Matlab, Simulink a matematické modelování

Miroslav Vlček, Miroslav Svítek,
Jan Přikryl, Bohumil Kovář a Martin Pěnička

Katedra aplikované matematiky
FD ČVUT Praha

http://euler.fd.cvut.cz/predmety/msp/

20. dubna 2006

Abstrakt

Toto je vývojová verze stručného manuálu k Matlabu a Simulinku pro použití při samostudiu a při cvičeních s předmětu K611MSAP (Modelování systémů a procesů). Je zcela určitě plná chyb a překlepů, berte to prosím v úvahu.

Změny

2006/04/06 jp Vypreparováno z různých dostupných textů. Nesedí některé odkazy, některé staré obrázky jsou špatně.

2006/04/07 jp Přidán model nabídka-poptávka.

2006/04/20 jp Změněny některé obrázky.

Obsah

Kapitola 1

Jemný úvod do Matlabu

Programový systém Matlab je velmi efektivním nástrojem pro vědecké a inženýrské výpočty v oblastech, kde se uplatňuje maticový počet. K jeho hlavním přednostem patří zejména jeho otevřenost tj. možnost rozšiřování o vlastní funkce, značný počet problémově orientovaných balíků již hotových funkcí tzv. toolboxů, poměrně jednoduchá syntaxe a kvalitní implementované algoritmy.

Po spuštění programu Matlab se otevře příkazové okno, tzv. command window. Rozvržení menu je podobné jako u jiných aplikací pod okny: File, Edit, Window a Help. Systém Matlab je casesensitive tzn., že rozlišuje např. stejně jako UNIX, velká a malá písmena. Příkazový řádek začíná promptem >>, za který se píší příkazy. V Matlabu není přísné deklarování typu proměnné tak jako např. v Pascalu, proměnné nejsou deklarovány, ale jsou definovány teprve při prvním přiřazení hodnoty. Příkaz pro přiřazení má tvar:

`>> promenna=vyraz`

Nevíme-li si rady, systém nám po odeslání příkazu hlásí chybu nebo pokud si nejsme jisti, jaké parametry může mít daný příkaz, použijeme příkaz:

`help` resp. `help < příkaz >`

Napíšeme-li příkaz `help help` dostaneme kompletní výpis popisu tohoto příkazu, jeho syntaxi a seznam příkazů, které se k `help` vztahují. Potlačení výpisu po provedení příkazu lze nastavit tím, že po příkazu v příkazové řádce napíšeme středník ";"

Přestože není účelem tohoto textu nahrazovat manuál Matlabu, v dalším uvedeme jeho základní vlastnosti. Příkazu `lookfor` použijeme k vyhledání určitého řetězce v souborech helpu. Příkaz pro vyhledání má syntaxi:

`lookfor < hledaný řetězec >`,

po odeslání následuje výpis nalezených příkazů, které obsahují hledaný řetězec. Jednoduchým příkazem demo spustíme v Matlabu prezentaci, ve níž si můžeme zopakovat naše znalosti z předmětů Úvod do počítačů I. . . , n.

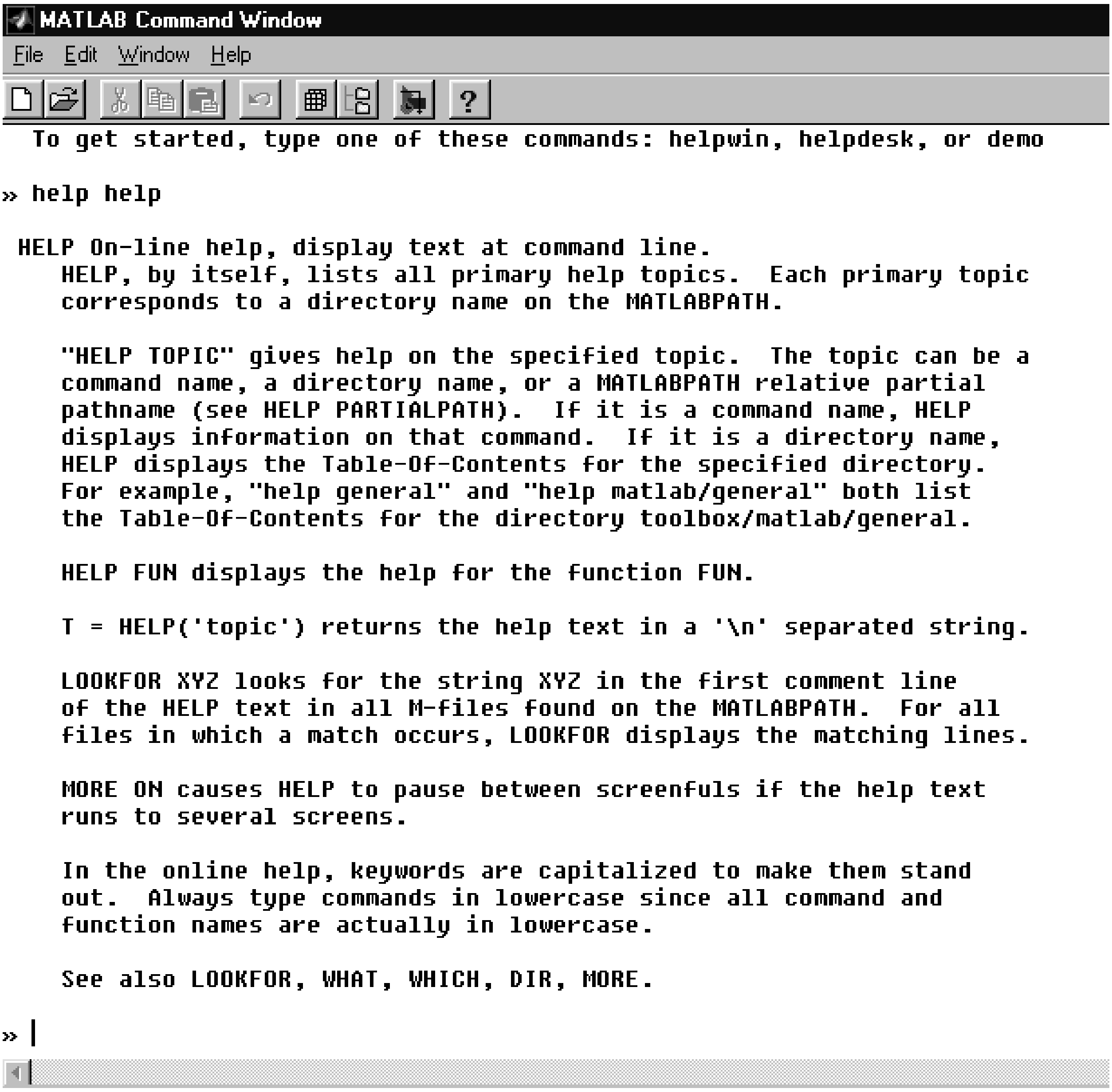
1.1 Základní matematické operace

1.1.1 Zápis komplexního čísla

Mějme komplexní číslo z ve složkovém tvaru:

$$z = a + ib$$

kde a, b jsou reálná čísla a i je imaginární jednotka Definujme v Matlabu komplexní číslo $z = 1 + i$. Za prompt >> zadáme číslo `z` následujícím způsobem:



