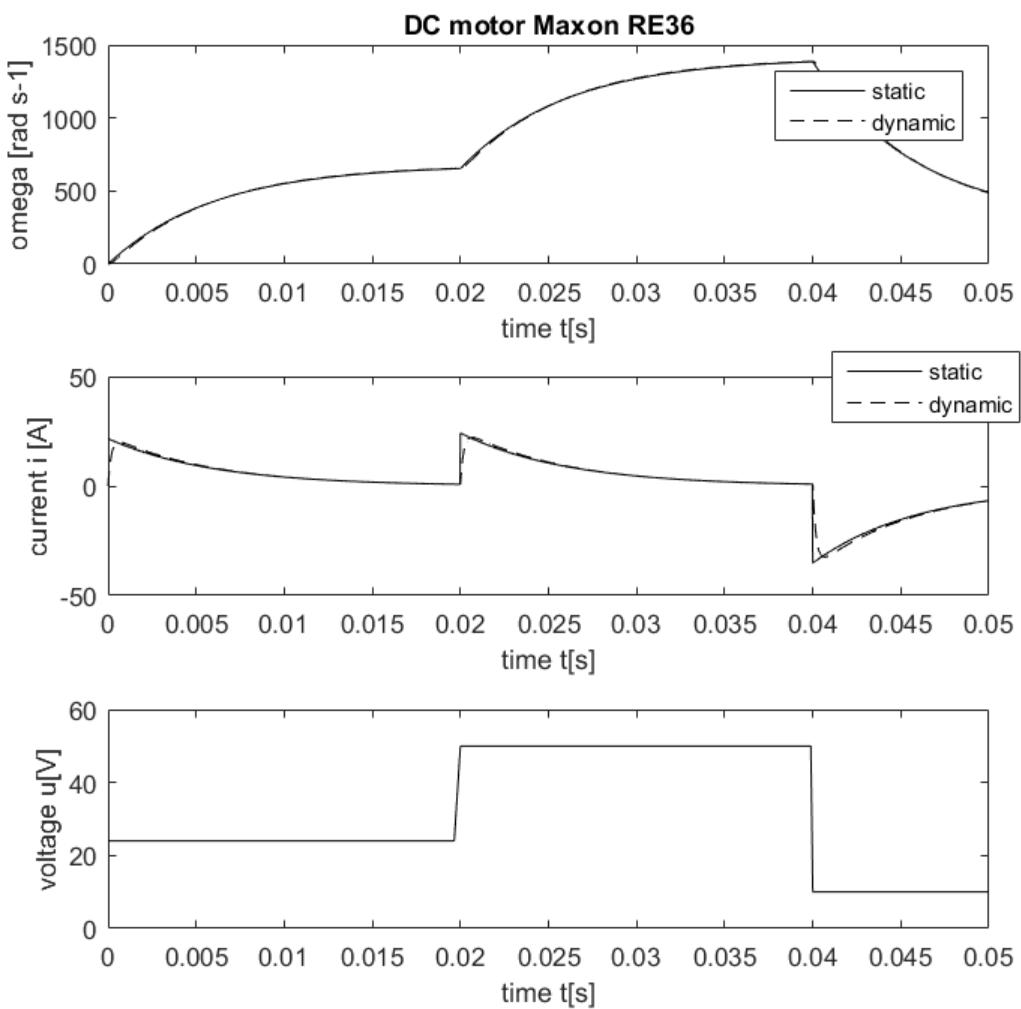
Je tedy zřejmé, že pokud chceme simulovat chování DC motoru, můžeme v naoprosté většině běžných úloh směle zanedbat jeho indukčnost⁴⁴.

4.3 Statická charakteristika (celého) DC motoru

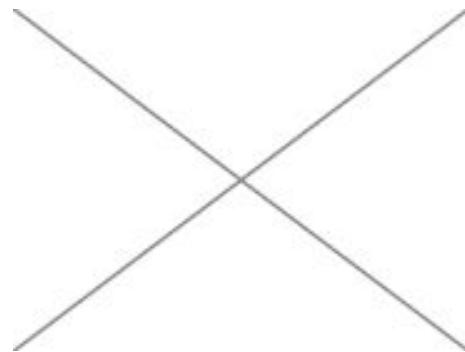
Pro výběr vhodného motoru a podobné jednoduché úvahy je užitečné ještě další zjednodušení - zanedbáme i dynamiku mechanické části a budeme uvažovat plně statický popis (charakteristiku) motoru:

$$0 = - \left(\frac{k_M^2}{R} + b \right) \omega + \frac{k_M}{R} u + \tau. \quad (4.9)$$

⁴⁴ Toto tvrzení platí v případě, že nás nezajímá právě onen málo znatelný přechodový děj na prostředním obrázku. Pokud budeme ladit regulátor proudové smyčky, pak je pochopitelně indukčnost významná.



Obrázek 4.2 Simulace DC motoru Maxon RE36 - porovnání dynamického a statického modelu elektrického subsystému



Obrázek 4.3 Statická charakteristika DC motoru

5 Lagrangeovy rovnice 2. druhu (LR2)

Obsah kapitoly:

5.1	Úvod	74
5.2	Příklad na LR2: Mostový jeřáb	76
5.3	Příklad na LR2: Dvojité kyvadlo	79
5.4	Sestavení LR2 s pomocí nástrojů pro symbolické výpočty	81

5.1 Úvod

Lagrangeovy rovnice 2. druhu (LR2) jsou návodem jak sestavit pohybové rovnice (dynamický model) mechanického systému. Pokud vyjádříme kinetickou a potenciální energii jako funkci zobecněných souřadnic q_i , stačí už mechanicky provést následující derivace:

$$\frac{d}{dt}\left(\frac{\partial E_k}{\partial \dot{q}_i}\right) - \frac{\partial E_k}{\partial q_i} + \frac{\partial E_p}{\partial q_i} = Q_i, \quad i = 1 \dots n \quad (5.1)$$

Výsledkem je totik obyčejných diferenciálních rovnic (ODE) druhého řádu kolik má soustava mechanických stupňů volnosti (n). Na pravé straně rovnice je zobecněná síla Q_i působící v souřadnici q_i .

Postup sestavení pohybových rovnic lze shrnout takto:

- definujeme n zobecněných souřadnic q_i pro každý z n stupňů volnosti
- určíme kinetickou a potenciální energii a vyjádříme ji pomocí souřadnic q_i a jejich derivací \dot{q}_i
- pro každou souřadnici $i = 1 \dots n$ provedeme rovnici (5.1) a získáme n ODE druhého řádu.

LR2 představují velmi užitečný nástroj pro *středně složité* mechanické systémy. Používat je pro systémy s jedním stupněm volnosti je zbytečné, vystačíme si s Newtonem (silovou nebo momentovou rovnováhou). Naopak, pokud má mechanismus mnoho stupňů volnosti a jednotlivá tělesa se pohybují obecně v rovině nebo v prostoru, může náročnost řešení snadno přerušt lidské možnosti. Výsledky derivací jednotlivých členů LR2 jsou totiž tak složité a nepřehledné, že to znemožňuje jejich úpravu a smysluplné použití - a to i při použití nástrojů pro automatickou práci s matem. výrazy (Maple, Mathematika, Symbolic toolbox a další).

Důležitá poznámka k vyjádření kinetické energie: Pokud koná těleso/hmotný bod obecný rovinný/prostorový pohyb, není úplně jednoduché zapsat rovnici jeho rychlosti (v kinematici jsme používali tužku, papír a vektory). Pro algoritmizaci pro počítač je lepší použít jiný postup:

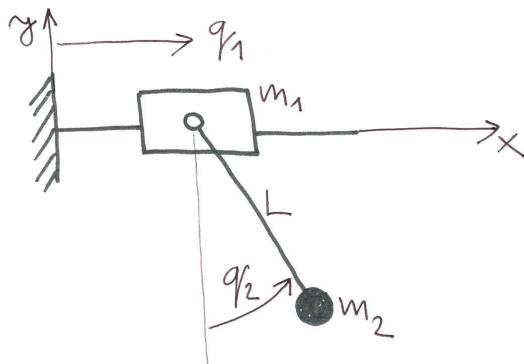
- vyjádříme polohu hmotného bodu v kartézských souřadnicích x, y pomocí zobecněných souřadnic q_i
- polohu x, y zderivujeme a získáme tak rychlosti \dot{x}, \dot{y} jako funkci q_i a \dot{q}_i
- druhou mocninu absolutní rychlosti hmotného bodu v_2 získáme takto: $v_2^2 = \dot{x}^2 + \dot{y}^2$ a to je vše, co pro E_k potřebujeme.

Důležitou vlastností LR2 je na rozdíl od Newtonova přístupu to, že *vylučuje z rovnic vazbové síly*. To je většinou vítané - vazbové síly nekonají práci a tedy je nepotřebujeme. Problém nastane v okamžiku, kdy chceme uvažovat v modelu suché tření⁴⁵ nebo chceme výsledky simulace použít pro konstrukční rozhodnutí (např. návrh ložisek nebo pevnostní výpočet). V takových případech LR2 nedostačují a musíme použít jiný přístup.

Tím jsme shrnuli vše podstatné a aplikaci ukážeme na příkladech.

⁴⁵ Síla/moment suchého tření závisí na normálové=vazbové síle, např. $T = Nf$.

5.2 Příklad na LR2: Mostový jeřáb



Obrázek 5.1 Schéma mostového jeřábu

Začneme příkladem systému se dvěma stupni volnosti, u kterého hezky vynikne užitčnost LR2. Obě tělesa budeme považovat za hmotné body.

Vyjádření E_k a E_p

- Definujeme 2 zobecněné souřadnice q_1 a q_2 podle obrázku 5.1.
- Vyjádříme polohu hmotného bodu 2 v kartézských souřadnicích x_2, y_2 pomocí zobecněných souřadnic takto:

$$x_2 = q_1 + L \sin(q_2) \quad (5.2)$$

$$y_2 = -L \cos(q_2) \quad (5.3)$$

- Polohu x_2, y_2 zderivujeme a získáme tak rychlosti \dot{x}_2, \dot{y}_2 jako funkci q_i a \dot{q}_i takto:

$$\dot{x}_2 = \dot{q}_1 + L \dot{q}_2 \cos(q_2) \quad (5.4)$$

$$\dot{y}_2 = L \dot{q}_2 \sin(q_2) \quad (5.5)$$

- Druhou mocninu absolutní rychlosti hmotného bodu v_2 získáme takto⁴⁶:

$$v_2^2 = \dot{x}_2^2 + \dot{y}_2^2 = L^2 \dot{q}_2^2 + 2L \dot{q}_1 \dot{q}_2 \cos(q_2) + \dot{q}_1^2 \quad (5.6)$$

- Kinetickou a potenciální energii celého systému nyní vyjádříme takto:

$$\begin{aligned} E_k &= \frac{1}{2} m_1 v_1^2 + \frac{1}{2} m_2 v_2^2 = \\ &= \frac{1}{2} m_1 \dot{q}_1^2 + \frac{1}{2} m_2 (L^2 \dot{q}_2^2 + 2L \dot{q}_1 \dot{q}_2 \cos(q_2) + \dot{q}_1^2) \end{aligned} \quad (5.7)$$

$$E_p = -m_2 g L \cos(q_2) \quad (5.8)$$

- Derivování podle LR2** Nyní postupně derivujeme E_k a E_p pro první zobecněnou souřadnici q_1 podle rov. (5.1):

⁴⁶ Tady často může člověk udělat chybu: řekne si, že kinetická energie bodu 2 je energie translace $\frac{1}{2} m_2 \dot{q}_1^2$ a energie rotace kolem závěsu $\frac{1}{2} m_2 L^2 \dot{q}_2^2$ a tyto energie sečte. Zapomene přitom, že bod 2 koná obecný rovinový pohyb a jeho rychlosť je tedy vektorovým součtem rychlosti \dot{q}_1 a $L \dot{q}_2$.

$$\frac{\partial E_k}{\partial \dot{q}_1} = m_1 \dot{q}_1 + m_2 (\dot{q}_1 + L \dot{q}_2 \cos(q_2)) \quad (5.9)$$

$$\frac{d}{dt} \left(\frac{\partial E_k}{\partial \dot{q}_1} \right) = m_1 \ddot{q}_1 + m_2 (\ddot{q}_1 - L \dot{q}_2^2 \sin(q_2) + L \dot{q}_2 \cos(q_2)) \quad (5.10)$$

$$\frac{\partial E_k}{\partial q_1} = 0 \quad (5.11)$$

$$\frac{\partial E_p}{\partial q_1} = 0 \quad (5.12)$$

Podobně provedeme pro q_2 :

$$\frac{\partial E_k}{\partial \dot{q}_2} = m_2 (L^2 \dot{q}_2 + L \dot{q}_1 \cos(q_2)) \quad (5.13)$$

$$\frac{d}{dt} \left(\frac{\partial E_k}{\partial \dot{q}_2} \right) = m_2 (L^2 \ddot{q}_2 + L \ddot{q}_1 \cos(q_2) - L \dot{q}_1 \dot{q}_2 \sin(q_2)) \quad (5.14)$$

$$\frac{\partial E_k}{\partial q_2} = -m_2 L \dot{q}_1 \dot{q}_2 \sin(q_2) \quad (5.15)$$

$$\frac{\partial E_p}{\partial q_2} = m_2 g L \sin(q_2) \quad (5.16)$$

- **Sestavení výsledných rovnic** Nyní vezmeme výrazy (5.10) – (5.12) a (5.14) – (5.16) a podle (5.1) sestavíme dvě pohybové rovnice. Bývá zvykem je zapsat do maticového tvaru, v tomto případě tohoto (lineárně implicitní tvar soustavy dvou ODE druhého řádu):

$$\mathbf{M}(q_2) \ddot{\mathbf{q}} + \mathbf{f}(q_2, \dot{q}_2) = \mathbf{0} \quad (5.17)$$

První rovnici (5.10) - (5.11) + (5.12) tedy sestavíme takto:

$$\begin{aligned} \frac{d}{dt} \left(\frac{\partial E_k}{\partial \dot{q}_1} \right) - 0 + 0 &= \\ = \boxed{(m_1 + m_2)} \quad \ddot{q}_1 + \boxed{(m_2 L \cos(q_2))} \quad \ddot{q}_2 + \boxed{-m_2 L \dot{q}_2^2 \sin(q_2)} &= \end{aligned} \quad (5.18)$$

$$= M_{11} \ddot{q}_1 + M_{12} \ddot{q}_2 + f_1 \quad (5.19)$$

V druhé rovnici se navzájem odečte část výrazu z (5.14) a (5.15) a výsledek je ve tvaru:

$$\begin{aligned} \frac{d}{dt} \left(\frac{\partial E_k}{\partial \dot{q}_2} \right) - \frac{\partial E_k}{\partial q_2} + \frac{\partial E_p}{\partial q_2} &= \\ = \boxed{(m_2 L \cos(q_2))} \quad \ddot{q}_1 + \boxed{(m_2 L^2)} \quad \ddot{q}_2 + \boxed{m_2 g L \sin(q_2)} &= \end{aligned} \quad (5.20)$$

$$= M_{21} \ddot{q}_1 + M_{22} \ddot{q}_2 + f_2 \quad (5.21)$$

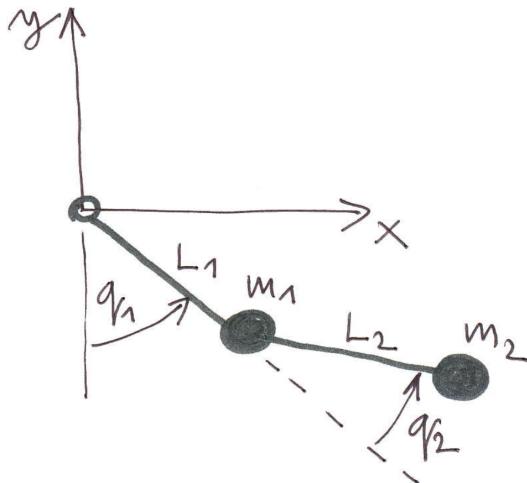
Pokud na systém nepůsobí žádné vnější síly⁴⁷, tedy $Q_1 = 0$ a $Q_2 = 0$, můžeme poskládat výsledné dvě pohybové rovnice do maticového tvaru (5.17):

⁴⁷ Tíhovou sílu jsme uvažovali zahrnutím E_p do LR2.

$$\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} + \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (5.22)$$

$$\begin{bmatrix} \frac{m_1 + m_2}{m_2 L \cos(q_2)} & \frac{m_2 L \cos(q_2)}{m_2 L^2} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} + \begin{bmatrix} -m_2 L \dot{q}_2^2 \sin(q_2) \\ m_2 g L \sin(q_2) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (5.23)$$

