Základy práce s programem Simulink

Michal Široký

Michal Široký, 2007

Úvod

Tato příručka je určena především studentům předmětů SIMUL, KY, TŘ a SM, vyučovaných Katedrou kybernetiky Fakulty aplikovaných věd Západočeské univerzity v Plzni.

Cílem příručky je být pomocným, doplňujícím a rozšiřujícím materiálem pro studenty, kteří si potřebují osvojit základy práce v programu Simulink, který je součástí výpočetního prostředí Matlab a slouží k simulacím chování dynamických systémů.

Čtenář se v tomto textu seznámí nejprve se základní obsluhou programu Simulink[1], počínaje jeho spuštěním a zorientováním se v ovládacích prvcích. Poté bude následovat přehled základních stavebních prvků pro tvorbu simulačních modelů a nakonec bude na příkladech předvedeno, jak se takové modely vytvářejí a jakým způsobem s nimi lze pracovat.

[1] Ke tvorbě příkladů obsažených v této publikaci byl použit Matlab verze *7.0.1 (R14) SP1* a Simulink verze *6.1 (R14SP1)*. V jiných verzích těchto programů se mohou vyskytnout menší či větší odlišnosti v jejich ovládání.

Kapitola 1

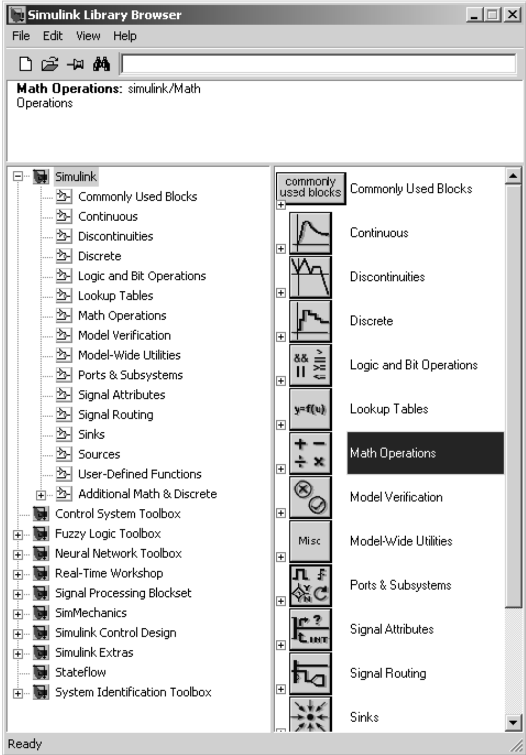
Spuštění Simulinku

Nejprve spustíme program Matlab a do příkazového okna napíšeme příkaz `simulink`. Objeví se okno nadepsané `Simulink Library Browswer` (prohlížeč knihoven), které by mělo vypadat podobně jako na obrázku 1.1. V panelu na levé straně okna vidíme ve stromové struktuře jednotlivé knihovny bloků a po vybrání knihovny se na pravé straně okna objeví seznam bloků v knihovně. Blok je základní stavební jednotka modelů v programu Simulink. Každý blok ve schématu modelu reprezetuje nějakou vlastnost nebo operaci.

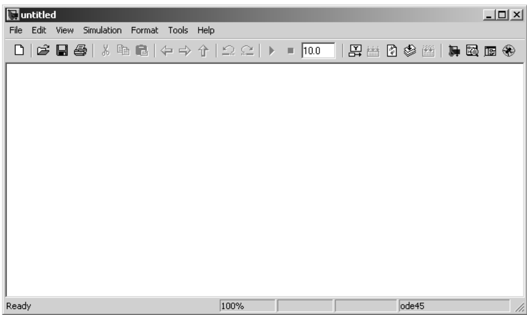
Schémata se vytvářejí v grafickém prostředí a abychom si mohli nějaké schéma sestavit, je nejprve nutné otevřít okno tohoto prostředí.

V prohlížeči knihoven aktivujeme volbu `File > New > Model`, čímž se otevře grafické prostředí pro tvorbu modelů, které je znázorněno na obrázku 1.2.

Pro další postup čtenáři doporučuji, aby si prošel následující kapitolu, která uvádí nejčastěji používané bloky a jejich funkce.



Obrázek 1.1: Prohlížeč knihoven



Obrázek 1.2: Okno grafického prostředí pro tvorbu modelů

Kapitola 2

Hlavní bloky

Jak již bylo zmíněno, každý blok reprezetuje nějakou vlastnost či funkci simulovaného systému, nebo operaci, kterou systém se signály jím procházejícími provádí, přičemž je dobré si uvědomit, že všechny funkce jsou v Simulinku funkcí času, přesněji řečeno simulačního času, jehož rychlost závisí na výkonu počítače, na kterém simulace probíhá. Obecně simulační čas běží mnohem rychleji než čas skutečný. Simulace vyjadřujíci dění v intervalu dvaceti minut může být vypočítána takřka v okamžiku.

Jednotlivé bloky z knihovny do okna modelu umísťujeme přetažením myší a u většiny bloků pak můžeme nastavovat jejich parametry v okně, které se otevře poklepáním na blok.

V našem výkladu se budeme zabývat pouze bloky z knihovny `Simulink`.

2.1 Sources

Sine Wave

Tento blok je generátorem funkce sinus s následujícími parametry výstupu:

$$y(t) = \mathit{Amp} \cdot \sin(2 \cdot \pi \cdot \mathit{Freq} \cdot t + \mathit{Phase}) + \mathit{Bias} \ ,$$

kde *Amp* je amplituda, *Freq* frekvence, *Phase* posun po horizontální ose a *Bias* posun po vertikální ose.