Obrázek 1.1: Command window v Matlabu

```
>> z=1+i
```

Po zadání nám Matlab vypíše: `1.0000 + 1.0000i` Budeme-li chtít zjistit u komplexního čísla jeho reálnou část $Re = a$ resp. imaginární část $Im = b$ napíšeme příkaz `real(z)`, resp. `imag(z)` Po odeslání následuje výpis:

```
>> ans =1
```

Převeďme komplexní číslo z do goniometrického tvaru, tj. do tvaru:

$$z = |\, z\, |\, (\cos\gamma + i \sin\gamma)$$

kde γ je argument komplexního čísla z a $|\, z\, |$ je modul komplexního čísla z . Argument $\gamma = \arctan(Re/Im)$. Argument komplexního čísla z zjistíte pomocí příkazu

```
>> angle(z)
```

modul $\mid z \mid = \sqrt{Re^2 + Im^2}$, vrací příkaz

```
>> abs(z)
```

1.1.2 Vektory a matice v Matlabu

Protože jsou proměnné v Matlabu presentovány maticemi, budeme se jim věnovat o trochu více.

- Vytvořme v Matlabu vektor u.

```
>> u=[2 4 5]
```

  V Matlabu se jedná o matici 1×3:

```
u˜=

    2     4     5
```

- Vytvořme sloupcový vektor v

```
>> v=[2;4;5]=[2 4 5]'
```

  V Matlabu se jedná o matici s jedním sloupcem o třech řádcích:

```
v˜=

    2
    4
    5
```

- Vygenerujme vektor. Pomocí příkazu

```
>> w=2:5
```

  můžeme vygenerovat vektor $w = [2 \quad 3 \quad 4 \quad 5]$ s krokem 1. Matlab nám vrátí:

```
  w =

    2     3     4     5
```

  Pomocí příkazu

```
>> u=1:2:7
```

  vygenerujeme vektor $u = [1 \quad 3 \quad 5 \quad 7]$ s krokem 2. Matlab vrací výpis:

```
   u˜=

    1     3     5     7
```

- Zadejme matici A z příkazové řádky:

```
>> A=[1 2 3;4 5 6]
```

Matlab nám vrátí:

```
A~=

   1     2     3
   4     5     6
```

Což je matice:

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Ze syntaxe příkazu je zřejmé, že matice se v Matlabu zadávají do hranatých závorek po řádcích, prvky v řádku jsou odděleny mezerou, jednotlivé řádky jsou potom odděleny středníkem.

- Vygenerujme čtvercovou matici, jejíž hlavní diagonálu tvoří vektor u. Použijeme příkazu:

```
>> U=diag(u)
```

Matlab nám vrátí:

```
U~=

     1     0     0     0
     0     3     0     0
     0     0     5     0
     0     0     0     7
```

což je matice:

$$\mathbf{U} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 7 \end{pmatrix}$$

- Je-li třeba vygenerovat nulovou matici, použijeme příkazu:

```
>> zeros(m,n)
```

kde čísla (m,n) znamenají dimensi generované nulové matice. Zadejme:

```
>> m=5
m =
     5

>> n=3
n =
     3
```

po zadání příkazu `>> B=zeros(m,n)`, nám Matlab vypíše:

```
B =
      0      0      0
      0      0      0
      0      0      0
      0      0      0
      0      0      0
```

- V Matlabu existuje podobný příkaz pro generování matice jedniček:

  ```
  >> ones(m,n)
  ```

  generuje jedničkovou matici dimense (m,n).

- Vytvořme jednotkovou matici.
  E dimense n použitím příkazu:

  ```
  >> E=eye(n)
  ```

  Kde n je dimensí matice. Jednotkovou matici dimense 3 vytvoříme tedy pomocí příkazu: `>> E=eye(3).`
  Matlab nám vrátí:

  ```
  E =

        1      0      0
        0      1      0
        0      0      1
  ```

  Což je v matematickém zápisu:

  $$\mathbf{E} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- N-tý sloupec matice vybereme pomocí příkazu

  ```
  >> u=E(:,n)
  ```

  Příklad 1.1 (výběr sloupce matice)
  Vyberte první sloupec matice E a uložte ho do proměnné u.
  Řešení:

  ```
  >>  u=E(:,1)

  u~=

        1
        0
        0
  ```

- Prvek matice zaměníme pomocí příkazu přiřazení:

```
>> E(m,n)=e
```

kde (m,n) je prvek na pozici m-té řádky a n-tého sloupce.

Příklad 1.2 (záměna prvku matice)
Zaměňte prvek na pozici (m,n) matice E.
Řešení:
Použijeme příkaz přiřazení `>> E(3,1)=5`, *Matlab vám vypíše matici:*

```
E =

     1     0     0
     0     1     0
     5     0     1
```

- Příkaz záměny m-té řádky matice A vektorem v

```
>> A(m,:)=v
```

- Příkaz záměny n-tého sloupce matice A vektorem v:

```
>> A(:,n)=v
```

- Výběr submatice:

```
>> A(m:i;n:j)
```

vrací řádky od m-té do i-té a sloupce od n-tého do j-tého jako submatici.

- Výběr řádků matice:

```
>> A([a b],:)
```

vrací řádky a a b a všechny sloupce n jako matici 2 x n.

- Násobení matic. Je dána matice $A = (a_{ij})$ typu (m,n) a matice $B = (b_{ij})$ typu (n,p). Součin matic A a B v tomto pořadí je potom matice $C = (c_{ij})$ typu (m,p)definována předpisem:

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + a_{i3}b_{3j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj} \tag{1.1}$$

Příklad 1.3 (násobení matic)
Vynásobte matice:

```
>> a=[1 2;3 4]

a =

     1     2
     3     4
```

```
>> b=[2 4]


b =

     2     4
```

Řešení:

```
>> C=b*a


c =

    14    20
```

Zkuste vynásobit matice v opačném pořadí, abyste si prohlédli chybové hlášení Matlabu.

- Transpozice matice , matici $A = a^T$ vrací příkaz

```
>> A=a'.
```

Příklad 1.4 (transpozice matice)
Transponujte matici c z příkladu 1.3 `>> f=c'`
Řešení:

```
f =

    14
    10
```

- Aritmetické operace s maticemi - souhrn . Matice můžeme v Matlabu: násobit `*`, dělit zprava `/`, dělit zleva !, sčítat `+`, odečítat `−`, umocňovat `^`, transponovat `'`, násobit prvky na stejných pozicích dvou matic mezi sebou `.*`.

- Determinant matice , `det(A)` vrací determinant matice A, za předpokladu, že je matice A čtvercová.