5.3 Příklad na LR2: Dvojité kyvadlo



Obrázek 5.2 Schéma dvojitého kyvadla

■ Vyjádření E_k a E_p

- Definujeme 2 zobecněné souřadnice q_1 a q_2 podle **obrázku 5.2**
- Vyjádříme *polohu* hmotného bodu 1 a 2 v kartézských souřadnicích pomocí zobecněných souřadnic takto:

$$x_1 = L_1 \sin(q_1) \quad (5.24)$$

$$y_1 = -L_1 \cos(q_1) \quad (5.25)$$

$$x_2 = L_1 \sin(q_1) + L_2 \sin(q_1 + q_2) \quad (5.26)$$

$$y_2 = -L_1 \cos(q_1) - L_2 \cos(q_1 + q_2) \quad (5.27)$$

- Polohy x_1, y_1, x_2, y_2 zderivujeme a získáme tak rychlosti jako funkce q_i a \dot{q}_i takto:

$$\dot{x}_1 = L_1 \dot{q}_1 \cos(q_1) \quad (5.28)$$

$$\dot{y}_1 = L_1 \dot{q}_1 \sin(q_1) \quad (5.29)$$

$$\dot{x}_2 = L_1 \dot{q}_1 \cos(q_1) + L_2 \cos(q_1 + q_2) (\dot{q}_1 + \dot{q}_2) \quad (5.30)$$

$$\dot{y}_2 = L_1 \dot{q}_1 \sin(q_1) + L_2 \sin(q_1 + q_2) (\dot{q}_1 + \dot{q}_2) \quad (5.31)$$

- Druhou mocninu absolutní rychlosti obou hmotných bodů v_1 a v_2 získáme takto:

$$v_1^2 = \dot{x}_1^2 + \dot{y}_1^2 = \boxed{L_1^2 \dot{q}_1^2} \quad (5.32)$$

$$\begin{aligned} v_2^2 &= \dot{x}_2^2 + \dot{y}_2^2 = \\ &= \boxed{(L_1^2 + 2 \cos(q_2) L_1 L_2 + L_2^2)} \quad \dot{q}_1^2 \\ &\quad + \boxed{(2 \cos(q_2) L_1 L_2 + 2 L_2^2)} \quad \dot{q}_1 \dot{q}_2 \\ &\quad + \boxed{L_2^2} \quad \dot{q}_2^2 \end{aligned} \quad (5.33)$$

- Kinetickou a potenciální energii celého systému nyní vyjádříme takto:

$$E_k = \frac{1}{2}m_1v_1^2 + \frac{1}{2}m_2v_2^2 = \dots = f(q_2, \dot{q}_1, \dot{q}_2) \quad (5.34)$$

$$E_p = -m_1gL_1 \cos(q_1) - m_2g(L_1 \cos(q_1) + L_2 \cos(q_1 + q_2)) \quad (5.35)$$

- **Derivování podle LR2** Nyní postupně derivujeme E_k a E_p pro první z obecněnou souřadnicí q_1 podle rov. (5.1):

$$\begin{aligned} \frac{\partial E_k}{\partial \dot{q}_1} &= (m_1[L_1^2] + m_2[L_1^2 + 2L_1L_2 \cos(q_2) + L_2^2]) \dot{q}_1 \\ &\quad + [m_2(\cos(q_2)L_1L_2 + L_2^2)] \ddot{q}_2 \end{aligned} \quad (5.36)$$

$$\begin{aligned} \frac{d}{dt} \left(\frac{\partial E_k}{\partial \dot{q}_1} \right) &= (m_1[L_1^2] + m_2[L_1^2 + 2L_1L_2 \cos(q_2) + L_2^2]) \ddot{q}_1 \\ &\quad - 2m_2L_1L_2 \sin(q_2) \dot{q}_1 \dot{q}_2 \\ &\quad - m_2L_1L_2 \sin(q_2) \dot{q}_2^2 \\ &\quad + [m_2(\cos(q_2)L_1L_2 + L_2^2)] \ddot{q}_2 \end{aligned} \quad (5.37)$$

$$\frac{\partial E_k}{\partial q_1} = 0 \quad (5.38)$$

$$\frac{\partial E_p}{\partial q_1} = m_1gL_1 \sin(q_1) + m_2g(L_1 \sin(q_1) + L_2 \sin(q_1 + q_2)) \quad (5.39)$$

Podobně provedeme pro q_2 :

$$\begin{aligned} \frac{\partial E_k}{\partial \dot{q}_2} &= m_2[L_1L_2 \cos(q_2) + L_2^2] \dot{q}_1 + m_2[L_2^2] \dot{q}_2 \\ \frac{d}{dt} \left(\frac{\partial E_k}{\partial \dot{q}_2} \right) &= m_2[L_1L_2 \cos(q_2) + L_2^2] \ddot{q}_1 \\ &\quad - L_1L_2 \sin(q_2) \dot{q}_1 \dot{q}_2 \\ &\quad + m_2[L_2^2] \ddot{q}_2 \end{aligned} \quad (5.41)$$

$$\frac{\partial E_k}{\partial q_2} = -m_2L_1L_2 \sin(q_2) \dot{q}_1^2 - m_2L_1L_2 \sin(q_2) \dot{q}_1 \dot{q}_2 \quad (5.42)$$

$$\frac{\partial E_p}{\partial q_2} = m_2gL_2 \sin(q_1 + q_2) \quad (5.43)$$

- **Sestavení výsledných rovnic** Pohybové rovnice budeme chtít dostat do tvaru:

$$\mathbf{M}(\mathbf{q}) \ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \mathbf{0} \quad (5.44)$$

Matice tedy budou vypadat takto:

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} m_1L_1^2 + m_2(L_1^2 + 2L_1L_2 \cos(q_2) + L_2^2) & m_2(L_2^2 + L_1L_2 \cos(q_2)) \\ m_2(L_2^2 + L_1L_2 \cos(q_2)) & m_2L_2^2 \end{bmatrix} \quad (5.45)$$

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{bmatrix} -2m_2L_1L_2 \dot{q}_2 \sin(q_2) & -m_2L_1L_2 \dot{q}_2 \sin(q_2) \\ m_2L_1L_2 \dot{q}_1 \sin(q_2) & 0 \end{bmatrix} \quad (5.46)$$

$$\mathbf{g}(\mathbf{q}) = \begin{bmatrix} (m_1 + m_2)gL_1 \sin(q_1) + m_2gL_2 \sin(q_1 + q_2) \\ m_2gL_2 \sin(q_1 + q_2) \end{bmatrix} \quad (5.47)$$

5.4 Sestavení LR2 s pomocí nástrojů pro symbolické výpočty

Použití LR2 popsané v této kapitole vyžaduje derivování rozsáhlých výrazů a další operace s nimi. Tuto nepříjemnou práci (která je také často zdrojem chyb) je vhodné si ulehčit pomocí vhodných nástrojů, které umí provádět symbolické operace s matematickými výrazy⁴⁸.

Následující (neúplný) kus kódu ukazuje implementaci v Symbolic Math Toolboxu pro MATLAB. Potíž je v tom, že proměnné potřebujeme mít/nemít/mít⁴⁹ vyjádřené jako funkce času. Tyto přechody provedeme pomocí příkazu `subs`⁵⁰. Pro úpravu výrazů se pak hodí `simplify`⁵¹ a `coeffs`⁵². V případě použití softwaru Maple se bude syntaxe mírně lišit, princip ale zůstane stejný.

```
clc
clear
syms t q1 q2 dq1 dq2 ddq1 ddq2 L1 L2 m1 m2 g
x1 = L1*sin(q1)
x1 = subs(x1, 'q1','q1(t)')
dx1 = diff(x1,t)
dx1 = subs(dx1, 'diff(q1(t), t)', 'dq1')
dx1 = subs(dx1, 'q1(t)', 'q1')
...
v1_sqrt = dx1^2 + dy1^2
...
Ek = 1/2*m1*v1_sqrt + 1/2*m2*v2_sqrt
Ep = m1*g*y1 + m2*g*y2
...
dEk_dq1 = diff(Ek, dq1)
dEk_dq1 = subs(dEk_dq1, {'dq1',...},{'dq1(t)',...})
d_dEk_dq1_dt = diff(dEk_dq1, t)
d_dEk_dq1_dt = simplify(d_dEk_dq1_dt)
...
w = coeffs(d_dEk_dq1_dt, 'ddq1')
```

⁴⁸ Jde o software souhrně nazývaný CAS (Computer Algebra Systems), známými příklady jsou Maple, Mathematica, MathCAD a Symbolic Math Toolbox pro MATLAB.

⁴⁹ Nejprve derivujeme souřadnice podle času (poloha na rychlosť). Pak sestavíme energie a derivujeme podle rychlosťi (rychlosť nesmí být v tu chvíli funkci času). Pak derivujeme výsledek znova podle času (rychlosť musí být funkce času).

⁵⁰ Substituuje jeden výraz za druhý (nebo sadu výrazů). Více info v helpu.

⁵¹ Zjednoduší výraz. Občas se této funkci podaří i odečíst okem zjevně viditelné výrazy s opačnými znaménky...

⁵² „Posbírá“ v daném výrazu všechno, co obsahuje daný „podyáraz“.

V příkladech KD30 a KD31 jsou připraveny funkce, které konverze pro soustavy se dvěma stupni volnosti provádějí.

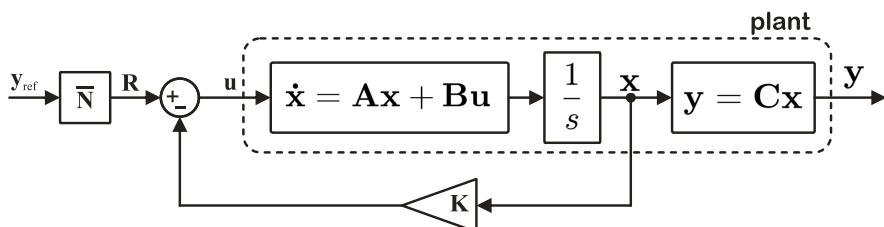
6 Stavové řízení

Obsah kapitoly:

6.1	Úvod	83
6.1.1	K čemu to je?	83
6.1.2	Lineární stavový model	84
6.2	Úvod	84
6.2.1	Stabilizace v nule	84
6.2.2	Stabilizace v bodě	84
6.3	Jak získat matici K	86
6.3.1	Metoda umísťování pólů	86
6.3.2	LQR	86
6.4	Stavový pozorovatel	86
6.4.1	Motivac	86
6.4.2	Deterministický princip separace	86
6.4.3	Jak určit zesílení L?	88
6.5	Otázky a neřešené úlohy	88

6.1 Úvod

6.1.1 K čemu to je?



Obrázek 6.1 asdfasdf

- patří do *Moderní teorie řízení*
- využívá celý stav \mathbf{x} systému
- je vhodné pro MIMO systémy (na rozdíl od PID regulátoru⁵³⁾
- nejčastěji se navrhuje jako LQR (Linear-Quadratic-Regulator)
- a v kombinaci s Kalmanovým filtrem tvorí LQG řízení (Linear-Quadratic-Gaussian control), které představuje zásadní přístup v oblasti lineárního řízení
- lze použít pro řízení tzv. podaktuovaných systémů – typickým příkladem je *inverzní kyvadlo*.

⁵³ V praxi je zdaleka nejpoužívanějším regulátorem PID. Je to dáno tím, že převážná většina úloh je SISO, případně se MIMO úlohy rozdělí na n samostatných SISO systémů. Díky vzájemnému ovlivnění jednotlivých pohybových os (crosscoupling) může ale být ladění PID parametrů obtížné.

6.1.2 Lineární stavový model

Lineární spojitý časově invariantní stavový model zapisujeme obvykle ve tvaru

$$\dot{\mathbf{x}} = \mathbf{Ax}(t) + \mathbf{Bu}(t) \quad (6.1)$$

$$\mathbf{y}(t) = \mathbf{Cx}(t) + \mathbf{Du}(t) \quad (6.2)$$

kde \mathbf{x} je stavový vektor (stav) systému, matice \mathbf{A} , \mathbf{B} a \mathbf{C} jsou konstantní, matice \mathbf{D} bývá nulová, \mathbf{u} je vstupem do systému a \mathbf{y} je výstupem. V dalším textu budeme např. označení $\mathbf{x}(t)$ pro zjednodušení nahrazovat \mathbf{x} .

Připomeňme, že dynamické vlastnosti systému (vlastní frekvence, stabilita, tlumení) jsou dány vlastními čísly matice \mathbf{A} .

Naším úkolem bude navrhnut pro daný dynamický systém řízení, které pracuje s celým stavem \mathbf{x} .

6.2 Úvod

6.2.1 Stabilizace v nule

Uvažujme, že chceme systém (6.1) stabilizovat v počátku $\mathbf{x} = \mathbf{0}$. Předpokládejme, že navrhнемe řízení ve tvaru

$$\mathbf{u} = -\mathbf{Kx} \quad (6.3)$$

kde \mathbf{K} je vhodná matice. Dosazením (6.3) do (6.1) snadno získáme

$$\dot{\mathbf{x}} = \mathbf{Ax} - \mathbf{BKx} = (\mathbf{A} - \mathbf{BK})\mathbf{x} \quad (6.4)$$

Dynamika soustavy je určena vlastními čísly matice $\mathbf{A} - \mathbf{BK}$. Pro vhodně zvolené \mathbf{K} bude tedy systém stabilní s rovnovážným bodem v nule.

6.2.2 Stabilizace v bodě

6.2.2.1 Doplňení o referenční vstup

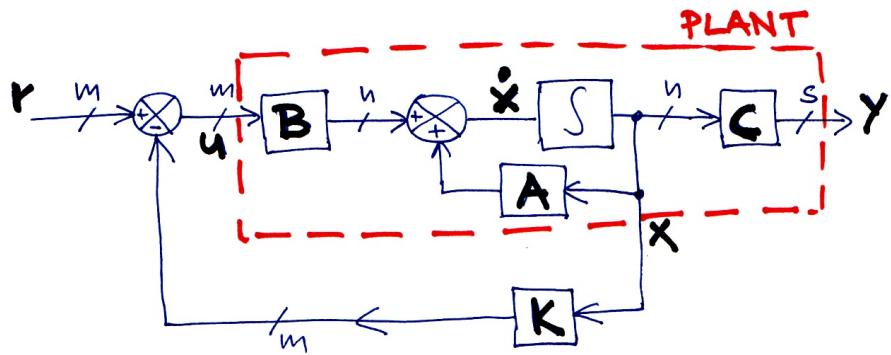
Předchozí případ vylepšíme o referenční vstup. Dostaneme tak

$$\begin{aligned}\mathbf{u} &= \mathbf{r} - \mathbf{Kx} \\ \dot{\mathbf{x}} &= (\mathbf{A} - \mathbf{BK})\mathbf{x} + \mathbf{Br} \\ \mathbf{y} &= \mathbf{Cx}\end{aligned}$$

Snadno zjistíme, že ustálená hodnota \mathbf{y} neodpovídá referenčnímu vstupu \mathbf{r} .

6.2.2.2 Vyvažovací matice \mathbf{N}

Uvažujme ustálený stav SISO ($m = s = 1$) systému (6.1) daný vstupem u_s , výstupem y_s a stavem \mathbf{x}_s .



Obrázek 6.2 asdfasdf

$$\mathbf{0} = \mathbf{Ax}_s + \mathbf{Bu}_s$$

$$\mathbf{y}_s = \mathbf{Cx}_s$$

Tyto dvě rovnice zapíšeme jako

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}_s \\ u_s \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ r_s \end{bmatrix} \quad (6.5)$$

Inverzní matice můžeme snadno určit ustálený vstup a stav

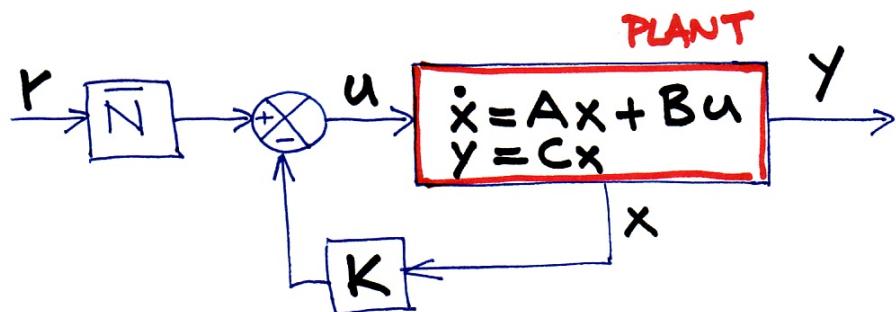
$$\begin{bmatrix} \mathbf{x}_s \\ u_s \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{0} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{0} \\ r_s \end{bmatrix} \quad (6.6)$$

Předpokládejme, že referenci r budeme násobit vektorem \mathbf{N} , bude pak platit rovnice

$$u_s = \mathbf{Kx}_s + \mathbf{Nr} \quad (6.7)$$

ze které (úvahou s $r = 1$) snadno získáme vztah pro výpočet \mathbf{N}

$$\mathbf{N} = \left(\frac{u_s}{r_s} + \mathbf{K} \frac{\mathbf{x}_s}{r_s} \right) \quad (6.8)$$



Obrázek 6.3 asdfasdf

Výsledný model tedy můžeme zapsat takto:

$$\dot{\mathbf{x}} = (\mathbf{A} - \mathbf{BK})\mathbf{x} + \mathbf{BNr}$$

$$\mathbf{y} = \mathbf{Cx}$$

V Matlabu můžeme použít funkci `Nbar = rscale(sys, K)`, která není součástí Control Toolboxu, ale lze ji najít na webu [[CTM](#)].