Step

Je generátorem skokové funkce z hodnoty `Initial value` na hodnotu `Final value` v simulačním čase `Step time`.

Clock (hodiny)

Jedná se o blok, jehož výstup odpovídá uplynulému simulačnímu času.

Constant

Tento blok generuje výstup s konstantní zadanou hodnotou. Tato hodnota se nastavuje v kolonce `Constant value`.

In (In1)

Představuje vstupní svorku pro signál do subsystému.

Ramp

Generuje tzv. rampovou funkci, což je signál, jehož průběhem je přímka s rovnicí $x = kt + q$. Ve vlastnostech bloku se nastavují tyto parametry:

`Slope` představuje k (směrnici přímky).

`Start time` udává hodnotu simulačního času, od které má začít generování rampového výstupu.

`Initial output` pak představuje q (posunutí po vertikální ose).

Ground

Používá se k „uzemění“ případných nepoužitých vstupních svorek u jiných bloků, čímž se při simulaci zamezuje, aby Simulink vypisoval varovné zprávy upozorňující na nezapojené vstupní svorky. Výstupem tohoto bloku je nula.

2.2 Sinks

Scope

Jde o jakési kukátko, nebo zapisovač, který graficky zaznamenává průběh signálů, které do něj přivedeme.

Display

Zobrazuje okamžitou číselnou hodnotu signálu, který je do něj přiveden. Pokud do bloku **Display** přivedeme signál ve formě vektoru, bude ukazovat hodnotu každé složky v samostatném okénku.

Terminator
Tento blok se používá k zaslepení případných nepoužitých výstupních svorek u jiných bloků, čímž se při simulaci zamezuje, aby Simulink vypisoval varovné zprávy upozorňující na nezapojené výstupní svorky.

To Workspace
Ukládá hodnoty přivedeného signálu do definované proměnné v pracovním prostoru Matlabu. Název výstupní proměnné je možno nastavit v políčku `Variable name`. Data uložená do této proměnné lze číst po zastavení, popřípadě pauzování simulace.

Out (Out1)
Představuje výstupní svorku signálu ze subsystému.

XY Graph
Jedná se v podstatě o souřadnicový zapisovač, který do plochy během simulace zakresluje body, jejichž vodorovná souřadnice odpovídá hodnotě signálu přivedeného na horní svorku a svislá souřadnice hodnotě přivedené na dolní svorku tohoto bloku. Po spuštění simulace se pak otevře okénko, kde můžeme vykreslování sledovat.

2.3 Continuous

Integrator
Numericky integruje hodnoty vstupního signálu v závislosti na čase. Výstupem integrátoru v daném čase je hodnota určitého integrálu vstupu od spuštění simulace do aktuálního simulačního času.

Do kolonky `Inital condition` se uvádějí počáteční podmínky (počáteční hodnota výstupu) integrátoru. Můžeme zapsat přímo číselnou hodnotu, nebo název proměnné uložené v aktuálním Matlab workspace, která tuto hodnotu reprezentuje.

Derivative
Numericky derivuje vstupní signál podle času. Výstupem derivátoru v daném čase je hodnota derivace vstupního signálu v tomto čase (derivace v bodě).

Transfer Fcn (Transfer Function)

Reprezentuje model systému zadaného ve formě Laplaceova přenosu. Do políčka `Numerator` zadáme koeficienty v čitateli přenosu, do políčka `Denominator` pak koeficienty ve jmenovateli, oboje v pořadí od nejvyšší do nejnižší mocniny Laplaceovy proměnné.

State-Space

Tento blok reprezentuje systém ve formě stavového modelu. Do políček `A`, `B`, `C` a `D` zadáváme příslušné matice, popřípadě vektory či skaláry stavového modelu.

2.4 Math Operations

Abs

Výstupem tohoto bloku je absolutní hodnota vstupu.

Add

Slouží ke sčítání hodnot dvou (nebo více) vstupních signálů. Výstupem (výsledkem) je pak jeden jednorozměrný signál. V parametrech bloku je možno nastavit, kolik má mít vstupních svorek, a zda se signály do nich přivedené mají přičítat, nebo odečítat.

Divide

Násobí, nebo dělí vstupní signály. Je možno zvolit, zda má daná matematická operace probíhat maticově, či prvek po prvku.

V kolonce `Number of inputs` se definuje počet a typ vstupních svorek (násobení či dělení).

V kolonce `Multiplication` se definuje způsob provádění operace (prvek po prvku či maticově). Pokud vybereme maticový typ, operace dělení pak představuje násobení inverzní maticí.

Gain

Blok **Gain** násobí vstup hodnotou zadanou v kolonce `Gain`, což může být konstanta či vektor. Podobně jako u bloku **Divide**, je možno v kolonce `Multiplication` zvolit, zda má násobení mít prvkový, či maticový charakter.

2.5 Signal Routing

Demux (Demultiplexor)

Slouží k rozdělování vícerozměrných signálů. Je schopen rozdělovat vektorový signál (přivedený na vstupní svorku) na jeho jednotlivé složky.

Mux (Multiplexor)

Směšuje jednotlivé signály (přivedené na vstupní svorky) do jediného vícerozměrného signálu.

Goto

Umožňuje „bezdrátový“ přenos signálu v modelu. Signál přivedený na vstupní svorku zasílá do korespondujícího **From** bloku. V kolonce `Tag` se nastavuje název tohoto **From** bloku.

From

„Bezdrátově“ přijímá signál z bloku **Goto** definovaného v kolonce `Goto Tag`.