- Délka vektoru.

```
length(a)
```

  vrací délku vektoru a

- Dimense matice,

```
[m,n]=size(A)
```

  vrací v proměnných m,n dimensi matice A

1.1.3 Polynomy

Nechť n je přirozené číslo a nechť $a_0, a_1, \ldots, a_n$ jsou reálná čísla. Funkce $P(x)$, kterou lze definovat pro všechna reálná čísla x předpisem

$$P(x) = a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n = \sum_{i=0}^{n} a_i x^{n-i}$$

se nazývá polynom. Čísla $a_0, a_1, \ldots, a_n$ se nazývají koeficienty polynomu [1] $P(x)$. Kořenem polynomu

$$P(\alpha) = \sum_{i=0}^{n} a_i x^{n-i}$$

nazýváme takové (obecně komplexní) číslo α, pro něž platí

$$P(\alpha) = \sum_{i=0}^{n} a_i \alpha^{n-i} = 0.$$

Úloha vyhledání kořenů polynomu . V prostředí Matlabu lze nalézt kořeny polynomu pomocí příkazu:

```
>> roots([..])
```

- Kořeny polynomu nalezneme pomocí příkazu:

  ```
  >> rots([..])
  ```

  Při použití tohoto příkazu Matlab vrací vektor kořenů polynomu. [2]

Příklad 1.5 (roots)
Mějme polynom $x^2 - 5x + 6$, stanovte kořeny tohoto polynomu.
Řešení:

```
>> P=[1 -5 6];
>> roots(P)
>> ans =
>>       3 2
```

polynom potom můžeme zapsat: $(x-2)(x-3)$

- Známe-li kořeny polynomu, pomocí příkazu:

  ```
  >> poly([..])
  ```

  nalezneme koeficienty polynomu.

Příklad 1.6 (vyhledání koeficientů polynomu)
Vyhledejte koeficienty polynomu z příkladu 1.5.
Řešení:

[1] Indexování koeficientů je ve shodně se syntaxí Matlabu
[2] Základní věta algebry říká, že polynom n – tého stupně s reálnými koeficienty má n kořenů v komplexním oboru

```
>> poly([2 3])
>> ans =
>>        1 -5 6
```

Tím jsme se dostali zpět k našemu výchozímu polynomu $x^2 - 5x + 6$, viz. příklad 1.5

- Násobení polynomů - příkaz:

```
>> conv(p,q)
```

realizuje konvoluci, kde p, q jsou koeficieny plynomů, které chceme násobit.

Příklad 1.7 (násobení polynomů)
V Matlabu zadejte polynomy $p = x + 2$ a $q = 4x + 3$ a vzájemně je vynásobte:
Řešení:

```
>> p=[1 2]; q=[4 3];
>> conv(p,q)
>> ans =
>>        4 11 6
```

Výsledek lze zapsat: $(x+2)(4x+3) = 4x^2 + 11x + 6$

- Dělení polynomů. Polynom vzniklý dělením dvou polynomů, za předpokladu, že dělený polynom je stejného nebo výššího stupně, vrací příkaz dekonvoluce:

```
>> [r,s]=deconv(p,q)
```

kde r, s jsou koeficienty výsledného polynomu.

Příklad 1.8 (dělení polynomů)
Mějme dány polynomy: $p = x^2 - 5x + 6$ a $q = 4x + 3$. Stanovte p/q.
Řešení:

```
>> p = [2 -5 6];
>> q = [4 3];
>> [r,s]=deconv(p,q)
>> r =
>>      1  2
>> s~=
>>      0  0
```

Výsledek dělení lze zapsat: $(x+2)$.

- Hodnotu polynomu s koeficienty p v bodě (x) vrací příkaz:

```
>> z=polyval(p,x)
```

Příklad 1.9 (hodnota polynomu v bodě)
Zadejte polynom $p = x + 2$ *a vypočtěte jeho hodnotu v bodech* $x = 1, 1.1, 1.2, 1.3 \dots 2$
Řešení:

```
>> p=[1 2]; x=1:0.1:2;
>> polyval(p,x)
>> ans =
>>        3.0000 3.1000 3.2000 3.3000 3.4000 3.5000
>>        3.6000 3.7000 3.8000 3.9000 4.0000
```

- Výpočet residua.

```
>> [k,l,m]=residue(p,q)
```

Příkaz vrací nuly k , póly l a konstantu m racionální lomené funkce, kde p jsou koeficienty polynomu čitatele a q jsou koeficienty polynomu jmenovatele.
Příkaz

```
>> [p,q]=residue(k,l,m)
```

vrací koeficienty čitatele a jmenovatele racionální lomené funkce.[3]

Příklad 1.10 (rozklad racionální lomené funkce)
Rozložte racionální lomenou funkci $(x^3 - 4x^2 + 7x - 3)/(x^3 - 5x^2 + 8x - 4)$ *na parciální zlomky.*
Řešení:

```
>> p=[1 -4 7 -3]; q=[1 -5 8 -4];
>> [k,l,m]=residue(p,q);
>> k~=
>>     0 3 1
>> l =
>>     2 2 1
>> m =
>>     1
```

Výsledkem je potom rozklad na parciální zlomky:

$$\frac{x^3 - 4x^2 + 7x - 3}{x^3 - 5x^2 + 8x - 4} = 1 + \frac{3}{(x-2)^2} + \frac{1}{x-1}$$

- Derivace polynomu:

3

```
>> polyder(p)
```

vrací koeficienty derivovaného polynomu. Příkaz

```
>> polyder(p,q)
```

vrací koeficienty derivace součinu $p \times q$ a konečně příkaz

```
>> [Q,D]=polyder(p,q)
```

vrací derivaci podílu polynomů p/q

Příklad 1.11 (derivace polynomu)
Derivujte polynom $(x^3 - 4x^2 + 7x - 3)$.
Řešení:

```
>> polyder[1 -4 7 -3]
>> ans =
>>       3 -8 7
```

Výsledkem derivace je polynom $3x^2 - 8x + 7$.

Příklad 1.12 (derivace podílu polynomů)
Derivujte podíl polynomů $(x+1)/(3x+4)$
Řešení:

```
>> a = [1 1]; b = [3 4];
>> [x, y] = polyder(a,b)
>> x =
>>     1
>> y =
>>     9 24 16
```

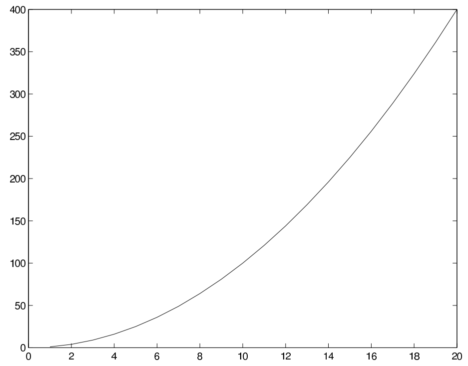
Výsledkem derivace je racionální lomená funkce:

$$\left(\frac{x+1}{3x+4}\right)' = \left(\frac{1}{9x^2+24x+16}\right)$$

1.2 Grafika v Matlabu

1.2.1 Grafy matematických funkcí

- Nejjednodušší příkaz pro vykreslení grafu je `plot(x,y)`, kde vstupem jsou dva vektory x a y stejné délky. Body (x_i, y_i) se vykreslují a jsou spojeny spojitou čarou. Není-li zadán vektor x, Matlab předpokládá, že $x(i) = i$ a pak osa x odpovídá pozici prvku x ve vektoru.