6.3 Jak získat matici \mathbf{K}

6.3.1 Metoda umíšťování pólů

6.3.2 LQR

Lineární kvadraticky optimální regulátor (LQR) definuje takovou konstantní matici \mathbf{K} , která minimalizuje následující kritérium⁵⁴:

$$J = \int_0^{\infty} (\mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u}) dt \quad (6.9)$$

kde \mathbf{Q} je váhová matice stavů a \mathbf{R} je váhová matice vstupů. Obě matice volíme často jako diagonální a vzájemný poměr hodnot definuje váhu daného stavu či vstupu.

V Matlabu máme k dispozici příkaz `K = lqr(A, B, Q, R)`.

6.4 Stavový pozorovatel

6.4.1 Motivac

V reálných úlohách často nastane případ, kdy nemůžeme nebo nechceme měřit *celý stav* systému⁵⁵. Řešením je použití stavového pozorovatele, kterým získáme na základě měření \mathbf{y} odhad stavu $\hat{\mathbf{x}}$.

Základní myšlenkou je použití modelu `eq:ss_modelAB(??)`. Předpokládáme, že model známe a při známém vstupu \mathbf{u} můžeme zístat odhad stavu $\hat{\mathbf{x}}$. Problémem je, že navržený pozorovatel pracuje v otevřené smyčce a tak libovolná nepřesnost modelu, počátečních podmínek \mathbf{x}_0 nebo porucha způsobí trvalou odchylku odhadu od skutečného stavu.

Přirozeným nápadem pak je využít měřený vektor \mathbf{y} ke korekci výše zmíněných nepřesností – do stavového pozorovatele tedy přidáme *zpětnou vazbu*. Nová rovnice pozorovatele bude mít tvar (obr. ??)

$$\dot{\hat{\mathbf{x}}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{L}(\mathbf{y} - \mathbf{C}\hat{\mathbf{x}}) \quad (6.10)$$

kde \mathbf{L} je vhodně zvolená matici.

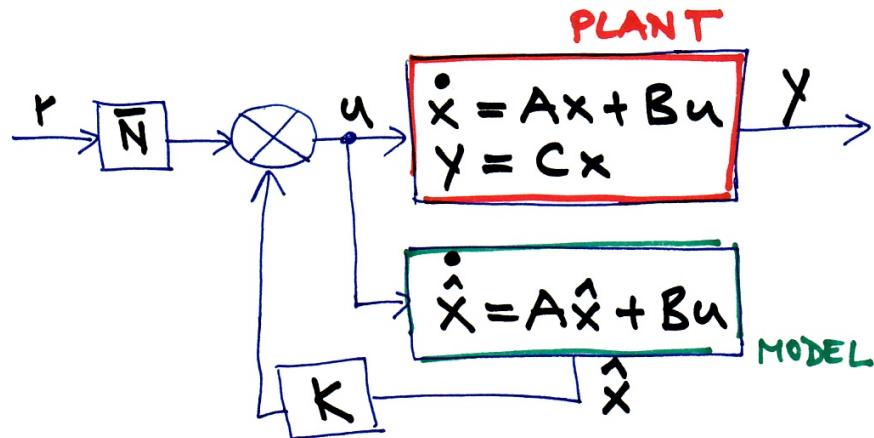
Otázkou tedy nyní je, zda lze zvolit matici \mathbf{L} , tak, aby „ne kazila“ stavové řízení \mathbf{K} a aby odhad stavu $\hat{\mathbf{x}}$ konvergoval ke skutečnému stavu \mathbf{x} .

6.4.2 Deterministický princip separace

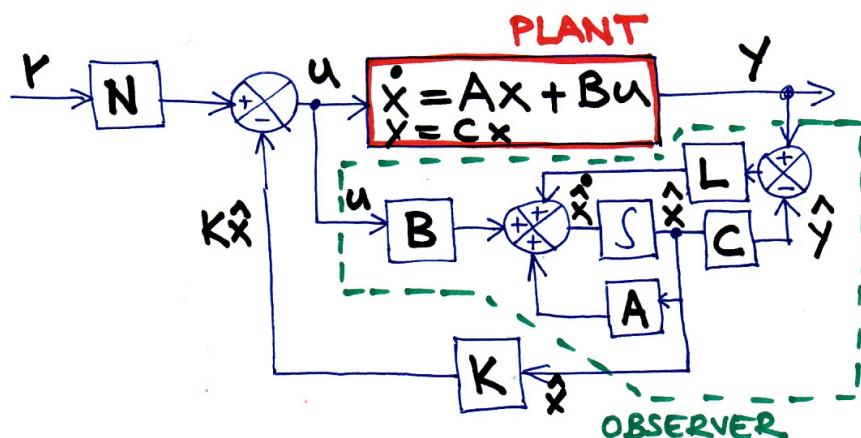
Nejprve dokážeme, že vlastnosti řízení (matici \mathbf{K}) a pozorovatele (matice \mathbf{L}) lze volit nezávisle [[str. 115](#)] Valasek 1995.

⁵⁴ Následující rov.(6.9) reprezentuje tzv. spojitý LQR pro nekonečný horizont. Pro zjednodušení jsme zde neuveďli třetí člen kritéria $J(2\mathbf{x}^T \mathbf{N} \mathbf{u})$, který se velmi často zanedbává.

⁵⁵ Příkladem budiž polohové řízení DC motoru vybaveného inkrementálním enkodérem (měří polohu). Rychlosť přímo měřit nemůžeme nebo nechceme (přidat tachodynamo je nákladné).



Obrázek 6.4 Stavový pozorovatel
(otevřená smyčka = nepoužitelné)



Obrázek 6.5 Stavový pozorovatel)

Dynamika modelu soustavy je popsána rovnicí:

$$\dot{\hat{x}} = A\hat{x} + Bu + L(y - C\hat{x}) \quad (6.11)$$

Definujme rozdíl \tilde{x} mezi skutečným stavem a jeho odhadem

$$\tilde{x} = x - \hat{x} \quad (6.12)$$

Zderivujeme, vyjádříme \dot{x} a $\dot{\hat{x}}$ a dosadíme do ??dynamika_pozorovatele.

$$\begin{aligned} \dot{\tilde{x}} &= \dot{x} - \dot{\hat{x}} = A(x - \tilde{x}) + Bu + L(y - Cx + C\tilde{x}) = \\ &= Ax - A\tilde{x} + Bu + \underbrace{Ly - LCx + LC\tilde{x}}_0 \end{aligned}$$

Odečteme-li rov. (6.2), dostáváme

$$-\dot{\tilde{x}} = -A\tilde{x} + LC\tilde{x} \quad (6.13)$$

Dynamika chyby odhadu stavu (pozorovatele) je tedy dána rovnicí

$$\dot{\tilde{x}} = (A - LC)\tilde{x} \quad (6.14)$$

a tedy vlastnostmi matice ($\mathbf{A} - \mathbf{LC}$) a lze ji tedy volit *nezávisle* na řízení \mathbf{K} .

6.4.3 Jak určit zesílení \mathbf{L} ?

- Umísťování pólů nebo LQR
- Kalmanův filtr.

6.5 Otázky a neřešené úlohy

Otázky

1. Jaké znáte dvě metody pro získání matice řízení \mathbf{K} při návrhu stavového řízení?
2. Zapište matici, která určuje dynamiku systému řízeného plnou stavovou zpětnou vazbou! (vyjděte ze stavového modelu, zapište rovnicí)
3. Co to je deterministický princip separace?
4. Zapište kritérium lineárního kvadratického regulátoru!

Úlohy

5. Navrhněte stavový regulátor pro systém na obr. 10.5.
 - a) Přepište model 10.14 – 10.15 do tvaru 6.1–6.2.
 - b) Pomocí LQR navrhněte plnou stavovou zpětnou vazbu.

7 Kalmanův filtr

Obsah kapitoly:

7.1	Úvod	90
7.1.1	Příklad: KF pro jednorozměrný systém	90
7.2	Lineární KF	90
7.3	Extended KF (EKF)	91
7.4	Detailly, příklady a odvození KF	92
7.4.1	Osvěžující poznámky o náhodné veličině	92
7.4.2	Příklad: Detailní rozbor a odvození KF pro 1d systém	96
7.4.3	Příklad: KF pro dvourozměrný systém a jeho geometrická interpretace	101
7.4.4	Příklad: Anténa 1dof (potenciometr a gyro)	104
7.5	Otázky a neřešené úlohy	108

7.1 Úvod

7.1.1 Příklad: KF pro jednorozměrný systém

TODO ... popis...

7.1.1.1 KF

KF pracuje ve dvou krocích: nejprve predikuje hodnotu stavu na základě modelu a následně ji koriguje podle měření.

■ Predikce

$$\begin{aligned}\hat{x}_{k+1|k} &= a\hat{x}_k \\ P_{k+1|k} &= a^2P_{k|k} + Q.\end{aligned}$$

■ Korekce

$$\begin{aligned}\hat{y}_{k+1|k} &= \hat{x}_{k+1|k} \\ K &= \frac{P_{k+1|k}}{P_{k+1|k} + R} \\ \hat{x}_{k+1|k+1} &= \hat{x}_{k+1|k} + K(y_{k+1} - \hat{y}_{k+1|k}) \\ P_{k+1|k+1} &= (1 - K)P_{k+1|k}.\end{aligned}$$

7.2 Lineární KF

Uvažujme diskrétní lineární stavový model:

$$\mathbf{x}_k = \Phi \mathbf{x}_{k-1} + \Gamma \mathbf{u}_k + \Lambda \mathbf{w}_k \quad (7.1)$$

$$\mathbf{y}_k = \mathbf{C} \mathbf{x}_k + \mathbf{D} \mathbf{u}_k + \mathbf{H} \mathbf{v}_k \quad (7.2)$$

Iterace KF začíná se známými hodnotami z předchozího kroku: odhad stavu $\hat{\mathbf{x}}_{k-1|k-1}$, kovarianční matice chyby odhadu $P_{k-1|k-1}$ a vstup do systému \mathbf{u}_k . provedeme měření \mathbf{y}_k v kroku k , které je zatížené šumem (chybou) a můžeme provést jeden krok výpočtu podle následujícího zápisu.

Nejprve vypočteme *predikci* stavu v časovém kroku k na základě modelu:

$$\hat{\mathbf{x}}_{k|k-1} = \Phi \hat{\mathbf{x}}_{k-1|k-1} + \Gamma \mathbf{u}_k \quad (7.3)$$

Následně vypočteme predikovanou matici \mathbf{P} :

$$\mathbf{P}_{k|k-1} = \Phi \mathbf{P}_{k-1|k-1} \Phi^T + \mathbf{Q} \quad (7.4)$$

Nyní použijeme měření ke *korekci* predikovaných hodnot.

$$\hat{\mathbf{y}}_{k|k-1} = \mathbf{C}\hat{\mathbf{x}}_{k|k-1} \quad (7.5)$$

$$\tilde{\mathbf{y}}_k = \mathbf{y}_k - \hat{\mathbf{y}}_{k|k-1} \quad (7.6)$$

$$\mathbf{S}_k = \mathbf{C}\mathbf{P}_{k|k-1}\mathbf{C}^T + \mathbf{R} \quad (7.7)$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{C}^T\mathbf{S}_k^{-1} \quad (7.8)$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k \quad (7.9)$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{C})\mathbf{P}_{k|k-1} \quad (7.10)$$

Nyní máme vypočteno $\hat{\mathbf{x}}_{k|k}$, $\mathbf{P}_{k|k}$, řídicí algoritmus vypočte na základě odhadu stavu akci \mathbf{u}_{k+1} , změříme \mathbf{y}_{k+1} a tím máme opět všechny potřebné hodnoty pro další krok $k+1$ KF.

7.3 Extended KF (EKF)

TODO

7.4 Detaily, příklady a odvození KF

7.4.1 Osvěžující poznámky o náhodné veličině

V KF hraje klíčovou roli *kovarianční matici*. Pokud chceme opravdu porozumět tomu, jak funguje KF, musíme dobře chápat i kovarianční matici.

Zatímco rozptyl (angl. variance) popisuje („nějak“, viz dále) vlastnosti skalární náhodné veličiny, kovarianční matice popisuje vlastnosti vektoru náhodných veličin. U vektoru je přitom situace komplikovanější – jednak nás zajímají vlastnosti jednotlivých náhodných veličin vektoru, ale navíc chceme vědět, zda mají tyto prvky vektoru nějaký *vzájemný* vztah.

7.4.1.1 Hustota pravděpodobnosti, střední hodnota, rozptyl

Předpokládáme, že vážený čtenář je v prostoru i čase vzdálen od skript Matematika IV nebo podobné literatury a uvedeme zde proto podrobné zopakování definic.

- Spojitou náhodnou veličinu X popisujeme *hustotou pravděpodobnosti*⁵⁶ $p(x)$. Hustota pravděpodobnosti nám může zodpovědět otázku: „Jaká je pravděpodobnost výskytu veličiny x v intervalu od x_1 do x_2 ?“ Odpověď⁵⁷ je tato:

$$P[x_1 \leq X \leq x_2] = \int_{x_1}^{x_2} p(x) dx. \quad (7.11)$$

Z uvedeného plyne, že integrál přes celý definiční obor Ω je roven 1:

$$\int_{\Omega} p(x) dx = 1. \quad (7.12)$$

- *Střední hodnotu*⁵⁸ vypočteme při znalosti $p(x)$ takto:

$$E(X) = \mu = \int_{\Omega} xp(x) dx, \quad (7.13)$$

Střední hodnota je momentem prvního řádu.

- *Rozptyl* (angl. variance) vypočteme takto:

$$var(X) = \sigma^2 = \int_{\Omega} (x - \mu)^2 p(x) dx, \quad (7.14)$$

Jedná se o moment druhého řádu, v mechanice je analogií moment setrvačnosti kolem osy.

- σ se nazývá *směrodatná odchylka* náhodné veličiny X .
- Pokud je $\mu = 0$, rozptyl je:

$$\sigma^2 = \int_{\Omega} x^2 p(x) dx, \quad (7.15)$$

⁵⁶ Angl. Probability Density Function, PDF.

⁵⁷ Možná bychom se rádi zeptali trochu jinak, třeba: „Jaká je pravděpodobnost toho, že měřený výrobek má rozměr 30,5 mm?“ Nebo: „Jaká je pravděpodobnost toho, že IQ daného člověka je 150?“ Takto položené otázky ale u *spojité* náhodné veličiny nemají smysl – nemůžeme se ptát na bod, musíme se vždy ptát na interval. Můžeme se tedy ptát „Kolik lidí má IQ mezi 85 a 115?“ Odpověď je 70%.

⁵⁸ Střední hodnota vyjadřuje v přirozeném jazyce něco jako pravděpodobně očekávanou hodnotu – odtud označení E z anglického Expected [<Dutton>].

(Zde je trochu názorněji vidět, že se jedná o moment druhého řádu.)

- Nejznámější, tzv. *normální* (Gaussovo) rozdělení zapisujeme takto: $X \sim N(\mu, \sigma^2)$. Hustota pravděpodobnosti je definovaná vztahem:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, x \in (-\infty, \infty). \quad (7.16)$$

7.4.1.2 Kovariance a koeficient korelace

- *Kovariance* je mírou lineární závislosti dvou náhodných veličin X a Y na sobě. Kovariaci vypočteme z hustoty pravděpodobnosti $p(x, y)$ takto:

$$\text{cov}(X, Y) = \iint xy p(x, y) dx dy = E(XY). \quad (7.17)$$

V mechanice je analogií deviační moment. Pokud jsou dvě náhodné veličiny nezávislé, pak je jejich kovariance nulová.

- *Koeficient korelace* je normovaná kovariance a nabývá hodnot v intervalu $< -1, 1 >$:

$$\rho_{X,Y} = \frac{\text{cov}(X, Y)}{\sigma_x \sigma_y}. \quad (7.18)$$

7.4.1.3 Vybrané kombinace a operace s náhodnými veličinami

- Násobení konstantou c a přičtení konstanty d :

$$Y = cX + d. \quad (7.19)$$

Střední hodnotu a rozptyl pak vypočteme takto:

$$E[Y] = cE[X] + d, \quad (7.20)$$

$$\sigma_y^2 = c^2 \sigma_x^2. \quad (7.21)$$

- Sečtení dvou veličin:

$$Y = V + W. \quad (7.22)$$

Střední hodnota a rozptyl:

$$E[Y] = E[V] + E[W], \quad (7.23)$$

$$\sigma_y^2 = \sigma_v^2 + 2\text{cov}(V, W) + \sigma_w^2. \quad (7.24)$$

7.4.1.4 Kovarianční matice

Při aplikaci KF (a samozřejmě nejen zde) pracujeme s náhodným vektorem \mathbf{x} ⁵⁹. Jednotlivé prvky x_1, x_2, x_3, \dots tohoto vektoru jsou náhodné veličiny, mají svoje střední hodnoty a rozptyly a mohou být vzájemně nezávislé nebo nějak vzájemně závislé. Tyto vlastnosti popisuje souhrnně *kovarianční matici*.