Manual Switch

Na výstupní svorku propouští signál z jedné ze dvou vstupních svorek. Přepnutí mezi svorkami se provádí dvojklikem na blok a funguje i během simulace.

2.6 Discontinuites

Saturation

Výstupem tohoto bloku je vstupní signál omezený shora hodnotou uvedenou v políčku `Upper limit` a zdola hodnotou v políčku `Lower limit`

Dead Zone

Schopností tohoto bloku je nahrazovat vstupní signál, jehož hodnota leží v oblasti necitlivosti, nulovou hodnotou výstupu. Oblast necitlivosti je vymezena hodnotami vepsanými do políček `Start of dead zone` (spodní hranice necitlivosti) a `End of dead zone` (horní hranice necitlivosti). Pokud hodnota vstupního signálu leží pod spodní hranicí necitlivosti, je na výstupu promítnuta tato hodnota pod nulovou osu právě o tolik, o kolik je menší než `End of dead zone`. Pokud je naopak hodnota na vstupu větší než horní hranice necitlivosti, je na výstupu promítnuta nad nulovou osu právě o tolik, o kolik je větší než `End of dead zone`.

2.7 User-Defined Functions

Fcn (Function)

Do tohoto bloku (do kolonky `Expression`) můžeme zapsat libovolný matematický výraz, jehož proměnou je vstup tohoto bloku (ve výrazu značen u). Výstupem je pak výsledek matematického výrazu. Podle toho, kolik rozměrů má vstup, můžeme se odkazovat na jednotlivé jeho složky jako na $u(1) \ldots u(n)$.

Kapitola 3

Příklady

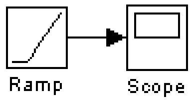
Nyní máme dostatečné znalosti k tomu, abychom mohli začít vytvářet v Simulinku vlastní modely dynamických soustav.

První model

Nejprve si otevřeme Matlab a potom prohlížeč knihoven Simulinku a okno grafického prostředí pro tvorbu modelů (dále jen *okno modelu*). Dále budeme potřebovat bloky **Ramp** a **Scope**, které si z knihovny do okna modelu umístíme přetažením myší. Bloky v okně modelu uspořádáme tak, aby byl **Scope** vpravo od **Ramp**. Na pravé straně bloku **Ramp** a na levé straně bloku **Scope** si můžete všimnout jakýchsi zobáčků. U **Ramp** jde o výstupní svorku a u **Scope** o vstupní svorku.

Levým tlačítkem myši chytíme výstupní svorku **Ramp** a přetáhneme ji na vstupní svorku **Gain**. Pokud se přetažení povedlo, propojili jsme výstup jednoho bloku se vstupem druhého a schéma vypadá jako na obrázku 3.1. Pokud jsme se netrefili na vstup **Scope**, bloky nebyly propojeny. Výstupní signál z **Ramp** pak končí volně v prostoru a je znázorněm přerušovanou červenou čárou s prázdnou šipkou. Tato situace je znazorněna na obrázku 3.2 a pokud nastala, musíme spojení opravit.

Na obou obrázcích si můžeme všimnout, že signál je vždy opatřen šipkou, která ukazuje směr jeho šíření. Pro práci v Simulinku je dobré brát na vědomí, že šipky značí to, že daný signál je orientovaný a šíří se jen „jednosměrně“. Simulink jako takový slouží pro řešení úloh popsatelných pomocí kauzálního programování, kdy je simulační schéma řešeno směrem od zdrojů (Sources) k „výpustím“ (Sinks).



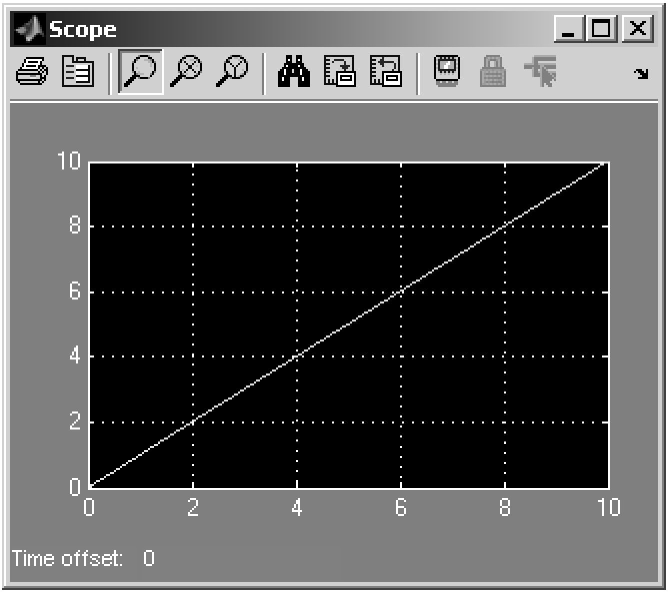
Obrázek 3.1: Správné propojení bloků

Obrázek 3.2: Chybné propojení bloků

Jako další krok zkontrolujeme nastavení velikosti časových kroků, po kterých se bude výpočet simulace „posouvat“. V okně modelu aktivujeme volbu `Simulation > Configuration Parameters`. V levém panelu okna, které se objeví zvolíme položku `Solver` a v pravém panelu pak najdeme položku `Max step size`, tedy maximální velikost kroku simulace. Tato hodnota je standardně nastavena na `auto`, což znamená, že ji určuje program sám. Někdy se ale stává, že program použije k simulování přiliš dlouhý krok a výsledky jsou pak nepřesné. Pak je vhodné nastavit maximální velikost kroku ručně. Pro řešní příkladů, s nimiž se většinou setkáme, obvykle volíme hodnotu `0.01`, což znamená, že simulace bude vyhodnocována po úsecích dlouhých nanejvýš 0,01 sekundy simulačního času. Pokud budete se Simulinkem řešit i jiné typy úloh, než jen ukázkové a „školní“ příklady, může se vám stát, že pro optimální výpočet úlohy budete muset volit délku kroku výrazně menší, či naopak větší. To ale bude zálažet na konkrétním případu. My pro naši úlohu nastavíme maximální délku kroku na 0,01 sekundy a změnu potvrdíme.