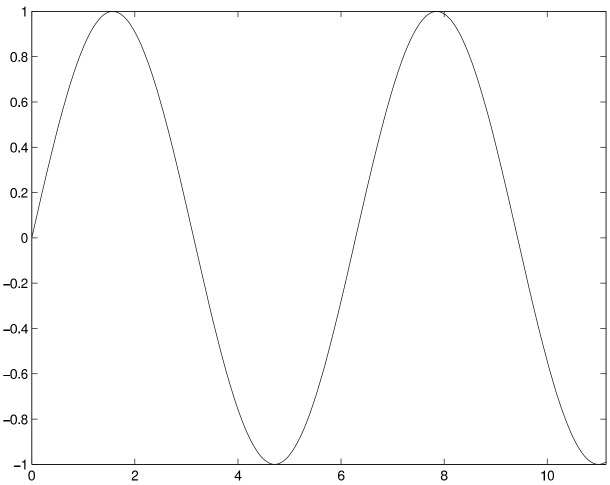


Obrázek 1.2: Aplikace příkazu `plot(x,y)`

Příklad 1.13 (plot)
Mějme vektor x =(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20) a vektor y
`>> y = x .* x`
y =(1 4 9 16 25 36 49 64 81 100 121 144 169 196 225 256 289 324 361 400). Zadáním příkazu `plot(x,y)` *dostaneme parabolu.*

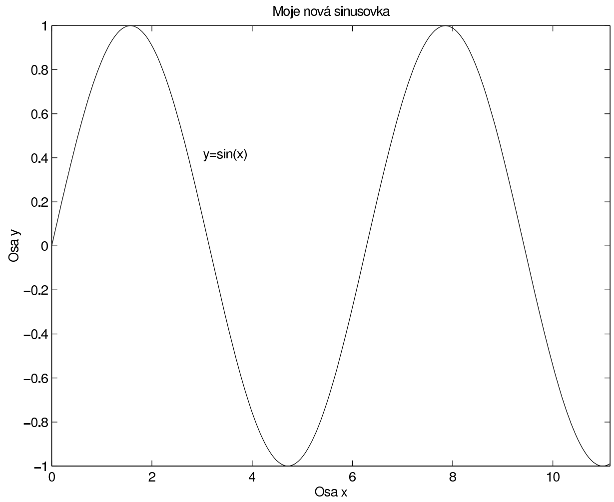
- Vykreslení grafu funkce v Matlabu: Graf lze v Matlabu vykreslit pomocí příkazu
 `fplot('funkce',[interval])`

Příklad 1.14 (fplot)
Zobrazte funkci $y = sin(x)$ *na intervalu* $< 0,8*\pi >$.
Zadejte `fplot('sin(x)',[0 8*pi])`. *Matlab vykreslí sinusovku. Obr.1.3:*



Obrázek 1.3: Sinusovka

- Popis resp. název grafu lze provést pomocí příkazu: `title('Moje nová sinusovka')`. příkazu. Příkazem `xlabel('Osa x')` popíšete osu *x* a příkazem `ylabel('Osa y')` popíšete osu *y*. Chceme-li pro přehlednost grafu doplnit titulek,nebo kreslíme více funkcí do jednoho grafu, použijeme příkaz,
 `gtext('y=sin(x)')`
 který způsobí, že po vykreslení grafu máme možnost umístit do obrázku vhodný text. Viz. obrázek 1.4

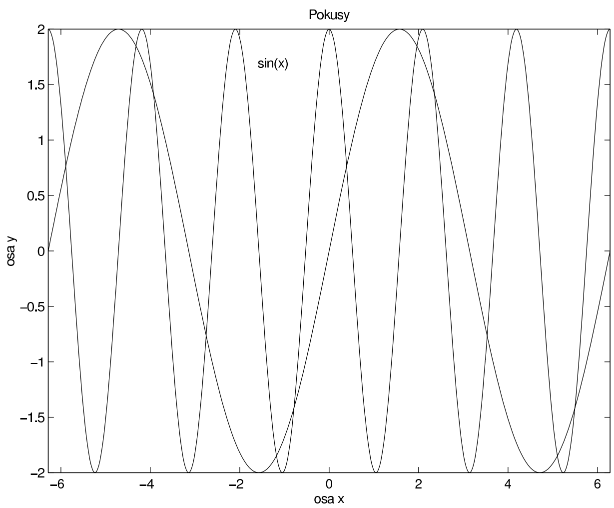


Obrázek 1.4: Sinusovka s popisem

- Zkusme do jednoho obrázku vykreslit průběh funkce $y = sin(x), y = cos(x), y = 2sin(x)$ a $y = 2cos(3x)$ v intervalu $< -2\pi, 2\pi >$, zkusme obrázek nadepsat, popsat osy a popsat i nakreslené průběhy funkcí. Posloupnost příkazů může vypadat takto:

```
>> fplot('sin(x)',[-2*pi 2*pi])
>> hold on% příkaz hold způsobí,
          % že se vám pro další graf neotevře nové okno,
          % graf se vykreslí do již nakresleného grafu.
>> fplot('cos(x)',[-2*pi 2*pi])
>> fplot('2*sin(x)',[-2*pi 2*pi])
>> 2*cos(3*x)',[-2*pi 2*pi])
>> title('Pokusy')
>> xlabel('osa x')
>> ylabel('osa y')
>> gtext('sin(x)')
```

Grafy na obrázku 1.5 jsou přinejmenším nepřehledné.



Obrázek 1.5: Grafy vykreslené do jednoho obrázku

Pro řešení této situace použijte příkaz pro vykreslení subgrafů :

```
>> subplot(2,2,1),% kde (2,2,1) definuje plochu 2 x 2 pro vykreslení grafů
                  % 1 je 1. graf v pořadí
   fplot('sin(x)',[-2*pi 2*pi])
>> title('sin(x)')
>> xlabel('x')
>> ylabel('y')
>> gtext('sin(x)')
>> subplot(2,2,2), fplot('cos(x)',[-2*pi 2*pi])
>> title('cos(x)')
>> xlabel('x')
>> ylabel('y')
>> gtext('cos(x)')
>> subplot(2,2,3), fplot('2*sin(x)',[-2*pi 2*pi])
>> title('2sin(x)')
>> xlabel('x')
>> ylabel('y')
>> gtext('2sin(x)')
>> subplot(2,2,4), fplot('2*cos(3*x)',[-2*pi 2*pi])
>> title('2cos(3x)')
>> xlabel('x')
>> ylabel('y')
>> gtext('2cos(3x)')
```

Celkový výsledek vidíte na obrázku číslo 1.6.