⁵⁹ V literatuře je někdy (skalární) náhodná veličina označována velkým písmenem X a vektor náhodných veličin \mathbf{X} . V následujícím textu se ale přikloníme ke konvenci obvyklé při popisu maticových operací – vektor budeme označovat malým tučným a matice velkým tučným písmenem.

Kovarianční matice má na diagonále rozptyly (variance) jednotlivých náhodných veličin a mimo diagonálu kovariance:

$$\mathbf{Q} = \begin{bmatrix} \sigma_1^2 & cov(x_1, x_2) & cov(x_1, x_3) & \dots \\ cov(x_2, x_1) & \sigma_2^2 & \dots & \dots \\ cov(x_3, x_1) & cov(x_3, x_2) & \sigma_3^2 & \dots \\ \dots & \dots & \dots & \dots \end{bmatrix}. \quad (7.25)$$

Z definice vyplývá jedna z vlastností kov. matice – symetrie podle diagonály. Analogicky k rovnici (7.17) platí:

$$\mathbf{Q} = E[\mathbf{x}\mathbf{x}^T] \quad (7.26)$$

Příklad: pro náhodný vektor $\mathbf{x} \in \mathbb{R}^2$, u kterého jsou oba prvky vektoru nezávislé, má kovarianční matice tvar:

$$\mathbf{Q} = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}. \quad (7.27)$$

[Ex.: KA01] Význam závislosti a nezávislosti jednotlivých náhodných veličin si nejlépe ukážeme na příkladu. Vytvoříme dva vektory náhodných veličin $\mathbf{x} = [X, Y]$ a $\mathbf{x} = [X, Z]$ a porovnáme jejich kovarianční matice:

```
clc,clear
n = 1e6;
X = 3*randn(n,1); % nahodna velicina X
Y = randn(n,1); % nezavisla nahodna velicina Y
Z = randn(n,1) + 0.5*X; % zavisla nahodna velicina Z

subplot(1,2,1)
plot(X,Y,'k.'), grid on, xlabel('X'),ylabel('Y'), axis equal
subplot(1,2,2)
plot(X,Z,'k.'), grid on, xlabel('X'),ylabel('Z'), axis equal
Q_XY = cov(X,Y)
Q_XZ = cov(X,Z)
shg
```

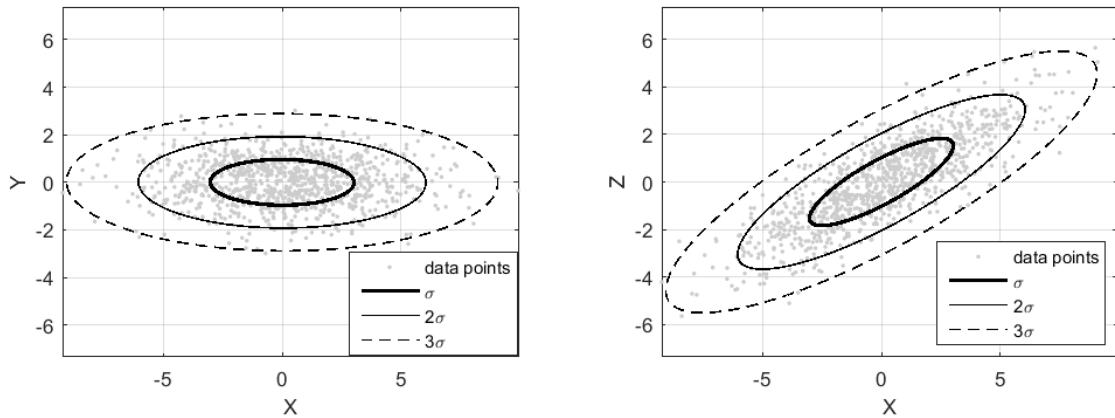
Náhodná veličina Y je nezávislá na X , zatímco náhodná veličina Z je na X (časťčně) závislá. Pokud vypočítáme příslušné kovarianční matice, dostaneme tyto hodnoty:

```
Q_XY =
    9.0049    0.0015
    0.0015    0.9992
Q_XZ =
    9.0049    4.5020
    4.5020    3.2493
```

Vidíme, že matice Q_{XY} má mimo diagonálu (numerické) nuly, zatímco matice Q_{XZ} tam má nenulové hodnoty indikující závislost.

Grafickou představu dostaneme vykreslením jednotlivých bodů $[x,y]$ resp. $[x,z]$. Na obr. 7.1 vidíme, že „závislost otáčí osy elipsy (Gaussova klobouku)“. Na obr. jsou také vykresleny 3 elipsy, které odpovídají rozptylu σ (cca 68% dat), 2σ (cca 95% dat) a 3σ (cca 99,7% dat). Ty získáme pomocí vlastních vektorů a vlastních čísel kovarianční matice takto⁶⁰

```
n=100;
p=0:pi/n:2*pi;
[eigvec,eigval] = eig(Q);
xy = [cos(p'),sin(p')] * scale* sqrt(eigval) * eigvec';
x = xy(:,1);
y = xy(:,2);
```



Obrázek 7.1 Vykreslení bodů dvourozměrné náhodné veličiny (vlevo: nezávislé X, Y ; vpravo: závislé X, Z)

7.4.1.5 Transformace kovarianční matice

Uvažujme vektor náhodných veličin \mathbf{x} , ve kterém mají všechny jednotlivé prvky vektoru nulovou střední hodnotu. Pokud mají náhodné veličiny normální rozdělení, zapisujeme to takto: $\mathbf{x} \sim N(\mathbf{0}, \mathbf{Q}_x)$, kde \mathbf{Q}_x je kovarianční matice.

Otázkou teď je, jak se změní kovarianční matice, pokud provedem s veličinou \mathbf{x} lineární transformaci:

$$\mathbf{y} = \mathbf{Ax}. \quad (7.28)$$

Kovarianční matici nové veličiny \mathbf{y} vypočteme podle (7.26) takto:

$$\mathbf{Q}_y = E[\mathbf{y}\mathbf{y}^T]. \quad (7.29)$$

Do rovnice dosadíme z (7.28) a po úpravě⁶¹ dostaneme:

$$\mathbf{Q}_y = E[\mathbf{A}\mathbf{x}\mathbf{x}^T\mathbf{A}^T] = \mathbf{A}E[\mathbf{x}\mathbf{x}^T]\mathbf{A}^T = \mathbf{A}\mathbf{Q}_x\mathbf{A}^T. \quad (7.30)$$

⁶⁰ MATLAB File Exchange: <https://www.mathworks.com/matlabcentral/fileexchange/4705-error-ellipse>

⁶¹ Použijeme pravidlo $(\mathbf{Ax})^T = \mathbf{x}^T\mathbf{A}^T$.

7.4.2 Příklad: Detailní rozbor a odvození KF pro 1d systém

Smyslem tohoto příkladu je ukázat detailně princip KF a odvození Kalmanova zesílení na nejjednodušším příkladu, kdy stavový vektor má dimenzi 1. Veškeré matice v KF jsou pak nahrazeny skalárními hodnotami, což umožňuje podstatně jednodušší pochopení. Zároveň můžeme podpořit chápání maticových operací budováním užitečných analogií se skalárními výpočty.

[Ex.: KA02]

7.4.2.1 Definice problému

Uvažujme těleso, které se pohybuje translačně a působí na něj viskózní tření a náhodná (neznámá) síla. Jako stavovou proměnnou zvolíme rychlosť v a diferenciální rovnici pak můžeme zapsat ve tvaru:

$$m\dot{v} = -bv + F, \quad (7.31)$$

kde m, b jsou známé konstanty, F je neznámá síla. Naším úkolem bude *co nejpřesněji odhadovat rychlosť*, přičemž její měření je zatíženo šumem.

Po diskretizaci (TODO LINK) a doplnění rovnice měření dostáváme následující diskrétní lineární model systému (místo označení v budeme dále používat standardní označení stavu x):

$$x_k = ax_{k-1} + w \quad (7.32)$$

$$y_k = x_k + v, \quad (7.33)$$

kde a je známá konstanta a w a v jsou náhodné veličiny (šumy) s normálním (Gaussovým) rozdělením a odpovídajícími rozptyly σ_w a σ_v . Hodnoty rozptylů v praxi přesně neznáme.

Cílem této sekce je:

- znovu detailně rozvést postup výpočtu KF,
- odvodit optimální Kalmanovo zesílení K a
- diskutovat o volbě parametrů filtru Q a R .

7.4.2.2 KF

KF pracuje ve dvou krocích: nejprve predikuje hodnotu stavu na základě modelu a následně ji koriguje podle měření.

■ **Predikce** Při predikci využijeme model (7.32) pro výpočet odhadu „kde by systém v dalším kroku měl být“. Náhodný šum w zde nezahrnujeme, protože ho neznáme a jeho nejlepší odhad ($E[w]$) je 0.

$$\hat{x}_{k|k-1} = a\hat{x}_{k-1|k-1} \quad (7.34)$$

Dále aktualizujeme hodnotu kovariance P ⁶². Z rovnice (7.30) víme, co se stane s kovarianční maticí pokud náhodný vektor projde transformací. Tuto myšlenku můžeme v jednorozměrném příkladu snadno jako $P_{k|k-1} = aP_{k-1|k-1}a = a^2P_{k-1|k-1}$. Také si musíme uvědomit, že jsme v rovnici (7.34) udělali další krok a v reálném systému (7.32) došlo k přičtení náhodné síly w – *musíme tedy zvýšit nevěrohodnost (v tomto 1d příkladu rozptyl) odhadu P přičtením hodnoty Q .*

⁶² V 1d příkladu má P význam rozptylu.

$$P_{k|k-1} = a^2 P_{k-1|k-1} + Q. \quad (7.35)$$

- **Korekce** Ve fázi predikce jsme odhadovali, kde by mohl stav systému být na základě znalosti modelu. Nyní ho upravíme na základě měření. Přitom víme, že i měření je zatíženo šumem v (rovnice (7.33)), výsledný odhad tedy bude nějakým kompromisem mezi predikcí z modelu a naměřenou hodnotou.

Nejprve vypočteme odhad měření z modelu, v tomto případě je přímo roven stavu (hodnota $C = 1$):

$$\hat{y}_{k|k} = \hat{x}_{k|k-1}. \quad (7.36)$$

Následně upravíme hodnotu odhadu stavu takto:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K(y_k - \hat{y}_{k|k-1}). \quad (7.37)$$

Důležitou otázkou je, jak určíme hodnotu Kalmanova zesílení K – nejprve uvedeme výsledek a v dalším textu ho odvodíme:

$$K = \frac{P_{k|k-1}}{P_{k|k-1} + R} \quad (7.38)$$

K dokončení fáze korekce nám už zbývá pouze upravit kovarianci:

$$P_{k|k} = (1 - K)P_{k|k-1}. \quad (7.39)$$

7.4.2.3 Simulace

Chování KF nyní odsimuluujeme v MATLABu. Výsledek simulace pro následující skript je na obr. 7.2.

```
clc,clear
n = 30;
a = 0.9;
sigma_F = 0.15;
sigma_M = 0.6;
x(1) = 5;
y(1) = x(1);
% nastavení KF
Q = sigma_F^2 % optimální volba
R = sigma_M^2 % optimální volba
x_(1) = 0;
P(1) = 0;

for k = 2:n
    %% Simulace modelu
    x(k) = a * x(k-1) + sigma_F * randn;
    y(k) = x(k) + sigma_M * randn;
    %% KF
    % Predikce
    x_(k) = a*x_(k-1) ;
```

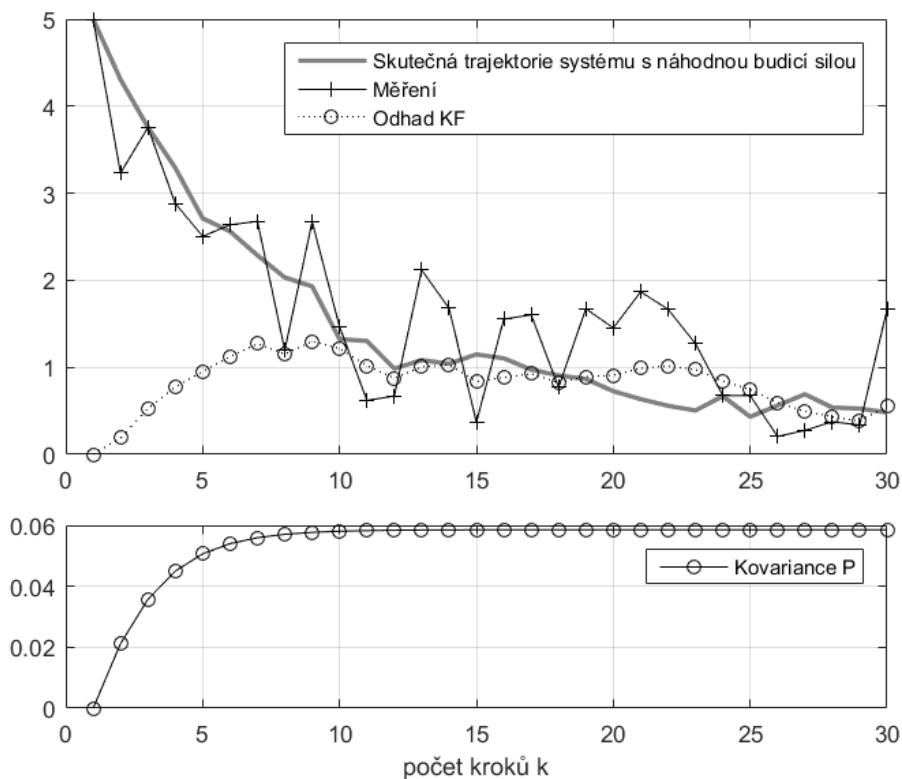
```

P(k) = a^2*P(k-1)+Q;
% Korekce
K = P(k) / (P(k)+R);
x_(k) = x_(k) + K*(y(k) - x_(k));
P(k) = (1-K)*P(k);
end

```

Z výsledků simulace můžeme udělat několik závěrů:

- KF funguje, filtruje naměřený signál, přitom odhadovaná hodnota je „někde mezi“ reálnou a naměřenou hodnotou.
- Hodnota P se po nějaké době (asi 10 kroků) ustálí na konstantní hodnotě.
- Z rovnice (7.38) vyplývá, že pokud je konstantní P , bude konstantní i K .



Obrázek 7.2 Výsledek simulace pro 1d příklad a optimálně nastavené kovariance Q a R .

7.4.2.4 Odvození Kalmanova zesílení pro 1d případ

V rovnici (7.38) jsme uvedli výpočet K , nyní provedeme jeho odvození.

Problém je postaven tak, že:

- při známém modelu systému (7.32),
- modelu měření (7.33) a
- známých modelech šumu w a v (uvažujeme normální rozdělení s nulovou střední hodnotou a známým rozptylem σ_w resp. σ_v)

máme za úkol optimálně odhadnout stav \hat{x}_k . Optimalita je definována jako minimalizace rozptylu chyby odhadu \tilde{x} :

$$\tilde{x}_k = x_k - \hat{x}_{k|k}. \quad (7.40)$$

Z předchozích rovnic provedeme odvození takto:

$$\begin{aligned} \tilde{x}_k &= x_k - \hat{x}_{k|k} = \\ &= ax_{k-1} + w - (a\hat{x}_{k-1|k-1} + K(y_k - a\hat{x}_{k-1|k-1})) = \\ &= ax_{k-1} + w - (a\hat{x}_{k-1|k-1} + K(ax_k + w + v - a\hat{x}_{k-1|k-1})) = \\ &= \dots \\ &= a(1-K)(x_{k-1} - \hat{x}_{k-1|k-1}) + (1-K)w - Kv \\ \tilde{x}_k &= a(1-K)\tilde{x}_{k-1} + (1-K)w - Kv \end{aligned}$$

Pro lepší přehlednost zápisu zavedeme ještě substituci:

$$\tilde{x}_k = \alpha\tilde{x}_{k-1} + \beta w - \gamma v. \quad (7.41)$$

Z odvozeného výrazu vyjádříme rozptyl:

$$\begin{aligned} E[\tilde{x}_k \tilde{x}_k] &= E[\alpha^2 \tilde{x}_{k-1}^2 + \alpha\beta\tilde{x}_{k-1}w - \alpha\gamma\tilde{x}_{k-1}v + \\ &\quad + \alpha\beta\tilde{x}_{k-1}w + \beta^2w^2 - \beta\gamma wv - \\ &\quad - \alpha\gamma\tilde{x}_{k-1}v - \beta\gamma wv + \gamma^2v^2]. \end{aligned}$$

Vzhledem k vlastnostem střední hodnoty můžeme uvažovat každý výraz zvlášť. Dále, pokud předpokládáme, že veličiny w , v a \tilde{x}_{k-1} jsou nezávislé (nekorelované), pak střední hodnota jejich součinu (kovariance) bude nulová. Dále si uvědomíme jak jsou definovány kovariance (zde rozptyly) P , Q a R a můžeme pak psát:

$$\begin{aligned} E[\tilde{x}_k \tilde{x}_k] &= E[\alpha^2 \tilde{x}_{k-1}^2 + \beta^2w^2 + \gamma^2v^2] = \\ &= \alpha^2 E[\tilde{x}_{k-1}^2] + \beta^2 E[w^2] + \gamma^2 E[v^2] = \\ &= \alpha^2(1-K)^2 P + (1-K)^2 Q + K^2 R = \\ &= (1-K)^2(a^2P + Q) + K^2R. \end{aligned}$$

Tuto hodnotu chceme minimalizovat, tedy:

$$\frac{\partial}{\partial K}(E[\tilde{x}_k \tilde{x}_k]) = -2(1-K)(a^2P + Q) + 2KR = 0$$

Po úpravě dostáváme výsledný vzorec:

$$K = \frac{a^2P + Q}{a^2P + Q + R}. \quad (7.42)$$

7.4.2.5 Konstantní hodnota P a K

Z výsledků simulace vidíme, že po určité době se hodnota P a tedy i K ustálí na konstantní hodnotě. Pro aplikaci KF (pro lineární systém) tedy nemusíme provádět predikci a korekci P a celý výpočet se značně zjednoduší:

■ Predikce

$$\hat{x}_{k|k-1} = a\hat{x}_{k-1} \quad (7.43)$$

■ Korekce

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K(y_k - \hat{x}_{k|k-1}). \quad (7.44)$$

Přitom hodnota K je konstantní a vypočteme ji podle rov. (7.42).