Naši pozornost přesuneme opět k oknu modelu. V jeho nabídkové liště vidíme textové políčko s číselnou hodnotou, která je obvykle `10.0`. Toto políčko udává délku simulačního času ve vteřinách. Jde časový interval od počátku simulace, pro který bude simulace vypočítána. Můžeme si to představit tak, že simulace bude vypočítána pro časový interval od 0 do 10 sekund. Délku simulace můžeme nastavit libovolně velkou a pokud chceme, aby běžela nepřetržitě, napíšeme do políčka `inf` (infinity). My necháme délku simulačního času `10.0`.

Dvojklikem na blok **Scope** otevřeme vykreslovací okno tohoto bloku, které je zatím prázdné. Nyní jsme připraveni spustit simulaci. V nabídkové liště okna modelu klepneme na spouštěcí „play“ trojúhelníček. Za okamžik je simulace dokončena a ve vykreslovacím okně vidíme průběh výstupního signálu bloku **Ramp**. Pro optimální rozsah os klepneme na ikonku dalekohledu. Vykreslovací okno by mělo vypadat jako na obrázku 3.3. Právě



Obrázek 3.3: Vizualizace průběhu rampové funkce pomocí **Scope**

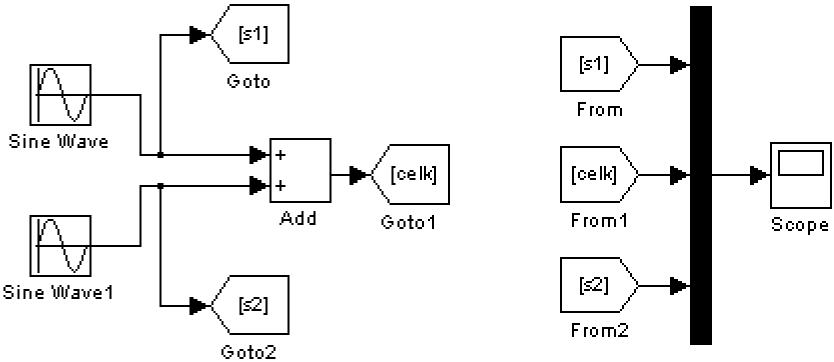
jsme sestavili náš první model, simulovali jeho chování a prohlédli si výsledky pokusu. Před tím, než přejdete k dalšímu příkladu, zkuste trochu experimentovat se změnami prarametrů bloků a simulace v tomto modelu.

Skládání harmonického vlnění

V tomto příkladu budeme simulací zjišťovat, co vznikne součtem dvou sinusových průběhů s blízkými frekvencemi.

Vytvoříme si nový soubor modelu a do jeho okna přetáhneme z knihovny 2 bloky **Sine Wave**, 3 bloky **Goto** a 3 bloky **From** a dále ještě **Mux**, **Add** a **Scope**. Bloky pospojujeme podle schématu na obrázku 3.4 a parametry v blocích a v simulaci nastavíme podle seznamu na následující stránce.

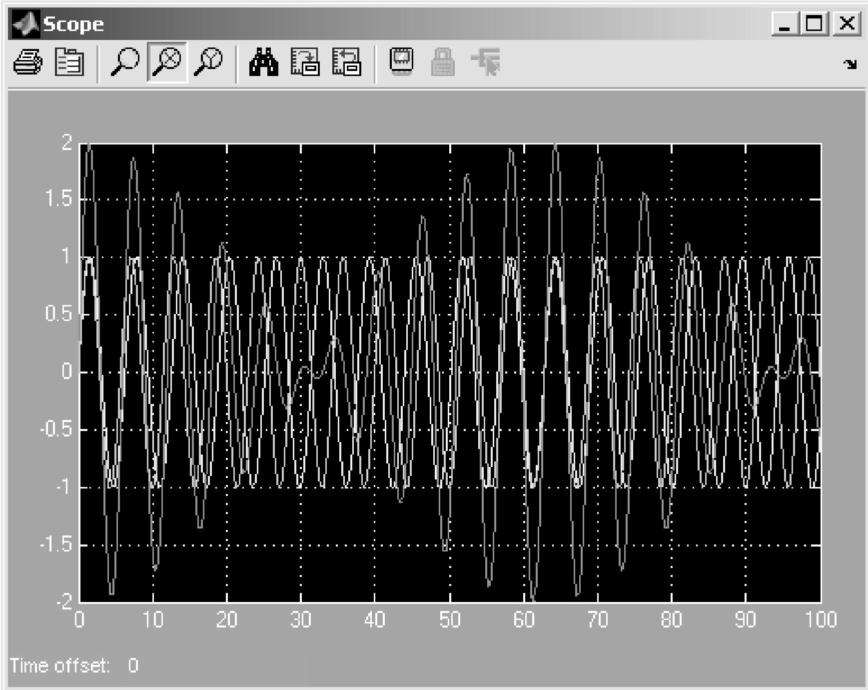
- Frekvenci u **Sine Wave** necháme, u **Sine Wave1** nastavíme na `1.1`.
- `Tag` u **Goto** nastavíme na `s1`, u **Goto1** na `celk` a u **Goto2** na `s2`.
- `Goto Tag` u **From** nastavíme na `s1`, u **From1** na `celk` a u **From2** na `s2`.
- Maximální velikost simulačního kroku definujeme rovnu `0.01` a délku simulace nastavíme na 100 sekund.