1.2.2 Další grafy 2D

- Čárový graf viz. obrázek 1.7

```
>> x=0:0.05:5
>> y=sin(x^2)
>> plot(x,y)
```

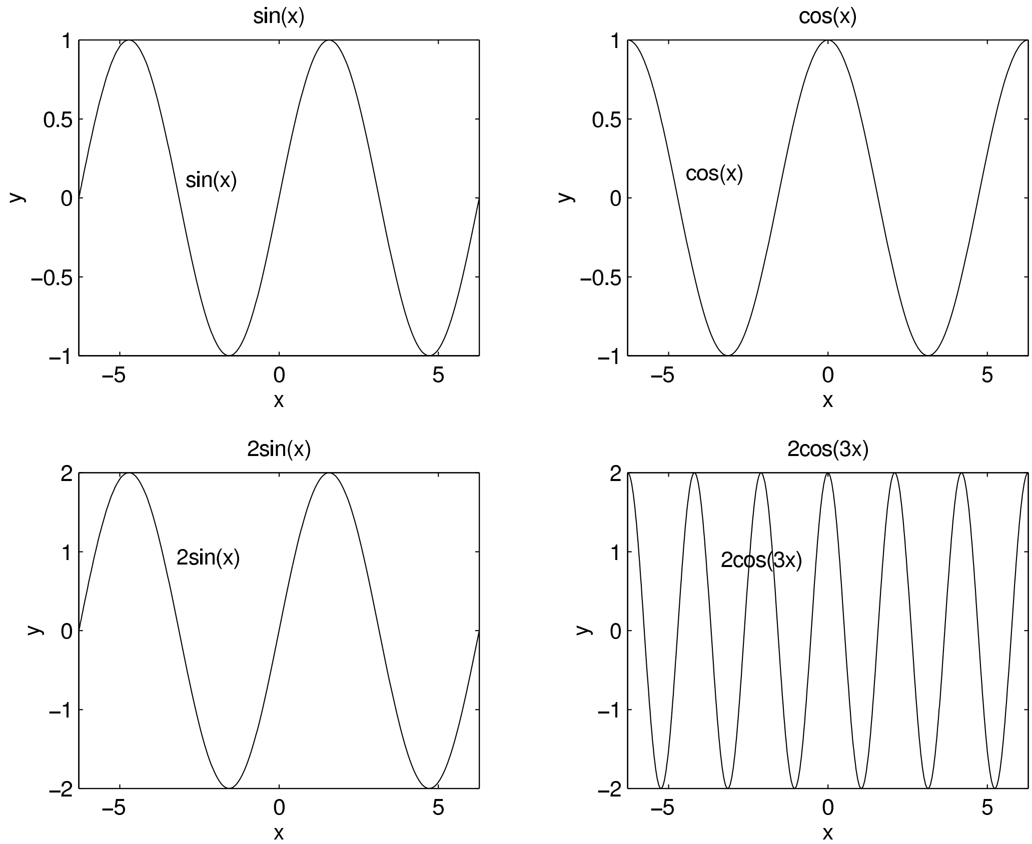
- Sloupcový graf viz. obrázek 1.8

```
>> x = -2.9:0.2:2.9
>> bar(x,exp(-x.*x))
```

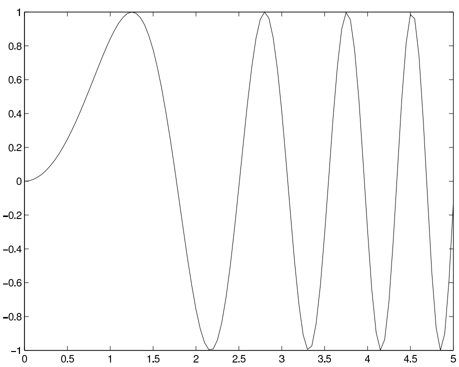
- Stupňový graf viz. obrázek 1.9

```
>> x=0:0.25:10
>> stairs(x,sin(x))
```

- Graf v polárních souřadnicích viz. obrázek 1.10



Obrázek 1.6: Vaše grafy, teď již přehledněji



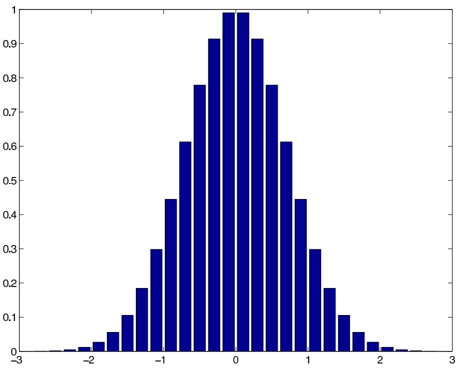
Obrázek 1.7: Příklad čárového grafu

```
>> t=0:.01:2*pi
>> polar(t,abs(sin(2*t)*cos(2*t)))
```

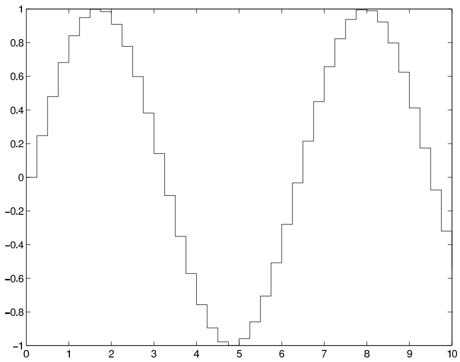
- Graf diskrétního signálu viz. obrázek 1.11

```
>>x = 0:0.1:4
>>y = sin(x^2)*exp(-x)
>>stem(x,y)
```

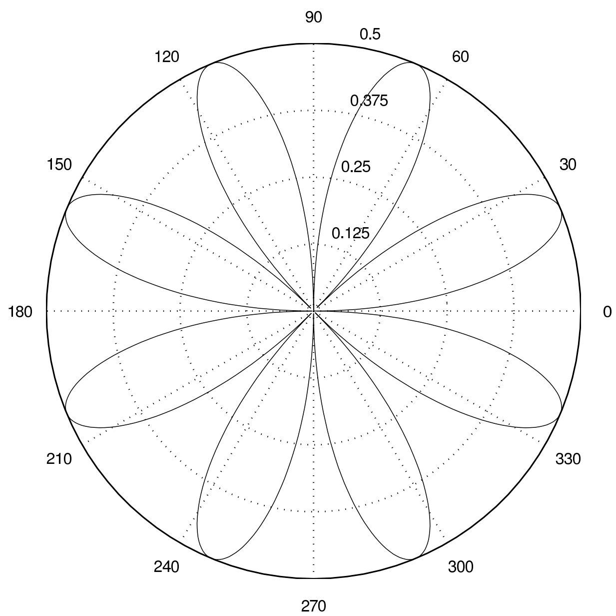
V dalším se budeme věnovat i nadále grafickému zobrazení funkcí tentokrát ve 3D.