Ustálenou hodnotu P můžeme dopředu spočítat jednoduše z rovnosti $P_k = P_{k-1}$ a po vyjádření z rovnic (7.35), (7.38) a (7.39). Po úpravě dostaneme:

$$P_{k|k} = \frac{Ra^2 - Q - R + \sqrt{(Q + R - Ra^2)^2 + 4a^2RQ}}{2a^2}. \quad (7.45)$$

7.4.2.6 Diskuse k volbě parametrů filtru \mathbf{Q} a \mathbf{R}

- Hodnota Q je mírou toho, jak moc důvěrujeme modelu systému. Čím menší je Q , tím více důvěrujeme modelu. Pokud je $Q = 0$, pak důvěrujeme plně modelu a nebereme do úvahy měření.
- Hodnota R definuje obdobnou důvěru v měření. Pokud bude $R = 0$, tak plně důvěrujeme měření a $\hat{x} = y$.
- Hodnota odhadu \hat{x} se tedy pohybuje mezi dvěma krajnímimezemi – plná důvěra modelu a plná důvěra měření. Při nastavování parametrů KF tedy volíme nějaký kompromis mezi důvěrou v modelu a v měření.
- Lze ukázat, že podstatný je poměr Q/R , absolutní velikost obou parametrů není důležitá.
- V praxi žádný model neznáme zcela přesně (struktura rovnic) ani neznáme zcela přesně jeho parametry. Q tedy může zahrnovat kromě náhodného působení na model (zde je to náhodná síla) také naši nedůvěru v model samotný (jeho strukturu a parametry).
- Nejmenší odchylky odhadu od skutečného stavu systému dosáhneme tehdy, pokud je Q a R nastaveno v souladu se skutečným šumem modelu a měření. V tomto příkladu to je pokud $Q = \sigma_F^2$ a $R = \sigma_M^2$. Simulací lze snadno ukázat, že pokud zvolíme Q odlišně, výsledná chyba (rozptyl) odhadu od skutečnosti bude větší.
- V praxi samozřejmě musíme nastavení odhadovat a nikdy se také nedozvíme jak je náš odhad správný.

7.4.3 Příklad: KF pro dvourozměrný systém a jeho geometrická interpretace

[Ex.: KA03]

Smyslem tohoto příkladu je ukázat na co nejjednodušším systému *geometrický význam* kovarianční matice a celého algoritmu KF.

Budeme sledovat pohyb „podivného mobilního robotu“, který:

- se pohybuje v rovině $\mathbf{x} = [x_1, x_2]^T$,
- má vlastní dynamiku, která ho „přitahuje“ k bodu $[0; 0]$,
- působí na něj známý vstup $\mathbf{u} = [u_1; u_2]^T$, který ho posouvá nezávisle v souřadnicích x_1, x_2 a
- také náhodný neznámý vstup $\mathbf{w} = [w_1; w_2]^T$.

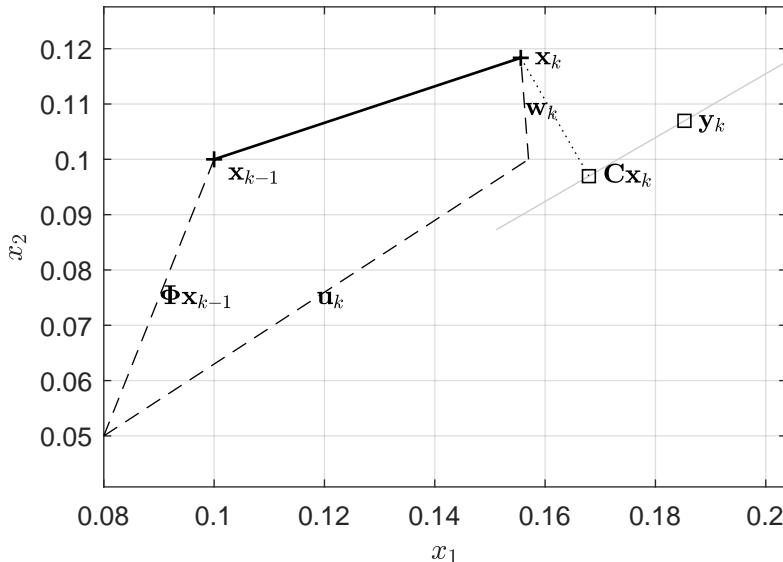
Poněkud neobvyklé je, že polohu robotu měříme pouze jedním sensorem, jehož osa je pootočena o úhel $\alpha = \frac{\pi}{6}$ vůči ose x_1 .

Odpovídající zápis (podle (7.1)-(7.2)) vypadá takto:

$$\mathbf{x}_k = \begin{bmatrix} 0.8 & 0 \\ 0 & 0.5 \end{bmatrix} \mathbf{x}_{k-1} + \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \quad (7.46)$$

$$y_k = \left[\cos \frac{\pi}{6} \quad \sin \frac{\pi}{6} \right] \mathbf{x}_k + v_k \quad (7.47)$$

Z rovnice tedy vidíme, že vstupy u_1 a u_2 a stejně tak náhodný šum w_1 a w_2 se přímo přičítají k poloze v odpovídající souřadnici – tedy posouvají polohu robotu.

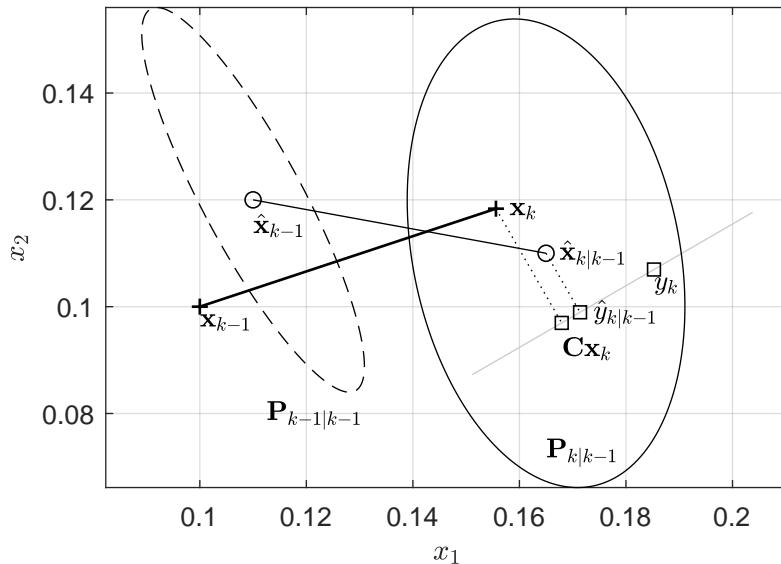


Obrázek 7.3 Detailní pohled na jeden krok simulace a průměr stavu do osy měření

Nyní rozebereme detailně jeden krok simulace a výpočtu KF. Na obr. 7.3 vidíme geometrický význam jednotlivých členů rovnice (7.1), tj. posun robotu vlivem vlastní dynamiky $\Phi\mathbf{x}$, vlivem vstupu \mathbf{u} a šumu \mathbf{w} . Robot se tedy posune z bodu \mathbf{x}_{k-1} do bodu \mathbf{x}_k . Pokud by sensor nebyl zatížen šumem, změřili bychom hodnotu \mathbf{Cx}_k , místo toho ale změříme y_k . Je dobré připomenout, že v tomto obrázku známe pouze \mathbf{u} a y_k , ostatní proměnné jsou neznámé.

Dále se podíváme na fázi predikce na obr. 7.4. Hodnotu $\hat{\mathbf{x}}_{k-1}$ známe z předchozího kroku KF. Podle rov. (7.3) vypočteme $\hat{\mathbf{x}}_{k|k-1}$, je přitom zřejmé, že predikce dokáže zohlednit dynamiku systému $\Phi \mathbf{x}_{k-1}$ i vstup \mathbf{u} , ale nezná šum \mathbf{w} .

Podobně podle rov. (7.4) vypočteme predikci kovarianční matici. Vidíme, že se kovarianční matice zvětšila – zvýšila se tedy nepřesnost odhadu polohy $\hat{\mathbf{x}}$. Na obr. 7.4 také vidíme průměr odhadu $\hat{\mathbf{x}}_{k|k-1}$ do souřadnicového systému měření. Teoreticky měřená hodnota $\mathbf{C}\mathbf{x}_k$, skutečně měřená y_k a odhadovaná z predikce $\hat{y}_{k|k-1}$ jsou tedy různé.



Obrázek 7.4 Predikce stavu a kovariance

Druhým krokem KF je korekce. Na obr. 7.5 vidíme korigovanou polohu $\hat{\mathbf{x}}_{k|k}$ i kovariaci $\mathbf{P}_{k|k}$. Vidíme, že kovarianční matice se zploštila ve směru osy měření (tj., měření nám dodalo informaci, proto můžeme snížit nejistotu odhadu), ale zůstala „dlouhá“ ve směru kolém na měření – protože v tomto směru jsme se z měření nic nového dozvědět nemohli. Vidíme také, že odhad polohy $\hat{\mathbf{x}}_{k|k}$ se posunul směrem doprava na nahoru. Z rov. (7.9) vidíme, že tato korekce je dána výrazem

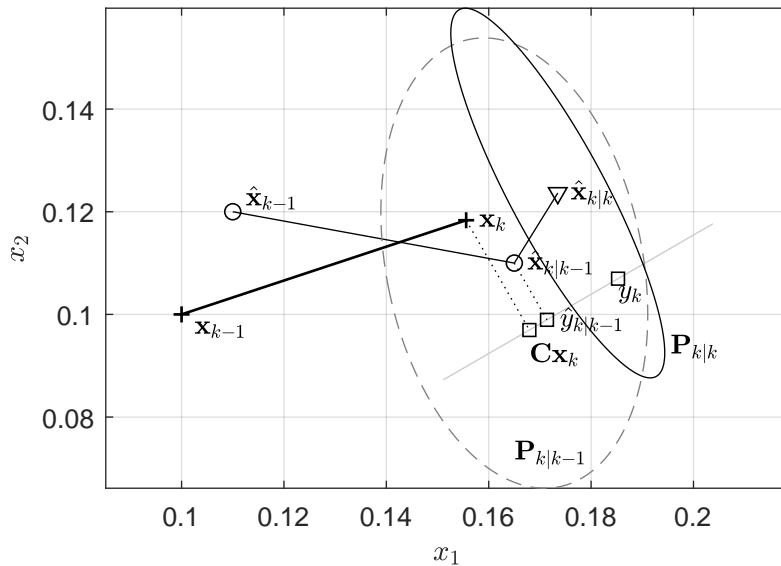
$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k (y_k - \hat{y}_{k|k-1}) \quad (7.48)$$

Když uvážíme dimenze jednotlivých členů v rov. (7.7) a (7.8), můžeme Kalmanovo zesílení zapsat takto:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{C}^T \frac{1}{S} \quad (7.49)$$

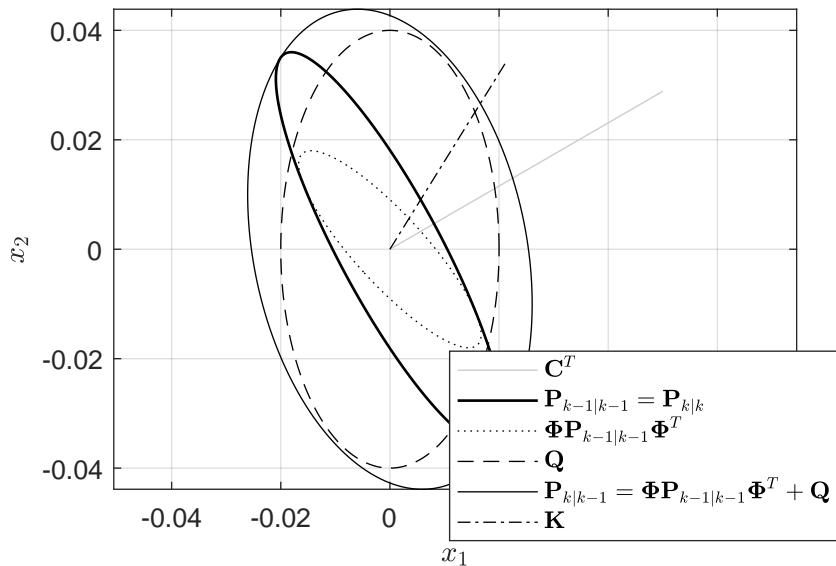
Směr korekce je v tomto příkladu tedy dán součinem $\mathbf{P}_{k|k-1} \mathbf{C}^T$, přičemž víme, že \mathbf{C}^T je směr osy měření.

Pokud bychom neměli k dispozici $\mathbf{P}_{k|k-1}$, bylo by nejlogičtějším směrem korekce \mathbf{C}^T – posunuli bychom se ve směru osy měření, protože žádnou jinou informaci nám měření nemůže poskytnout. V případě KF ale dodatečnou informaci v podobě $\mathbf{P}_{k|k-1}$ máme a směr \mathbf{C}^T tedy touto kovariancí modifikujeme (rov. (7.49)).



Obrázek 7.5 Korekce stavu a kovariance

Tvary kovariančních matic můžeme dobře porovnat na obr. 7.6.



Obrázek 7.6 Porovnání kovariančních matic v jednotlivých krocích KF a označení směru korekce Kalmanovým zesílením \mathbf{K}

Otázky, úvahy a úkoly

- Zkuste odsimulovat a promyslet jak se změní úloha pokud „robot nebude přitahován k nule“, tedy: $\Phi = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. Jak se bude vyvíjet kovariance $\mathbf{P}_{k|k}$ a proč?

7.4.4 Příklad: Anténa 1dof (potenciometr a gyro)

[Ex.: KA04]

Při digitální komunikaci pozemní stanice se satelitem musí anténa velmi přesně sledovat polohu satelitu. V tomto příkladu situaci zjednodušíme a budeme uvažovat pouze otáčení tělesa antény kolem jedné osy. Jako sensory použijeme potenciometr (úhel + gausovský šum) a MEMS gyro (úhlová rychlosť + gausovský šum). Ukážeme, že KF může snížit rozptyl odhadu úhlu a polohy.

Uvažujme rotační pohyb tělesa s viskózním třením, známým řídicím vstupem u (moment) a neznámou poruchou w (šum):

$$I\ddot{\varphi} = -b\dot{\varphi} + u + w \quad (7.50)$$

Odpovídající stavový popis i se zahrnutím šumu:

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{b}{I} \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ \frac{1}{I} \end{bmatrix} u + \begin{bmatrix} 0 \\ \frac{1}{I} \end{bmatrix} w \quad (7.51)$$

kde $\mathbf{x} = [\varphi, \dot{\varphi}]^T$. Rovnice měření:

$$\mathbf{y} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{x} + \mathbf{v}, \quad (7.52)$$

kde $\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$ je vektor dvou vzájemně nezávislých náhodných veličin.