Obrázek 3.4: Schéma pro simulaci skládání harmonického vlnění

Nyní otevřeme vykreslovací okno bloku **Scope** a spustíme simulaci. Zobrazené průběhy by měly vypadat jako na obrázku 3.5. S amplitudou 1 kolem osy kmitají původní sinusové průběhy a výsledné vlnění vzniklé jejich součtem má proměnnou amplitudu.

Použití bloku **Scope** pro zobrazování výsledků simulace má výhodu v tom, že hodnoty zobrazuje ihned, jakmile jsou vypočteny, ale neumožňuje jejich export ani tisk. Pokud budeme chtít získaná data nějak dále zpracovávat, budme muset náš model trochu vylepšit. Do schématu tedy přidáme blok **To Workspace**, který vytažením jeho vstupní svorky napojíme na signál vedoucí ke **Scope**. Jako hodnotu `Variable name` ve vlastnostech **To Workspace** uvedeme `vlneni` a u položky `Save format` vybereme `Structure With Time`, což znamená, že hodnoty signálů, které vedou do **To Workspace** budou po skončení simulace uloženy do proměnné `vlneni`, která se vytvoří v aktuálním workspace Matlabu. Výsledné schéma by mělo vypadat jako na obrázku 3.6 na straně 18.



Obrázek 3.5: Vizualizace vlnění pomocí **Scope**

Nyní spustíme simulaci a po jejím dokončení se podíváme do obsahu workspace, kde by měla být proměnná `vlneni`, ve které jsou uloženy vypočtené hodnoty vlnění včetně časových okamžiků, ke kterým se tyto hodnoty vztahují. Výsledné vlnění pak můžeme nechat vykreslit následujícími příkazy:

```
figure
grid on
hold on
plot(vlneni.time,vlneni.signals.values(:,2),'k')
xlabel('cas [s]')
ylabel('amplituda')
title('Skladani vlneni')
```

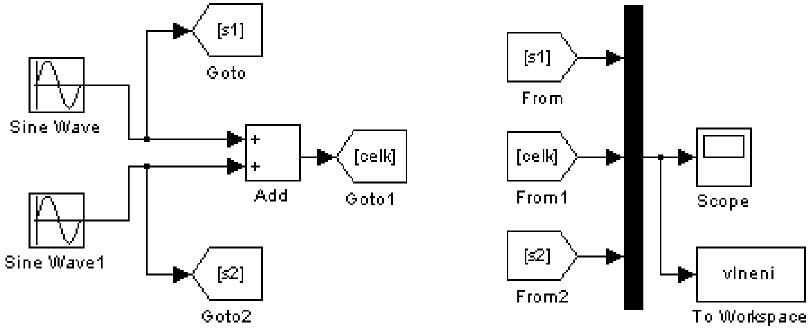
Matlab by měl vytvořit graf podobný tomu na obrázku 3.7 na straně 18. Hodnoty původních vlnění do grafu přidáme pomocí příkazů:

```
plot(vlneni.time,vlneni.signals.values(:,1),'--k')
plot(vlneni.time,vlneni.signals.values(:,3),':k')
```

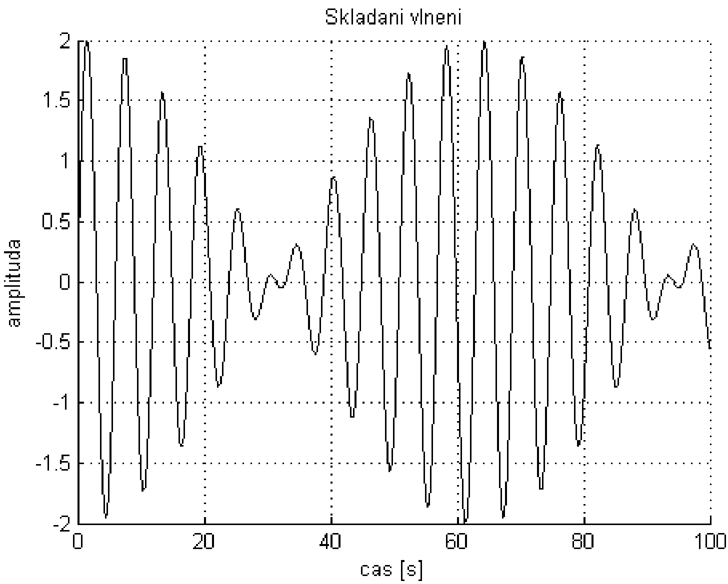
Graf vytvořený pomocí funkce `plot` jde již snadno uložit do obrazového formátu, nebo přímo vytisknout.

Další příklad se bude týkat reálné problematiky, se kterou se studenti při výuce setkávají, a to simulace chování systémů popsaných diferenciálními rovnicemi.

Než se ale do dalšího příkladu pustíte, zkuste ještě trochu experimentovat s aktuálním modelem. Zkuste například zjistit, jak bude změna rozdílu ve frekvencích skládaných vlnění ovlivňovat vlnění výsledné.