Obrázek 1.8: Příklad sloupcového grafu



Obrázek 1.9: Příklad schodového grafu

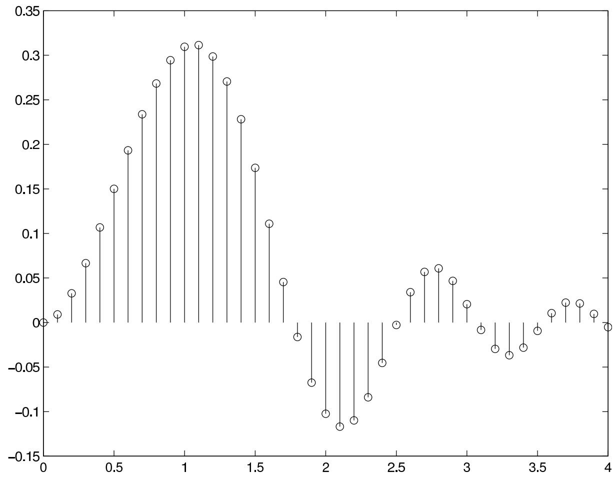


Obrázek 1.10: Příklad grafu v polárních souřadnicích

1.2.3 3D Grafy

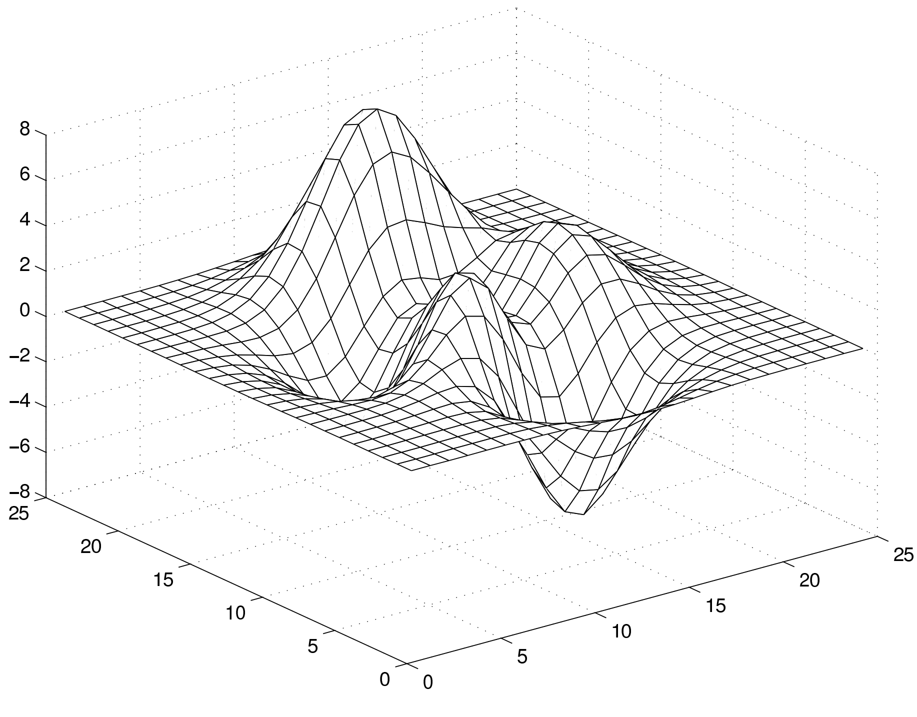
- Zobrazení plochy, obrázek 1.12

```
>> z=peaks(25)
```



Obrázek 1.11: Příklad grafu diskrétního signálu

```
>> mesh(z)
```

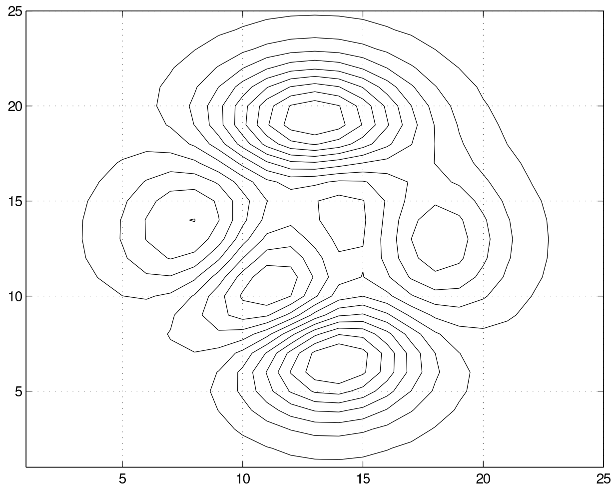


Obrázek 1.12: Příklad 3D grafu plochy

- Zobrazení vrstevnic, obrázek 1.13

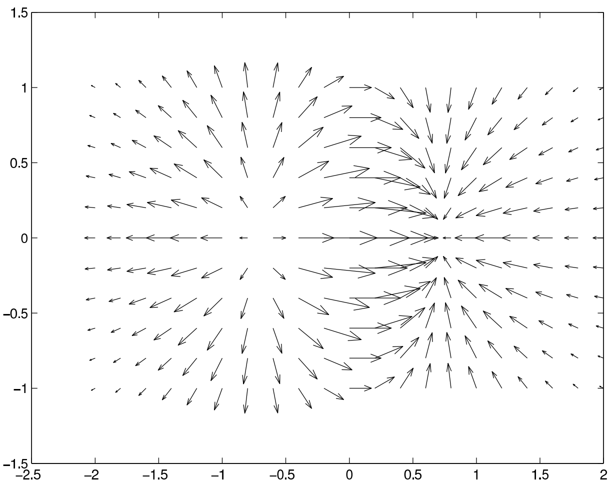
```
>> z=peaks(25)
>> contour(z,16)
```

- Zobrazení silového pole, obrázek 1.14



Obrázek 1.13: Příklad grafu vrstevnic

```
>> x = -2:.2:2; y = -1:.2:1
>> [xx,yy] = meshgrid(x,y)
>> zz = xx.*exp(-xx^-yy.^2)
>> [px,py] = gradient(zz,.2,.2)
>> quiver(x,y,px,py,2)
```



Obrázek 1.14: Příklad grafu silového pole

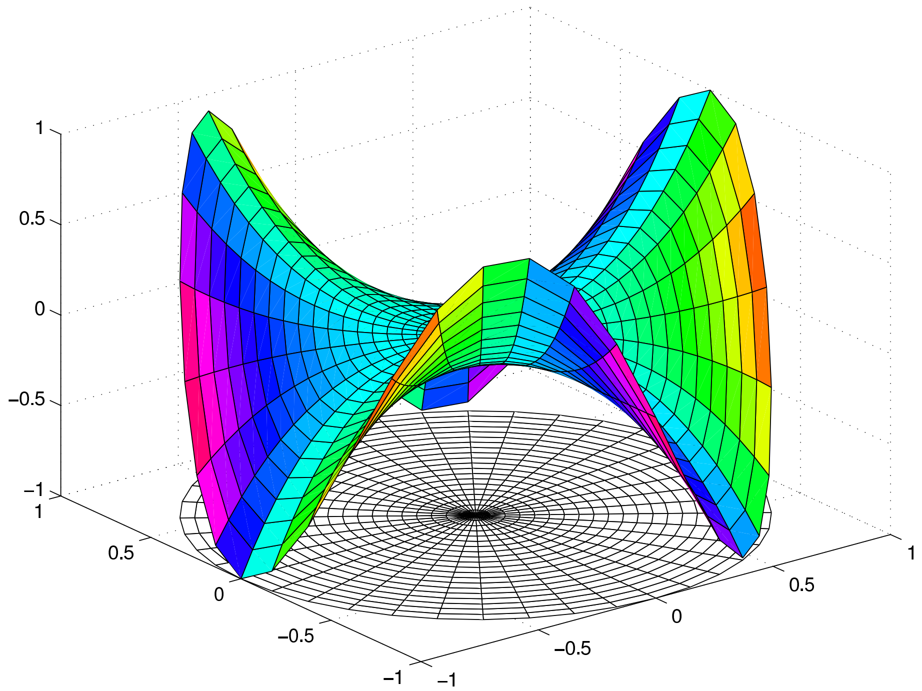
- Posloupnost příkazů k zobrazení povrchu.

```
>> z=peaks(25)
>> surfl(z)
>> shading interp
>> olormap(pink)
```