Provedeme diskretizaci způsobem popsáným v kap. 3.3 a získáme:

$$\mathbf{x}_k = \Phi \mathbf{x}_{k-1} + \Gamma u + \Gamma w \quad (7.53)$$

$$\mathbf{y}_k = \mathbf{C} \mathbf{x}_k + \mathbf{v}_k \quad (7.54)$$

Jeden krok simulace diskrétního dynamického systému bude vypadat takto:

```
w(i) = sigma_sys*randn; % náhodná (neznámá) síla
x(:,i) = Phi*x(:,i-1) + Gamma*u(i) + Gamma*w(i); % systém
y(:,i) = C*x(:,i) + [sigma_pot*randn ; sigma_gyr*randn] ; % měření
```

Pro simulaci tedy potřebujeme definovat tři hodnoty rozptylu: rozptyl náhodné síly σ_s a rozptyly potenciometru σ_p a gyro σ_g .

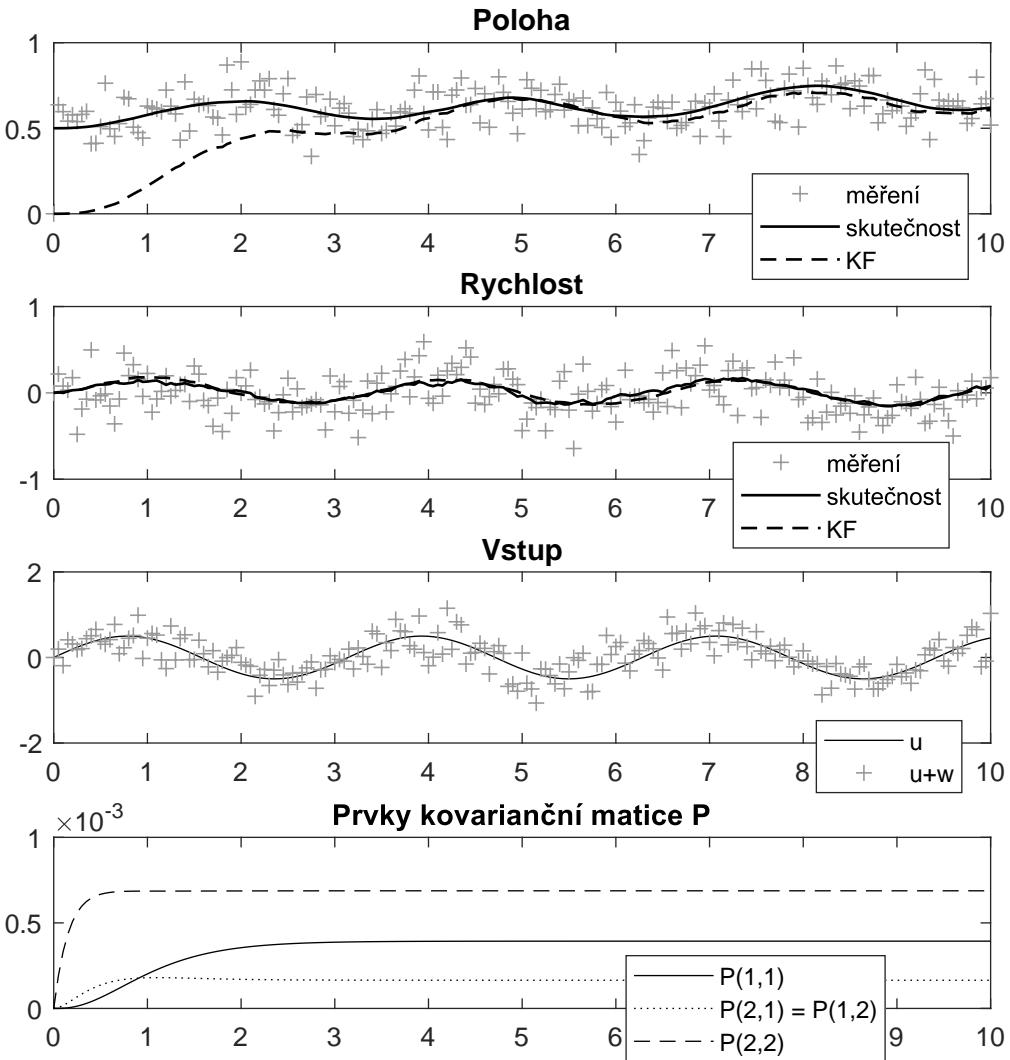
Jeden krok KF:

```
% Predikce
x_pred(:,i) = Phi*x_est(:,i-1) + Gamma*u(i);
P_pred = Phi*P*Phi' + Q;
y_pred = C*x_pred(:,i);
% Korekce
dy = y(:,i) - y_pred;
S = C*P_pred*C' + R;
K = P_pred*C'*inv(S);
x_est(:,i) = x_pred(:,i) + K*dy;
P = P_pred - K*C*P_pred;
```

Pro KF musíme definovat matice \mathbf{Q} a \mathbf{R} . Optimální nastavení těchto matic je ve shodě se simulací systému, tedy:

$$\mathbf{Q} = \sigma_s^2 \mathbf{\Gamma} \mathbf{\Gamma}^T \quad (7.55)$$

$$\mathbf{R} = \begin{bmatrix} \sigma_p^2 & 0 \\ 0 & \sigma_g^2 \end{bmatrix} \quad (7.56)$$

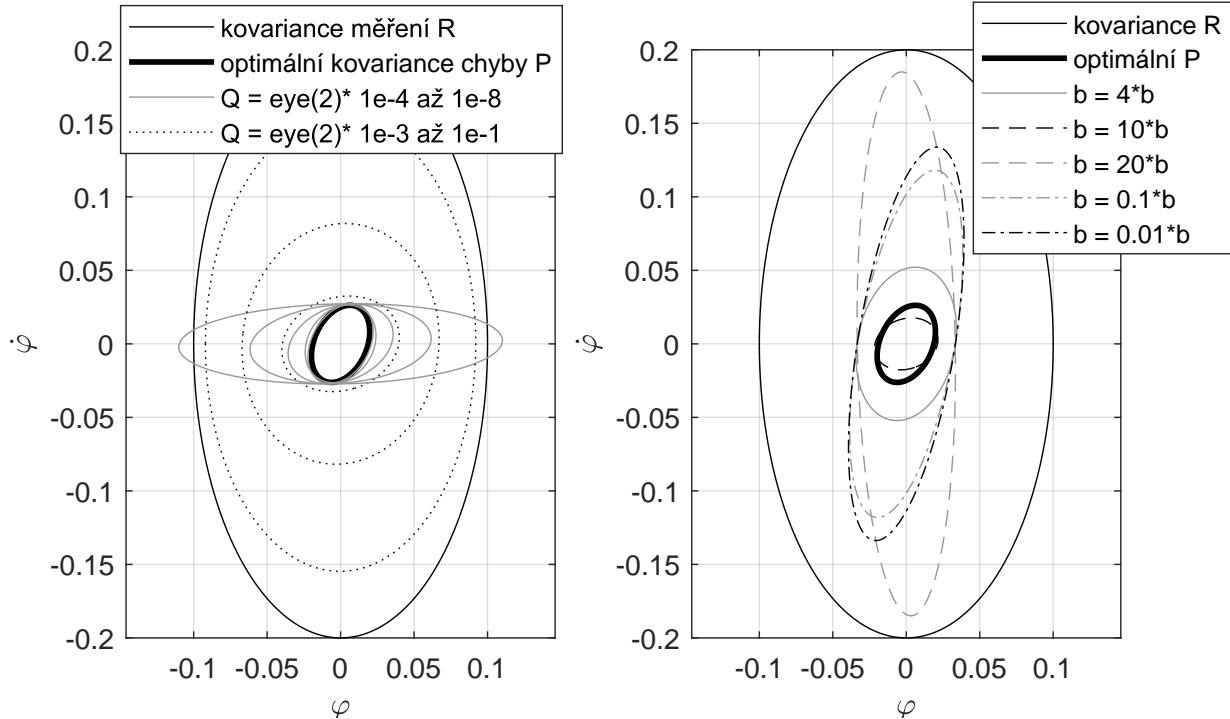


Obrázek 7.7 Výsledek simulace pro parametry: $b = 3$, $I = 1$, $T_s = 0.05$, $\sigma_s = 0.3$, $\sigma_P = 0.1$ a $\sigma_g = 0.2$

Na obr. 7.7 vidíme výsledek simulace pro zvolené parametry a hodnoty šumů. Po cca 5 sek. se odhad polohy přiblíží skutečnosti a pak už ji sleduje velmi dobře. Dojde také k ustálení hodnot matice \mathbf{P} .

Výsledek je také dobře znázorněn na obr. 7.8. Především zde porovnáváme kovariaci měření \mathbf{R} (takto nepřesnou znalost máme pouze s použitím sensorů bez filtrace)

s výslednou kovariancí chyby odhadu. Dále je zde znázorněno jak se výsledná kovariance změní při použití diagonální matice \mathbf{Q} (vlevo) nebo při nepřesném odhadu parametru b systému (vpravo).



Obrázek 7.8 Vizualizace kovariančních matic

Otázky, úvahy a úkoly

- Vysvětlete proč má \mathbf{Q} tvar podle rov. (7.55)!
- Zakreslete vedle rovnic (7.53) a (7.54) dimenze jednotlivých proměnných!

7.4.4.1 Offset sensoru

[Ex.: KAO6] MEMS gyroskop je typický sensor, u kterého se vždy musíme vypořádat s offsetem (posunutí nulové hodnoty měřené úhlové rychlosti). Základní kalibraci je možné u sensoru provést v klidu, hodnota offsetu se ale v čase mění (např. vlivem teploty) a proto je u aplikací, kde vyžadujeme vysokou přesnost, nutné offset odhadovat i během provozu zařízení (online).

Offset budeme považovat za další (třetí) stavovou proměnnou x_3 a měření na gyroskopu tedy bude mít tvar $y_G = x_3 + x_2$. Celou rovnici měření pak zapíšeme takto:

$$\mathbf{y} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \mathbf{x} + \mathbf{v} \quad (7.57)$$

Při odhadu parametrů pomocí KF uvažujeme, že se jejich hodnota v čase nemění a dynamiku proto modelujeme jednoduše takto:

$$\dot{x}_3 = 0. \quad (7.58)$$

Stavový popis (7.51) tedy rozšíříme do tvaru:

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -\frac{b}{I} & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ \frac{1}{I} \\ 0 \end{bmatrix} u + \begin{bmatrix} 0 \\ \frac{1}{I} \\ 0 \end{bmatrix} w \quad (7.59)$$

7.5 Otázky a neřešené úlohy

Otázky

1. Uvažujme normální rozdělení pravděpodobnosti. Kolik procent hodnot se nachází v intervalu $\pm\sigma$ a $\pm 2\sigma$ (přibližně)?
2. Jaká je souvislost rozptylu a směrodatné odchylky?
3. Nakreslete elipsu reprezentující vektor dvou navzájem nezávislých náhodných veličin s normálním rozdělením a hodnotami směrodatných odchylek $\sigma_1 = 0.5$ a $\sigma_2 = 0.2$ (elipsu kreslete pro hodnotu 1σ).

Úlohy

4. Uvažujte dynamický systém

$$m\ddot{x} = -b\dot{x} - kx + u + w,$$

kde u je známá vstupní síla, w je neznámá náhodná síla a hodnoty parametrů jsou: $m = 5$, $b = 0.3$ a $k = 10$.

- a) Přepište tuto ODE do tvaru lineárního stavového modelu se spojitým časem.
- b) Zapište rovnici měření, kdy budeme měřit pouze polohu x .
- c) Proveďte diskretizaci (bez využití příkazu c2d).
- d) Proveďte simulaci odezvy na jednotkový skok na vstupu u pomocí příkazu lsim.
- e) Proveďte simulaci odezvy pomocí diskretizovaného modelu a porovnejte oba výsledky.
- f) Uvažujme náhodnou sílu w (normální rozdělení) se směrodatnou odchylkou $\sigma_s = 0.3$ a šum na sensoru $\sigma_p = 0.8$. Definujte optimální matice \mathbf{Q} a \mathbf{R} pro KF. Zapište algoritmus KF a vykreslete do grafu skutečnou, měřenou a odhadovanou polohu.
- g) Z chyby odhadu (rozdíl mezi skutečnou a odhadovanou polohou a rychlostí) vypočtěte kovarianční matici. Porovnejte ji s ustálenou hodnotou matice \mathbf{P} z výpočtu KF.

8 Identifikace systémů

Obsah kapitoly:

8.1 Statické modely	109
8.2 Otázky a neřešené úlohy	109

8.1 Statické modely

Statický model je obecně nelineární statická závislost r výstupů na p vstupech (MIMO). Každý takový MIMO model lze převést (rozdělit) na r MISO modelů (r vstupů a jeden výstup).

MISO model pak lze matematicky reprezentovat:

- lineární závislostí
- polynomem
- tabulkou
- neuronovou sítí (vícevrstvá dopředná síť, RBF,...)
- fuzzy modelem
- lokálním lineárním modelem (LOLIMOT, LWR, LWPR,...)

Při vyhodnocování použitelnosti a vhodnosti reprezentace MISO modelu je vhodné vzít v úvahu tato kritéria:

- Je model lineární v parametrech?
- Je bázová(é) funkce globální nebo lokální?
- Jaké jsou interpolační a extrapolaci vlastnosti modelu?
- Je dána struktura modelu nebo budeme provádět její optimalizaci?
- Jak je daný model náročný na zdroje? (data v paměti, rychlosť optimalizace struktury a parametrů modelu, rychlosť výpočtu odesvy modelu)
- Můžeme vytvořený model s nalezenými parametry vhodně vizualizovat?
- Můžeme strukturu a parametry modelu interpretovat/vysvětlit na základě fyzikální podstaty modelovaného problému?
- Je možné měnit strukturu/parametry modelu online?

8.2 Otázky a neřešené úlohy

Otázky

1. Definujte *statický* a *dynamický* systém!
2. Vysvětlete zkraty SISO, MISO, MIMO v kontextu statických modelů. Vysvětlete dekompozici MIMO systému na několik MISO modelů!
3. Vysvětlete pojem *bázová funkce* v kontextu statických MISO modelů!
4. U následujících typů modelů uveďte zda mají lokální nebo globální bázovou funkcí:
 - a) 1D tabulka (1 vstup, 1 výstup)
 - b) 2D tabulka (2 vstupy, 1 výstup)
 - c) polynom $y = \theta_1 x^2 + \theta_2 x + \theta_3$
 - d) RBF síť

5. Jaké jsou zásadní praktické výhody použití tabulky (LUT = look-up table) pro aproximaci MISO statické závislosti?
6. U následujících statických modelů uvedte zda jsou lineární či nelineární v parametrech θ_i :
 - a) $y = \theta_1 x + \theta_2$
 - b) $y = \theta_1 x^2 + \theta_2 x + \theta_3$
 - c) $y = \theta_1 \frac{x}{\theta_2} + \frac{1}{\theta_3}$
7. Vysvětlete problém *vychýlení odhadu parametrů*!

Úlohy

8. Uvažujme dynamický systém v diskretizované podobě:

$$x_k = a_1 x_{k-1} + a_2 x_{k-2} + c u_{k-2}. \quad (8.1)$$

Pro tento systém provedte odhad parametrů pomocí OLS!

- a) Definujte skokový vstup $u = 1$.
- b) Proveďte simulaci diskrétního modelu s parametry $a_1 = \dots$, $a_2 = \dots$, a $c = \dots$. Prozatím neuvažujme šum. Takto vytvořená data budeme považovat za data měření.
- c) Naměřená data použijte pro formulaci problému OLS. Ukažte, že pomocí OLS vypočteme hodnoty a_1 , a_2 a c .
- d) Dále simulaci vylepšíme o šum na sensoru ($y = x + w$, kde w je náhodná veličina s normálním rozdělením a nulovou střední hodnotou). Uvažujte $\sigma = \dots$. Ukažte, že při použití OLS dojde k vychýlení odhadu.
- e) Výpočet opakujte s postupně vzrůstajícím σ a vykreslete závislost odhadu parametrů na σ .
9. Na předchozí úlohu se šumem aplikujte *metodu pomocné proměnné* (Instrument Variable method). Ukažte, že odhad konverguje ke správným parametrům.

9 Feedforward control aneb přímovazebné řízení

Obsah kapitoly:

9.1	Lyrický úvod	111
9.2	Schéma FFC a inverzní dynamika	112
9.2.1	Matematikovo kyvadlo	113
9.2.2	Plánovač trajektorie pro FFC	113
9.3	Příklad: Rychlostní řízení 1dof	114
9.3.1	Model systému	114
9.3.2	FFC	114
9.3.3	Aplikace	114
9.4	Příklad: Stabilizace antény 1dof	115
9.4.1	Model systému	115
9.4.2	Model poruchy	116
9.4.3	Čistý FFC	116
9.4.4	FFC + zpětná vazba	116
9.5	Příklad: Řízení mostového jeřábu	117
9.6	Otázky a neřešené úlohy	117

9.1 Lyrický úvod

Jedním z nejdůležitějších vynálezů, který umožnil rozvoj naší civilizace, je *zpětná vazba* (angl. *feedback*). Je základním konceptem v teorii (i praxi) řízení systémů, známým příkladem je Wattův odstředivý regulátor, který významně urychlil vývoj a možnost nasazení parních strojů. Zpětnovazební řízení funguje tak, že měříme aktuální hodnotu řízené veličiny, porovnáme ji se žádanou hodnotou a podle jejich rozdílu vykonáme akci. Takto funguje nejen Wattův regulátor, ale i např. obyčejná lednička.