Obrázek 3.6: Simulační schéma s výstupem do workspace



Obrázek 3.7: Vizualizace vlnění pomocí funkce `plot`

Řešení lineární diferenciální rovnice 2. řádu s konstantními koeficienty

Rovnici máme nejprve zadánu v obecném tvaru:

$$a_2y''(t) + a_1y'(t) + a_0y(t) = b_0u(t);$$
$$y(0) = y_{p0},\ y'(0) = y_{p1},\ u(0) = u_{p0}$$

Pokud se na zadanou diferenciální rovnici díváme jako na popis chování nějakého reálného systému, pak můžeme říci, že $y(t)$ představuje výstup systému závislý na čase, $y'(t)$ a $y''(t)$ jeho derivace a $u(t)$ je vstup systému.

Dále jsou zde také obecně zadány počáteční podmínky systému (y_{p0}, y_{p1} a u_{p0}).

Nyní z rovnice formálně vyjádříme jednotlivé derivace proměnné $y(t)$:

$$\begin{aligned} y''(t) &= -\frac{a_1}{a_2}y'(t) - \frac{a_0}{a_2}y(t) + \frac{b_0}{a_2}u(t) \\ y'(t) &= \int_0^t y''(\tau)d\tau \\ y(t) &= \int_0^t y'(\tau)d\tau \end{aligned}$$

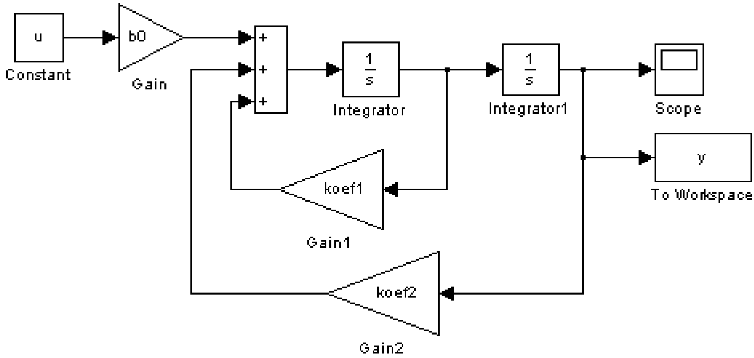
Tohoto zápisu nyní využijeme ke grafickému řešení této rovnice pomocí programu Simulink.

Pokud budeme chtít zjistit odezvu systému na jednotkový skok, použijeme jako hodnotu vstupu systému $u(t)$ blok **Constant** s hodnotou výstupu nastavenou na 1.

Dále využijeme bloky **Gain**, **Integrator**, **Sum**, **Scope** a také blok **To Workspace**.

Bloky pospojujeme podle schématu na obrázku 3.8. Otočení bloků **Gain1** a **Gain2** do správného směru docílíme následujícím způsobem: Klepneme na blok pravým tlačítkem a v kontextovém menu zvolíme `Format > Flip Block`.

`Constant value` v **Constant** nastavíme na proměnnou `u`. (Hodnotu této a dalších proměnných nadefinujeme v příkazovém okně Matlabu až před spuštěním simulace.) Zesílení v **Gain** nastavíme na `b0`, v **Gain1** na `koef1`, v **Gain2** na `koef2`. Počáteční podmínku v **Integrator** nazveme `yp1` a v **Integrator1** `yp0`. Název výstupní proměnné v **To Workspace** nastavíme na `y` a `Save format` na `Structure With Time`.



Obrázek 3.8: Simulační schéma pro řešení diferenciální rovnice 2. řádu

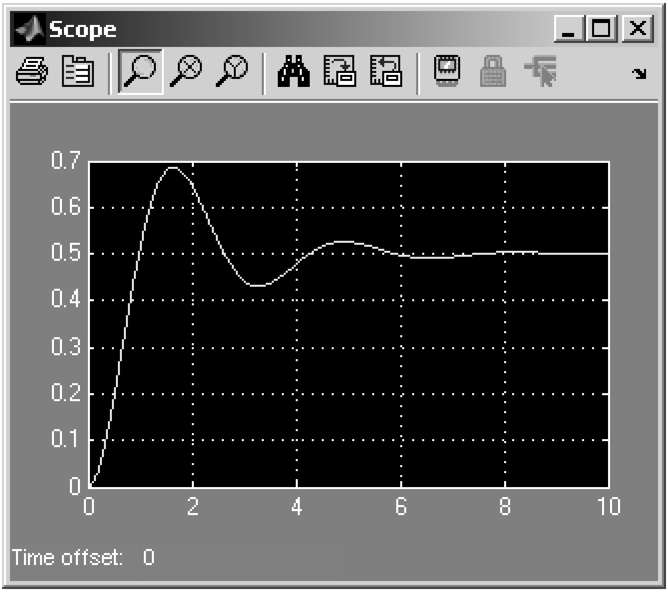
Nyní nastává oakmžik k odhalení hodnot koeficientů a počátečních podmínek diferenciální rovnice. Hodnoty budou následující: $a_2 = 1$; $a_1 = 1,2$; $a_0 = 4$; $b_0 = 2$; $y_{p0} = 0$; $y_{p1} = 0$; $u_{p0} = 1$.

Pro nadefinování proměnných v Matlab workspace do příkazového okna Matlabu napíšeme následující příkazy:

```
a0=4
a1=1.2
a2=1
b0=2
yp0=0
yp1=0
u=1
koef1=-a1/a2
koef2=-a0/a2
```

Možná se teď budete ptát, proč jsme hodnoty koeficientů nenapsali přímo do jednotlivých bloků. Odpověď je jednoduchá: Tím, že jsme do bloků napsali názvy proměnných, mohli jsme pak jejich hodnoty nadefinovat hromadně v příkazovém okně a stejným způsobem je můžeme také v případě potřeby hromadně změnit, což je velmi výhodné.