1.2.4 Zobrazení komplexních funkcí komplexní proměnné

- Komplexní graf funkce, obrázek 1.15 $f(z) = z^3$

```
>> z=cplxgrid(20)
>> cplxmap(z,z^3)
```



Obrázek 1.15: Příklad grafu komplexní funkce $f(z) = z^3$

- Kompexní graf funkce, obrázek 1.16 $f(z) = (z^4 - 1)^{(1/4)}$

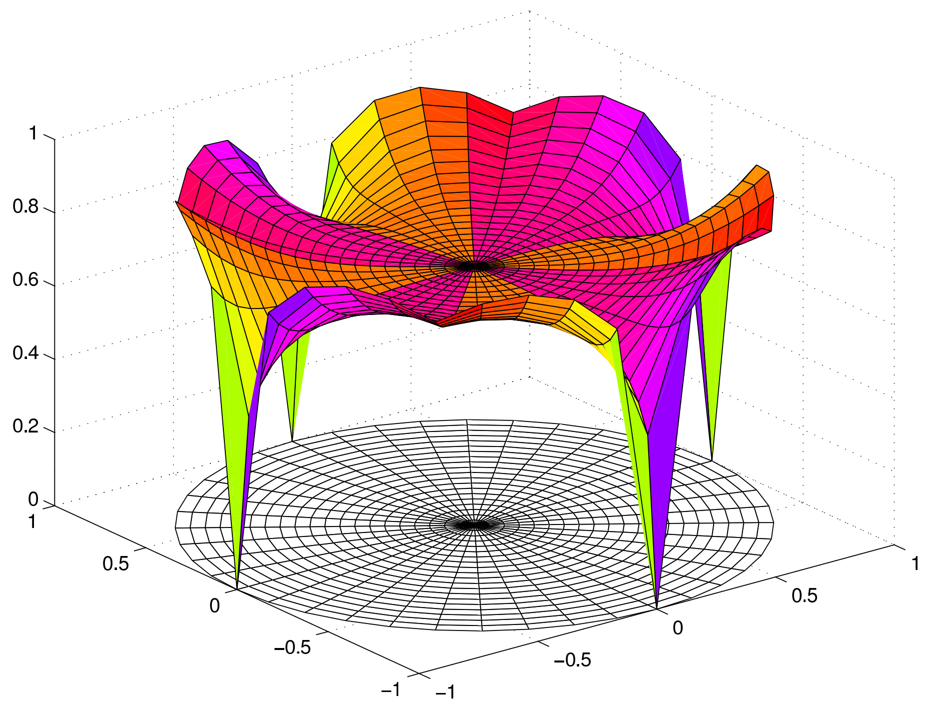
```
>> z=cplxgrid(20)
>> cplxmap(z,(z^4-1)^(1/4))
```

1.3 Něco více o Matlabu

1.3.1 Vstup a výstup dat v Matlabu

Až doposud jsme zadávali data do Matlabu prostřednictvím klávesnice a to tak, že jsme vypsali příkazový řádek. Později ovšem, to až si budeme psát vlastní příkazy a funkce resp. M-fily, přijdeme na to, že potřebujeme zadávat data v dialogu prostřednictvím klávesnice. Pro tyto účely je v Matlabu definována funkce `input` . Chceme-li si přečíst číslo z klávesnice zadáme příkaz `>> x=input(t)`, kde t je řetězec, který slouží coby prompt při čtení žádané hodnoty. Např. `>> V=input('počet vlků: ')`.

Chceme-li přečíst řetězec napište `>> retezec=input(t,'retezec')`. Formát výstupu lze nastavit příkazem `format short`, `format long`, `format hex`, další nastavení formátu výstupu nalezneme v helpu: `help format`. Chceme-li si uložit svoje data použijeme příkaz `save`, Matlab uloží naše data do souboru `matlab.mat`, samozřejmě si můžeme naše data uložit do našeho souboru: `>> save nassoubor`. Příkazem `>> load nassoubor` můžeme zase naše data načíst.



Obrázek 1.16: Příklad grafu komplexní funkce $f(z) = (z^4 - 1)^{(1/4)}$

1.3.2 M – files

Všechny naše pokusy můžeme uložit do tzv. M-files . M-file jsou skripty tj. textové soubory s příponou `.M`, které nám umožňují provést posloupnost příkazů najednou, aniž bychom je museli zadávat po jednom do příkazové řádky z klávesnice. Do editoru M-files se dostaneme v Matlabu přes menu `File` → `New` → `M-file`

1.3.3 Řídicí příkazy a funkce

Matlab nám umožňuje psát vlastní příkazy a funkce čímž se stává z maticového kakulátoru programovacím jazykem. Základem pro tvoření algoritmů jsou cykly FOR, WHILE a příkaz větvení IF-ELSE.

Cyklus FOR má syntaxi:

```
for n=vektor
    příkazy
end
```

kde `n` je řídící proměnná tj. určuje, kolikrát se bude celý cyklus opakovat, a při každém průchodu smyčkou se jí přiřadí hodnota příslušného prvku.

- Nyní si ukážeme jak vytvořit cyklus FOR. Pokus proveďte v Matlabu v okně `File` → New → M-file

```
%Pokusný M-file - cyklus FOR
n=input('Zadejte dimensi matice:');
for i=1:n
    for j=1:n
        A(i,j)=j+(i-1)*n
    end
```

```
  end;
  A˜
```

Uložíte-li si M-file do adresáře, na který máme nastavenou cestu (cestu do aktuálního adresáře nastavíme v menu: `File` → `Set Path..`, a potom napíšeme jeho název do příkazové řádky, Matlab nám vypíše: `Zadejte dimensi matice:` Zadejme dimensi 4 a Matlab nám vrátí matici:

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix}$$

1.3.4 Cykly v Matlabu

- Dalším cyklem je WHILE . Opět si uvedeme nejprve příklad:

```
%Pokus se smyčkou WHILE
clear;
k=input('Zadejte dimensi matice:');
i=1;
n=1;
while n<k^2
    while i<=k
        j=1;
        while j<=k
            A(i,j)=n;
            j=j+1;
            n=n+1;
        end
        i=i+1;
    end
end
A˜
```

Matlab vám vypíše: `Zadejte dimensi matice:` Zadejte dimensi 3 a Matlab vám vrátí matici:

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Příkazy uvnitř smyčky WHILE se provádějí pokud je splněna logická podmínka. Syntaxe smyčky je:

```
while logická podmínka
      příkazy
end
```

- Dalším příkazem je příkaz větvení. Syntaxe příkazu IF - ELSE:

```
if podmínka1
   příkazy
elseif podmínka2
   příkazy
else
   příkazy
end
```

1.3.5 Relační operátory v Matlabu

$<$	menší než
$<=$	menší nebo rovno
$>$	větší
$>=$	větší nebo rovno
$==$	rovná se
$=$	nerovná se

Tabulka 1.1: Tabulka relačních operátorů

1.3.6 Logické operátory

&	AND
|	OR
~	NOT

Tabulka 1.2: Tabulka logických operátorů

1.3.7 Elementární matematické funkce v Matlabu

1.3.8 Funkce v Matlabu

V Matlabu máme mimo tvorby vlastních M-files také možnost tvorby vlastních funkcí. Funkce se liší od M-file tím, že má vstupní a výstupní argumenty a výstupní argumenty, a že proměnné vytvořené uvnitř funkce jsou lokální. Definujeme-li si novou funkci, postupujeme stejně, jakoby jsme si psali nový M-file, ale do záhlaví napíšeme deklaraci ve tvaru:

```
function vystupni_argument = jmeno_funkce(vstup1, vstup2,...)
```
nebo
```
function[vystup1, vystup2,...]=jmeno_funkce(vstup1, vstup2...)
```

Příklad 1.15 (faktoriál)
Definujte funkci, která vrací faktoriál tj. n!