Přímovazební řízení (angl. *feedforward control (FFC)*) působí na systém přímo bez znalosti jeho aktuálního stavu – bez zpětné vazby. Velikost akčního zásahu (vstupu do řízeného systému) se určí na základě žádané hodnoty, znalosti o systému (modelu) a případně měření poruchy působící na systém.

Nejjednodušším příkladem FFC může být řízení otáček DC motoru nějakého přístroje pomocí nastavení pevné hodnoty napětí. Při zátěži (kterou neznáme a neměříme) otáčky klesnou, ale u řady aplikací nám to nemusí vadit.

Příkladem FFC, které měří poruchu, z každodenní praxe může být *ekvitermní regulace*⁶³, která měří poruchovou veličinu (venkovní teplotu) a podle ní (se znalostí tepelných ztrát domu) nastavuje výkon otopné soustavy.

⁶³ Ekvitermní regulace je způsob řízení výkonu otopného systému v domě bez měření teploty uvnitř domu. Měří se pouze venkovní teplota a podle tzv. ekvitermních křivek se nastavuje výkon vytápění. Tyto ekvitermní křivky musí být zjištěny (dolaheny) individuálně pro každý dům a představují vlastně nelineární statický tepelný model stavby. Je zřejmé, že ekvitermní regulace neumí reagovat na jevy typu pootevřené okna nebo naopak instalaci hrnčířské pece v objektu. Při stabilních podmínkách ale funguje velmi dobře. Praxe zde jasně ukazuje, že princip FFC může být výhodnější než použití zpětné vazby. A mimochodem: Ekvitermní regulace by se vlastně vůbec neměla jmenovat regulace, protože tento pojed spojen se zpětnou vazbou.

FFC se v učebnicích, moudrých knihách a výuce na univerzitách skoro nevyskytuje, v praxi se ovšem používá hojně. Dá se říci, že s rozvojem mechatroniky, tj. mimo jiné s možností implementovat na řídicí mikropočítáč „jakýkoli“ i velmi složitý a nelineární model/algoritmus se oblast aplikace FFC podstatně rozšířila. Prof. Rolf Isermann použil slovo *feedforward* ve svém skvělém přehledovém článku *Mechatronic systems – Innovative products with embedded control* [[<ISERMANN200814>](#)] celkem 7x.

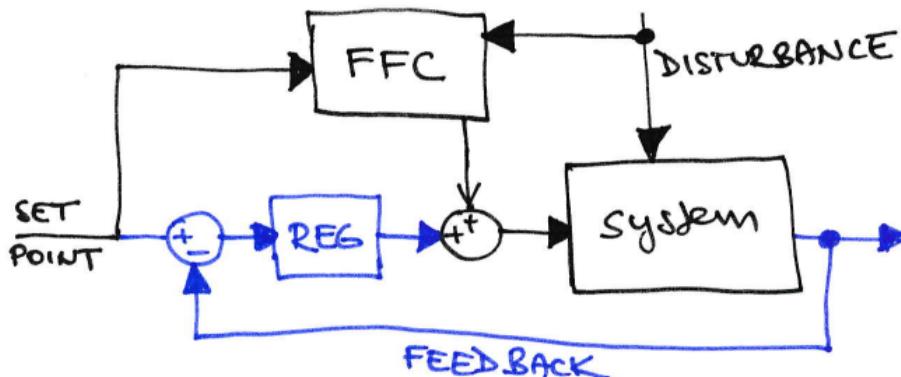
Motivací pro použití FFC může být:

- řízenou veličinu nelze snadno nebo levně měřit (příklad Mostový jeřáb)
- pomocí FFC můžeme částečně nebo úplně v systému vykompenzovat nelinearity, tj. systém linearizovat a dále ho snadněji řídit pomocí nástrojů lineárního řízení (příklad kompenzace suchého tření v pohonu)
- systém je dostatečně deterministický, tj.: známe jeho model a případně můžeme měřit poruchu vstupující do systému – pak je použití FFC často levnější a výhodnější, než realizace zpětnovazebního řízení.

O tom, že FFC funguje i při řízení lidského těla se můžeme přesvědčit, když uchopíme do ruky nějaký předmět, u kterého předpokládáme např. podstatně větší hmotnost („kámen z polystyrenu“). Co se stane? Ruka s „kamenem“ vystřelí nahoru. Proč? Protože model FFC při řízení pohybu ruky pracoval s nějakým předpokladem potřebné síly a touto *dopředu* působil. Po chvíli samozřejmě dojde (vlivem zpětné vazby) ke stabilizaci a podruhé už si dáme větší pozor (tj. dojde k online adaptaci dynamického modelu FFC).

9.2 Schéma FFC a inverzní dynamika

Na obrázku 9.1 je vidět uspořádání řízení pomocí FFC doplněného klasickou zpětnou vazbou. Je vidět, že vstupem do FFC může být žádaná hodnota a měřená porucha.



Obrázek 9.1 Schéma kombinace FFC a zpětné vazby

V předchozím textu jsme se zmínili, že FFC používá *znalost* modelu systému – matematicky jde o inverzní dynamický model. Pro systém druhého rádu (častý případ při řízení SISO systémů) vypadá tento model obecně takto:

$$f(\ddot{q}, \dot{q}, q) = \tau. \quad (9.1)$$

Pokud je systém lineární, pak např. takto:

$$I\ddot{q} + b\dot{q} + kq = \tau. \quad (9.2)$$

9.2.1 Matematikovo kyvadlo

Dobrým příkladem je nelineární matematické kyvadlo:

$$mL^2\ddot{q} + b\dot{q} + mgL \sin(q) = \tau. \quad (9.3)$$

Nejjednodušším případem FFC je kompenzace⁶⁴ nelineární tíhové složky:

$$\tau_{FFC} = mgL \sin(q_R), \quad (9.4)$$

kde q_R je žádaná hodnota. Pokud by řízení bylo realizováno pouze pomocí FFC, pak bude dynamika systému vypadat takto:

$$mL^2\ddot{q} + b\dot{q} + mgL \sin(q) = mgL \sin(q_R). \quad (9.5)$$

Po odeznění přechodového děje ($\ddot{q}, \dot{q} = 0$) se poloha ustálí na hodnotě $q = q_R$. Takto jsme tedy kompenzovali pouze statickou část systému. Zbytek „problému“ může být už vyřešen zpětnou vazbou, v okolí rovnovážné (=žádané) polohy pak pracujeme s (přibližně) linearizovaným systémem:

$$mL^2\ddot{q} + b\dot{q} = 0 + \tau, \quad (9.6)$$

kde τ je zpětnovazební vstup do systému.

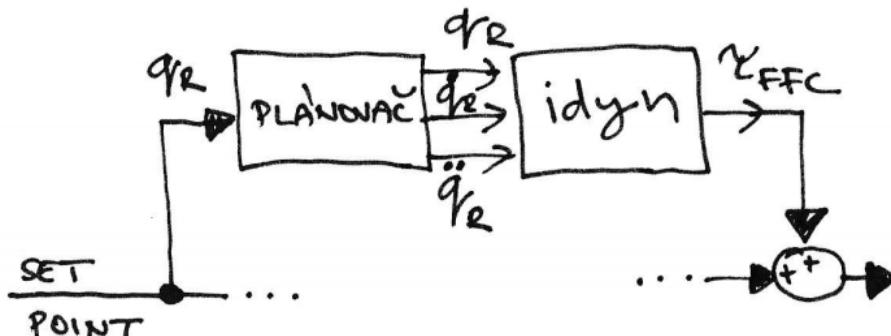
Jistě nás následně napadne, že bychom mohli kompenzovat nejen statickou část modelu $mgL \sin(q)$, ale třeba i viskózní tlumení $b\dot{q}$ a setrvačnost $I\ddot{q}$. τ_{FFC} by pak měla hodnotu

$$\tau_{FFC} = mgL \sin(q_R) + b\dot{q}_R + I\ddot{q}_R. \quad (9.7)$$

Vidíme ale, že pro výpočet FFC potřebujeme nyní nejen žádanou hodnotu q_R , ale také žádanou rychlosť a zrychlení – které typicky nejsou vstupem do řídicího systému. Nepoučeného čtenáře jistě napadne myšlenka použít numerickou derivaci, my ostatní ji ale zase raději rychle zapomeneme (***TODO*** odkaz na num. der.). Místo toho budeme muset použít nějaký vhodný *plánovač trajektorie*.

9.2.2 Plánovač trajektorie pro FFC

Na obr. 9.2 vidíme princip plánovače trajektorie pro systém druhého řádu. Pro inverzní dynamický model musíme mít k dispozici hodnoty $q_R(t), \dot{q}_R(t)$ a $\ddot{q}_R(t)$.



Obrázek 9.2 Schéma plánovače trajektorie pro systém druhého řádu

⁶⁴ Tato kompenzace se dá do přirozeného jazyka přeložit asi takto: „vím, že při úhlu q_R musím působit momentem τ_{FFC} abych vyrovnal tíhu břemene m na rameni L “.

Při znalosti vlastnostního systému druhého rádu (kap. 1.3) můžeme vytvořit jednoduchý plánovač aplikací přenosu 1.11 s parametry $\zeta = 1$, $k_{DC} = 1$.

9.3 Příklad: Rychlostní řízení 1dof

[Ex.: CT21]

9.3.1 Model systému

Uvažujme model pohonu ve tvaru

$$I\dot{\omega} + b\omega = \tau \quad (9.8)$$

kde I je moment setrvačnosti, b je viskózní tlumení a τ je vstupní moment.

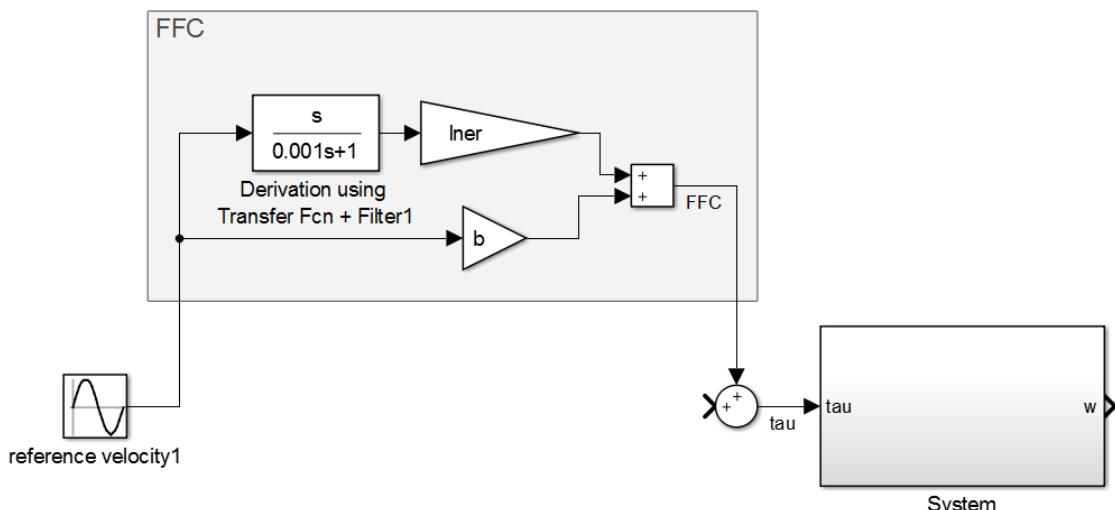
Cílem je řídit rychlosť pohonu podle referenčního průběhu $\omega_R(t)$.

9.3.2 FFC

Výpočet τ pro FFC je v tomto příkladu zcela zřejmý:

$$\tau(t) = I\dot{\omega}_R(t) + b\omega_R(t). \quad (9.9)$$

V praxi máme většinou k dispozici pouze referenční hodnotu ω_R a nikoli její derivaci. Referenční hodnotu tedy zderivujeme (vč. použití filtru).



Obrázek 9.3 FFC

9.3.3 Aplikace

Zatímco ve školách se o FFC moc nemluví, v praxi se používá. Dokladem toho je např. dokument k programovatelnému výkonovému prvku EPOS firmy MAXON [<maxonEPOS_FFC>]. Pomocí konfiguračního SW lze nastavit konstanty b a I pro FFC.

9.4 Příklad: Stabilizace antény 1dof

V tomto příkladu ukážeme možnost využití FFC při stabilizaci antény umístěné na pohybujícím se objektu - např. na lodi. Pro FFC využijeme možnosti měřit poruchu (rychlosť náklonu lodi).

Na obrázku 9.4 vidíme definici úhlů, cílem řízení je stabilizovat úhel φ na požadované konstantní hodnotě.



Obrázek 9.4 Schéma systému

9.4.1 Model systému

Základní kinematická vazba mezi úhly je tato:

$$\varphi = q + \delta, \quad (9.10)$$

což samozřejmě platí i pro derivace v čase:

$$\dot{\varphi} = \dot{q} + \dot{\delta} \quad (9.11)$$

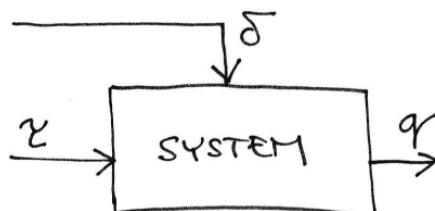
$$\ddot{\varphi} = \ddot{q} + \ddot{\delta} \quad (9.12)$$

Předpokládáme, že anténa je dokonale vyvážená (můžeme tedy zanedbat vliv tělové síly) a dynamiku soustavy popíšeme takto:

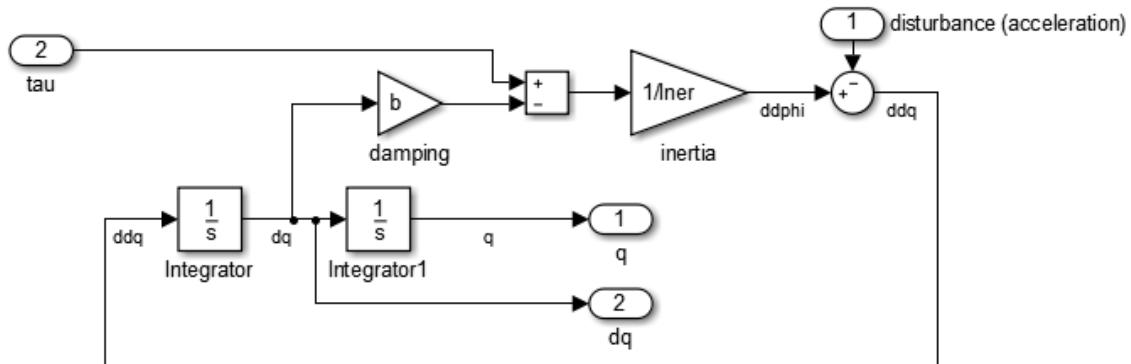
$$I\ddot{\varphi} + b\dot{\varphi} = \tau, \quad (9.13)$$

kde I je moment setrvačnosti stabilizovaného tělesa (antény), b je viskózní tlumení v rotační vazbě anténa-lodě a τ je vstupní moment (generovaný např. elektrickým servopohonem).

Z hlediska teorie řízení má systém 2 vstupy: 1 řízený τ a 1 (měřitelnou) poruchu δ . Odpovídající model v Simulinku je na obr. 9.6.



Obrázek 9.5 Schéma systému



Obrázek 9.6 Model systému v Simulinku

9.4.2 Model poruchy

Poruchu budeme modelovat jako sinusový signál, což ale nijak neubírá na obecnosti řešení. Budeme potřebovat δ i obě derivace:

$$\delta = a \sin(\omega_d t) \quad (9.14)$$

$$\dot{\delta} = \omega_d a \cos(\omega_d t) \quad (9.15)$$

$$\ddot{\delta} = -\omega_d^2 a \sin(\omega_d t) \quad (9.16)$$

9.4.3 Čistý FFC

Cílem stabilizace je konstantní hodnota φ , tedy musí platit: $\dot{\varphi} = 0$, $\ddot{\varphi} = 0$. Jednodušší analýzou rovnice 9.13 můžeme snadno navrhnout FFC:

$$I\ddot{\varphi} + b\dot{\varphi} = \tau, \quad (9.17)$$

$$I\ddot{\varphi} + b(\dot{\varphi} - \dot{\delta}) = \tau, \quad (9.18)$$

$$0 + b(0 - \dot{\delta}) = \tau, \quad (9.19)$$

Dostáváme tedy vypočtený moment ve tvaru:

$$\tau = -b\dot{\delta}. \quad (9.20)$$

Odvodili jsme tedy, že pokud budeme na systém působit momentem $\tau = -b\dot{\delta}$, bude platit $\varphi = \text{konst.}$ Simulační model je na obr. 9.7.