Nyní nastavíme maximální krok simulace na 0,01 skundy, délku simulace na 10 sekund, otevřeme vykreslovací okno **Scope** a spustíme simulaci. Po optimalizaci rozsahu os by výsledek měl vypadat jako na obrázku 3.9. Před očima máme graf funkce, která řeší zadanou diferenciální rovnici v intervalu od 0 do 10 sekund. Současně se jedná o odezvu zkoumaného systému na jednotkový skok. Současně s dokončením simulace se ve workspace obje-

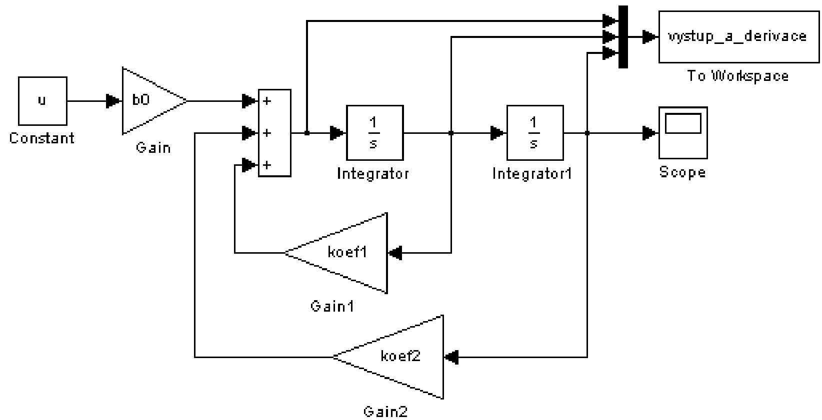


Obrázek 3.9: Vizualizace řešení diferenciální rovnice pomocí **Scope**

vila nová proměnná y, která obsahuje stajná data, která jsou zobrazena ve **Scope**. Obsah proměnné y můžeme nechat vykreslit sledem těchto příkazů:

```
figure
grid on
hold on
plot(y.time,y.signals.values(:,1),'k')
xlabel('cas [s]')
ylabel('amplituda')
title(strcat('Odezva na konstantni skok o velikosti u=',
      num2str(u)))
```

V některých úlohách, které jsou zařazeny například v předmětu Kybernetika, mají studenti za úkol graficky znázornit také závislost výstupu systému $y(t)$ na $y'(t)$, závislost $y(t)$ na $y''(t)$ a $y'(t)$ na $y''(t)$. My na našem modelu budeme tyto závislosti zjišťovat při nulovém vstupu, tj. $u(t) = 0$ a počátečních podmínkách $y(0) = 1$ a $y'(0) = 1$. Pro náš experiment upravíme model podle schématu na obrázku 3.10.



Obrázek 3.10: Předchozí simulační schéma s výstupem do workspace

Výstupní proměnnou v **To Workspace** nazveme `vystup_a_derivace`. Koeficienty a počáteční podnínky rovnice systému nadefinujeme v příkazovém okně následujícím způsobem:

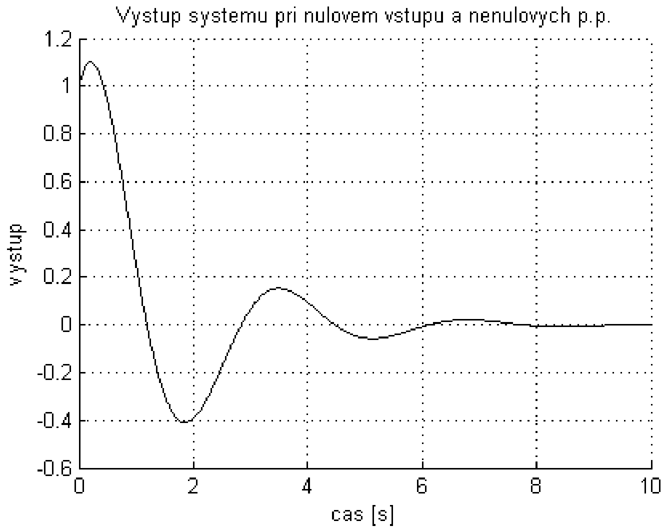
```
a0=4
a1=1.2
a2=1
b0=2
yp0=1 %pocatecni podminka bloku Integrator1
yp1=1 %pocatecni podminky bloku Integrator
u=0   %nulovy vstup
koef1=-a1/a2
koef2=-a0/a2
```

Spustíme simulaci a ve vykreslovacím okně **Scope** nyní vidíme graf funkce, která řeší diferenciální rovnici $y''(t) + 1.2y'(t) + 4y(t) = 0$ za těchto počátečních podmínek: $y(0) = 1$, $y'(0) = 1$.