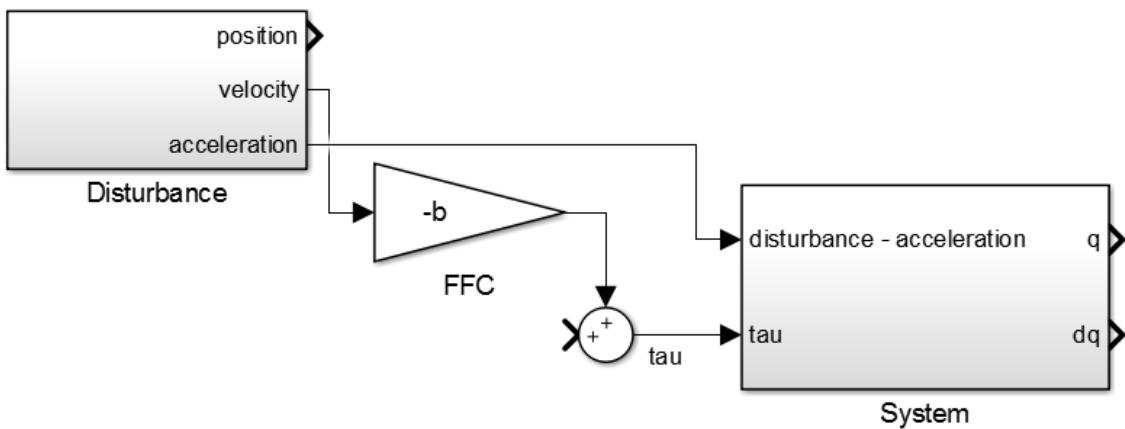
Použitelnost takto formulovaného FFC samozřejmě závisí na:

- možnosti měřit $\dot{\delta}$ (v tomto případě to lze díky gyrom) a na kvalitě měření (přesnost, šum, offset, drift),
- přesnosti odhadu b a
- na tom, zda systém neobsahuje další nemodelované jevy, např. suché tření.

V ideálním případě (v simulaci podle obr. 9.7) funguje FFC dokonale (tj. odchylka φ od nuly je na úrovni chyby numerického řešení).

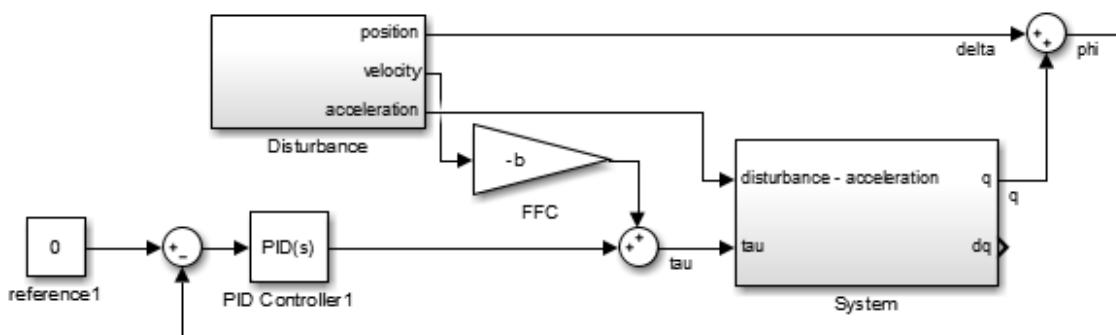
9.4.4 FFC + zpětná vazba

V reálném případě budeme patrně vždy chtít FFC doplnit i zpětnou vazbou, která bude kompenzovat nepřesnosti modelu. Konkrétní implementace závisí na tom, co



Obrázek 9.7 Čistý FFC

jsme schopni na reálném systému měřit. Na obr. 9.8 vidíme příklad, kdy měříme přímo φ , např. inklinometrem (průmět g do osy akcelerometru) nebo obecně IMU jednotkou s vlastním zpracováním signálu.



Obrázek 9.8 FFC + feedback

9.5 Příklad: Řízení mostového jeřábu

9.6 Otázky a neřešené úlohy

Otázky

- Zakreslete obecné schéma řízení systému pomocí kombinace zpětné vazby a FFC. Označte v obrázku:
 - řízený systém (system, plant)
 - vstup do systému (u)
 - výstup ze systému = měření (y)
 - referenční (žádanou) hodnotu
 - poruchu
 - zpětnovazební smyčku a regulátor
 - blok FFC
- Vysvětlete proč a kdy je při použití FFC potřeba plánovač trajektorie.

3. Vysvětlete jak funguje ekvitermní regulace používaná při vytápění staveb.
4. Vysvětlete jak můžeme realizovat plánovač trajektorie, zapište rovnice.

Úlohy

5. Zakreslete schéma FFC pro matematické kyvadlo podle obrázku [10.2](#). Uvažujte dvě varianty:
 - a) Kompenzujte pouze tíhovou sílu (moment).
 - b) Kompenzujte celou dynamiku systému.
Nezapomeňte na plánovač trajektorie, pokud je potřeba. Vytvořte model v Simulinku a ukažte jak (ne)funguje řízení pomocí čistého FFC (tj. bez zpětné vazby).
6. Vytvořte model s FFC řízením pro mechanický oscilátor se vstupem podle obr. [10.1](#). Kompenzujte pouze pružinu k a tlumení b . Použijte vhodný plánovač trajektorie, vytvořte model v Simulinku a ukažte jak (ne)funguje řízení pomocí čistého FFC (tj. bez zpětné vazby).

10 Přehled modelů systémů

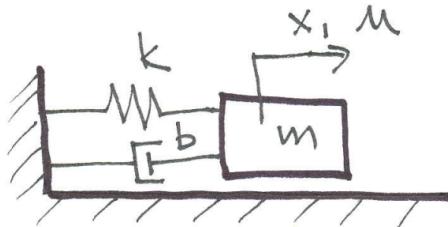
Obsah kapitoly:

10.1	Systém prvního řádu	119
10.2	Hmota s pružinou a tlumičem	119
10.3	Matematické kyvadlo	119
10.4	Mostový jeřáb	119
10.5	DC motor	120
10.5.1	Model s dynamikou elektrické části	120
10.5.2	Model se statickou elektr. částí	120
10.6	Dvě hmoty s pružinou a tlumičem (fixované k rámu)	121
10.7	Dvě hmoty s pružinou a tlumičem (volné)	121
10.8	Dvojité kyvadlo	121

10.1 Systém prvního řádu

$$\tau \dot{q} + q = ku \quad (10.1)$$

10.2 Hmota s pružinou a tlumičem



Obrázek 10.1 Schéma mechanického oscilátoru se vstupem

$$m\ddot{x} + b\dot{x} + kx = u \quad (10.2)$$

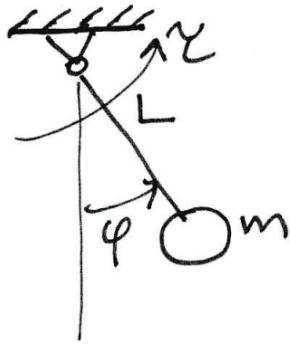
$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k/m & -b/m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t). \quad (10.3)$$

10.3 Matematické kyvadlo

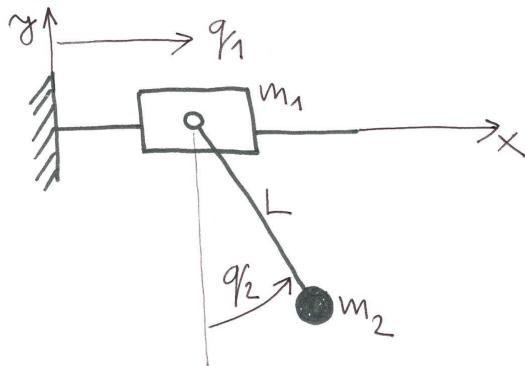
$$mL^2\ddot{\varphi} + b\dot{\varphi} + mgL \sin \varphi = \tau \quad (10.4)$$

10.4 Mostový jeřáb

$$\begin{bmatrix} m_1 + m_2 & m_2 L \cos(q_2) \\ m_2 L \cos(q_2) & m_2 L^2 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} + \begin{bmatrix} -m_2 L \dot{q}_2^2 \sin(q_2) \\ m_2 g L \sin(q_2) \end{bmatrix} = \begin{bmatrix} F_1 \\ 0 \end{bmatrix} \quad (10.5)$$



Obrázek 10.2
Matematické kyvadlo



Obrázek 10.3 Schéma
mostového jeřábu

10.5 DC motor

10.5.1 Model s dynamikou elektrické části

$$u = Ri + L \frac{di}{dt} + k_M \omega \quad (10.6)$$

$$J\dot{\omega} = k_M i - b\omega + \tau, \quad (10.7)$$

kde u je napájecí napětí, R odpor vinutí, i proud, L indukčnost, k_M konstanta, ω úhlová rychlosť, J moment setrvačnosti rotoru, b viskózní tlumení a τ vnější zátěžný moment.

$$\begin{bmatrix} \frac{di}{dt} \\ \frac{d\omega}{dt} \end{bmatrix} = \begin{bmatrix} -\frac{R}{L} & -\frac{k_M}{L} \\ \frac{k_M}{J} & -\frac{b}{J} \end{bmatrix} \begin{bmatrix} i \\ \omega \end{bmatrix} + \begin{bmatrix} \frac{1}{L} & 0 \\ 0 & \frac{1}{J} \end{bmatrix} \begin{bmatrix} u \\ \tau \end{bmatrix} \quad (10.8)$$

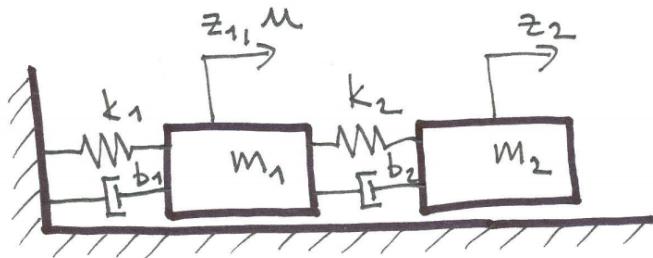
10.5.2 Model se statickou elektr. částí

$$u = Ri + k_M \omega \quad (10.9)$$

$$J\dot{\omega} = k_M i - b\omega + \tau, \quad (10.10)$$

$$J\dot{\omega} = -\left(\frac{k_M^2}{R} + b\right)\omega + \frac{k_M}{R}u + \tau. \quad (10.11)$$

10.6 Dvě hmoty s pružinou a tlumičem (fixované k rámu)



Obrázek 10.4 Schéma mechanického systému s 2dof a jedním vstupem (síla působící na hmotu 1)

$$m_1 \ddot{z}_1 = -k_1 z_1 + k_2(z_2 - z_1) - b_1 \dot{z}_1 + b_2(\dot{z}_2 - \dot{z}_1) + u$$

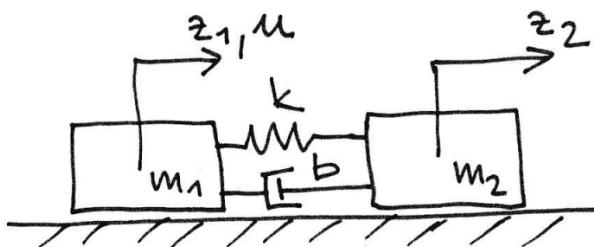
$$m_2 \ddot{z}_2 = -k_2(z_2 - z_1) - b_2(\dot{z}_2 - \dot{z}_1)$$

$$\begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix} \begin{bmatrix} \ddot{z}_1 \\ \ddot{z}_2 \end{bmatrix} = - \begin{bmatrix} k_1 + k_2 & -k_2 \\ -k_2 & k_2 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} - \begin{bmatrix} b_1 + b_2 & -b_2 \\ -b_2 & b_2 \end{bmatrix} \begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \end{bmatrix} + \begin{bmatrix} u \\ 0 \end{bmatrix} \quad (10.12)$$

$$\dot{\mathbf{x}} = [z_1, z_2, \dot{z}_1, \dot{z}_2]^T$$

$$\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{Bu} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{-k_1 - k_2}{m_1} & \frac{k_2}{m_1} & \frac{-b_1 - b_2}{m_1} & \frac{b_2}{m_1} \\ \frac{k_2}{m_2} & \frac{-k_2}{m_2} & \frac{b_2}{m_2} & \frac{-b_2}{m_2} \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{m_1} \\ 0 \end{bmatrix} u \quad (10.13)$$

10.7 Dvě hmoty s pružinou a tlumičem (volné)

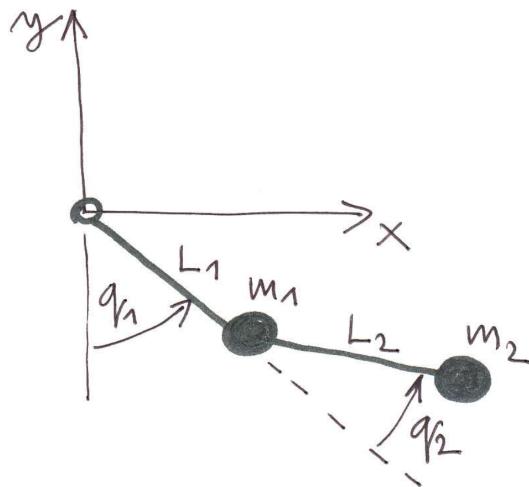


Obrázek 10.5 Dvě hmoty s pružinou a tlumičem (volné)

$$m_1 \ddot{z}_1 = k(z_2 - z_1) + b(\dot{z}_2 - \dot{z}_1) + u \quad (10.14)$$

$$m_2 \ddot{z}_2 = -k(z_2 - z_1) - b(\dot{z}_2 - \dot{z}_1) \quad (10.15)$$

10.8 Dvojité kyvadlo



Obrázek 10.6 Schéma
dvojitého kyvadla

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \mathbf{0} \quad (10.16)$$

Jednotlivé matice mají tento tvar:

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} m_1 L_1^2 + m_2 (L_1^2 + 2L_1 L_2 \cos(q_2) + L_2^2) & m_2 (L_2^2 + L_1 L_2 \cos(q_2)) \\ m_2 (L_2^2 + L_1 L_2 \cos(q_2)) & m_2 L_2^2 \end{bmatrix} \quad (10.17)$$

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{bmatrix} -2m_2 L_1 L_2 \dot{q}_2 \sin(q_2) & -m_2 L_1 L_2 \dot{q}_2 \sin(q_2) \\ m_2 L_1 L_2 \dot{q}_1 \sin(q_2) & 0 \end{bmatrix} \quad (10.18)$$

$$\mathbf{g}(\mathbf{q}) = \begin{bmatrix} (m_1 + m_2)gL_1 \sin(q_1) + m_2 g L_2 \sin(q_1 + q_2) \\ m_2 g L_2 \sin(q_1 + q_2) \end{bmatrix} \quad (10.19)$$

Použitá a doporučená literatura

- [1] K.J. Åström and B. Wittenmark, *Computer-controlled systems: theory and design (2nd ed.)* (Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990).
- [2] K. Dutton, S. Thompson, and B. Barraclough, *The Art of Control Engineering* (Prentice Hall, 1997).
- [3] R. Grepl, *Kinematika a dynamika mechatronických systémů* (CERM, Akademické nakladatelství, 2007).
- [4] R. Isermann, Mechatronic systems—Innovative products with embedded control, *Control Engineering Practice* **16**(1), 14 - 29 (2008).
- [5] L. Ljung, *System Identification - Theory for the User* (Prentice Hall, 1999).
- [6] Madsen2004, *Madsen2004* (2004).
- [7] Mathworks, *System Identification Toolbox*, Mathworks (2017).
- [8] Mathworks, MATLAB help, <https://www.mathworks.com/>, 2018.
- [9] Maxon, "Position Regulation with Feed Forward", *Application Note to EPOS Maxon control unit* (2008).
- [10] O. Nelles, *Nonlinear System Identification* (Springer, 2001).
- [11] J. Roubal and Petr Hušek a spol., *Regulační technika v příkladech* (BEN - technická literatura, 2011).
- [12] J. Skalický, *Teorie řízení* (FEKT, VUT v Brně, 2002).
- [13] C.M. University, Control Tutorial for Matlab, <http://ctms.engin.umich.edu/CTMS/>, 2018.
- [14] M. Valášek, *Mechatronika* (Praha: ČVUT, 1. vyd., ISBN: 80-01-01276-X, 1995).
- [15] Wikipedia, <https://www.wikipedia.org/>, 2018.

Člověk by měl umět vyměnit plenky, naplánovat invazi, porazit vepře, velet na lodi, navrhnut dům, napsat sonet, mít srovnáne účty, postavit zed, narovnat kost, utěšit umírajícího, plnit rozkazy, rozdávat rozkazy, spolupracovat, poradit si sám, řešit rovnice, vyhodnotit nový problém, kydat hnůj, naprogramovat počítač, uvařit chutné jídlo, účinně bojovat, statečně zemřít. Specializace je dobrá akorát pro hmyz.

Robert Heinlein