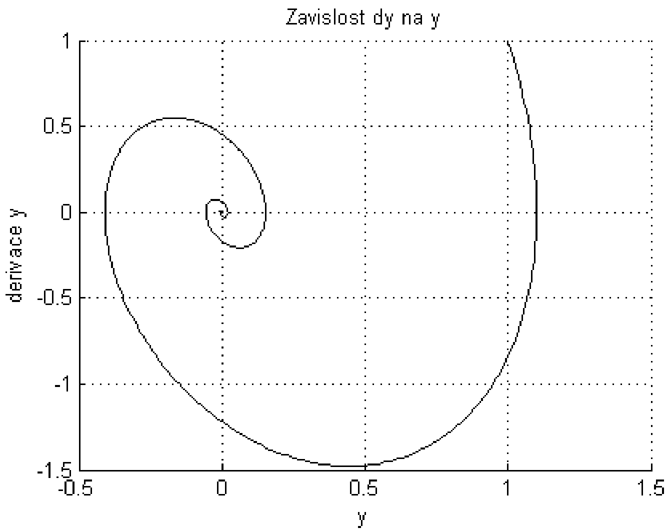
Kromě tohoto výsledku nás také zajímají výše zmiňované závislosti výstupu systému a jednotlivých jeho derivací. Potřebná data se po skončení simulace uložila do proměnné `vystup_a_derivace`. Následujícími příkazy si necháme výsledky vykreslit:

```
cas = vystup_a_derivace.time;
d2y = vystup_a_derivace.signals.values(:,1);
dy = vystup_a_derivace.signals.values(:,2);
y = vystup_a_derivace.signals.values(:,3);

figure
grid on
hold on
plot(cas, y, 'k')
xlabel('cas [s]')
ylabel('vystup')
title('Vystup systemu pri nulovem vstupu a nenulovych p.p.')

figure
grid on
hold on
plot(y, dy, 'k')
xlabel('y')
ylabel('derivace y')
title('Zavislost dy na y')

figure
grid on
hold on
plot(y, d2y, 'k')
xlabel('y')
ylabel('druha derivace y')
title('Zavislost d2y na y')

figure
grid on
hold on
plot(dy, d2y, 'k')
xlabel('derivace y')
ylabel('druha derivace y')
title('Zavislost d2y na dy')
```

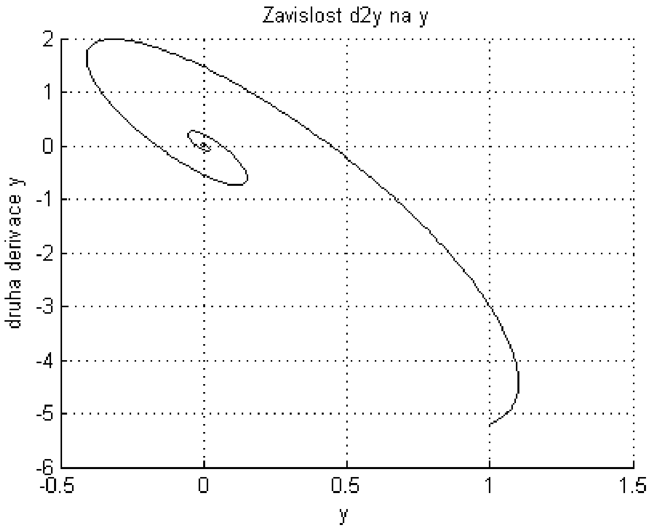
Matlab nám vykreslí grafy znázorněné na obrázcích 3.11 až 3.14.



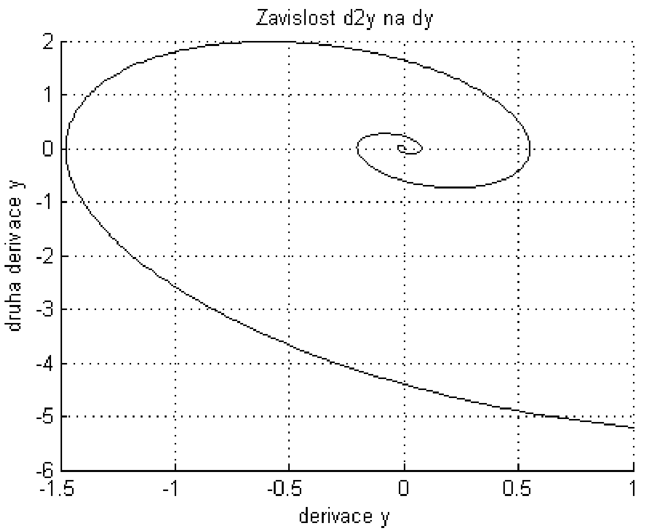
Obrázek 3.11: Výstup soustavy při nulovém vstupu



Obrázek 3.12: Závislost $y'(t)$ na $y(t)$



Obrázek 3.13: Závislost $y''(t)$ na $y(t)$



Obrázek 3.14: Závislost $y''(t)$ na $y'(t)$

Slovo závěrem

Pokud jste úspěšně vypracovali uvedené příklady a dočetli až sem, máte dostatečné znalosti a zkušenosti v práci s programem Simulink, abyste se mohli sami pustit do experimentování s tímto programem a s modely jím vytvořenými.

Potřebné infromace k další práci se Simulinkem a Matlabem naleznete ve vestavěné nápovědě programu, nebo na internetových stránkách firmy *The MathWorks, Inc* [1], či v publikaci *Matlab: Sbírka jednoduchých příkladů pro řešení elektrotechnických a fyzikálních úloh* [2].

Velmi dobrou literaturou k pochopení základních vztahů mezi reálnými systémy a jejich modely jsou skripta *Kybernetika* [3] a k prohloubení těchto znalostí poslouží například skripta *Teorie řízení* [4].

Tímto vám přeji mnoho pokroků v ovládání programu Simulink a vašem dalším studiu.

V závěru děkuji Ing. Jaroslavu Sobotovi za užitečné rady a připomínky, které pomohly zvýšit odbornou kvalitu tohoto textu.

Horní Bříza, 2007

Autor

Literatura

[1] The MathWorks, Inc: *http://www.mathworks.com/*

[2] J. Büllow: *Matlab: Sbírka jednoduchých příkladů pro řešení elektrotechnických a fyzikálních úloh, ZČU Plzeň, 2007*

[3] F. Tůma: *Kybernetika, ZČU Plzeň, 1966; reedice 1997, 1998*

[4] F. Tůma: *Teorie Řízení, ZČU Plzeň, 2005*

Obsah

Rejstřík