`abs`	absolutní hodnota
`angle`	fázový úhel
`sqrt`	druhá odmocnina
`real`	reálná část komplexního čísla
`imag`	imaginární část komplexního čísla
`conj`	komplexně sdružená matice
`round`	zaokrouhlení k nejbližšímu celému číslu
`fix`	zaokrouhlení směrem k 0
`floor`	zaokrouhlení směrem k $-\infty$
`ceil`	zaokrouhlení směrem k ∞
`sign`	funkce signum
`rem`	zbytek
`sin`	sinus
`cos`	cosinus
`tan`	tangens
`asin`	arcus sinus
`acos`	arcus cosinus
`atan`	arcus tangens
`atan2`	4 kvadrantový arcus tangens
`sinh`	hyperbolický sinus
`cosh`	hyperbolický cosinus
`tanh`	hyperbolický tangens
`exp`	exponenciála základu e
`log`	přirozený logaritmus
`log10`	logaritmus o základu 10
`bessel`	Besselova funkce
`gamma`	gamma funkce
`rat`	racionální aproximace

Tabulka 1.3: Tabulka elementárních matematických funkcí v Matlabu

Řešení:

```
function y=fakt(n)
%Funkce faktorial, vraci faktorial, zadavejte cela kladna cisla
%pokusna funkce pro pripad vyuky MSP :-)
%syntaxe: fakt(n)
i=1; fakt=1;
while i<n+1
      fakt=fakt*i;
      i=i+1;
end
y=fakt
```

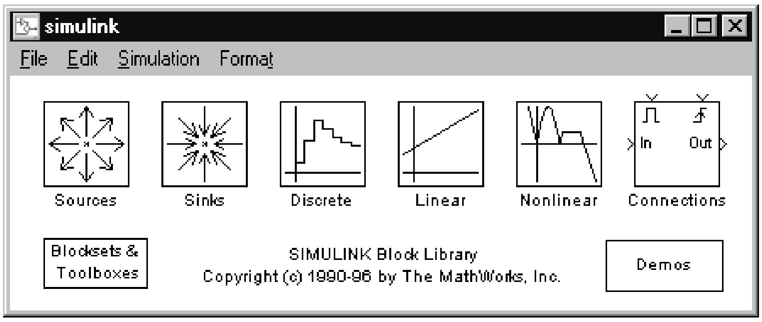
Všimněme si, že po deklaraci funkce následuje komentář. Je to přesně ten text, který se nám zobrazí v okně Matlabu, napíšeme-li `>> help fakt`

```
Funkce faktorial, vraci faktorial, zadavejte cela kladna cisla,
pokusna funkce pro pripad vyuky MSP :-) syntaxe: fakt(n)
```

Kapitola 2

Simulink

Nezbytnou součástí našeho modelování systémů jsou modely sestavené v Simulinku, který je součástí Matlabu. V dalším se budeme věnovat problematice sestavení modelu v Simulinku, dozvíme se, jaké bloky Simulink obsahuje, dozvíme se, co dané bloky znamenají co je vstupem bloků a co výstupem, se jak se dají některé z bloků nastavit.

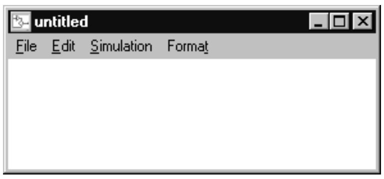


Obrázek 2.1: Okno Simulinku v Matlabu

Simulink spustíme z příkazové řádky jednoduchým příkazem:

```
>> simulink
```

otevře se nám okno viz. obrázek 2.1 a okno pro kreslení modelu, obrázek 2.2

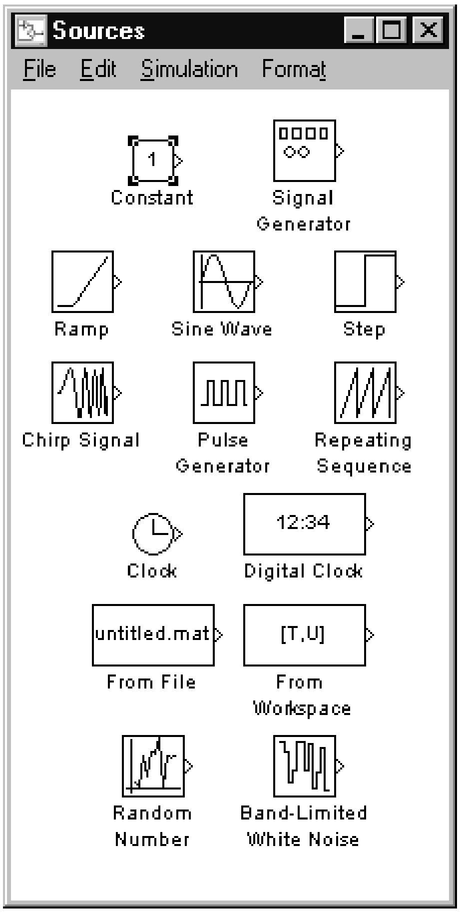


Obrázek 2.2: Okno pro kreslení modelu v Simulinku

2.1 Blok Sources

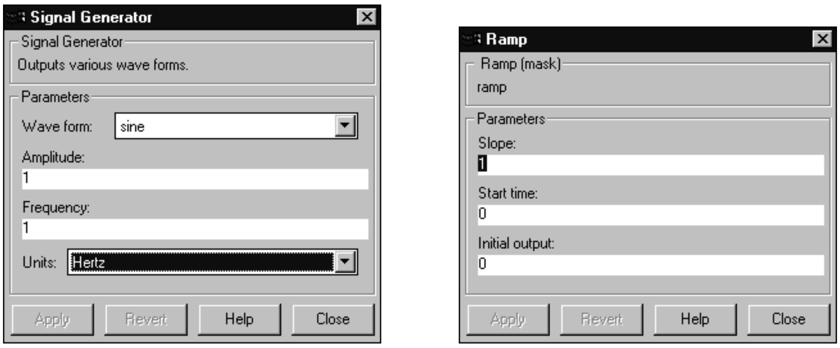
Vezměme teď jednotlivé druhy bloků postupně. První skupinou bloků je skupina `Sources` - zdroje, obrázek 2.3.

- Prvním v řadě je blok `Constant` - konstanta. Po rozkliknutí bloku se vám otevře okno viz. obrázek: **??** ve kterém máte možnost zadat hodnotu konstanty.

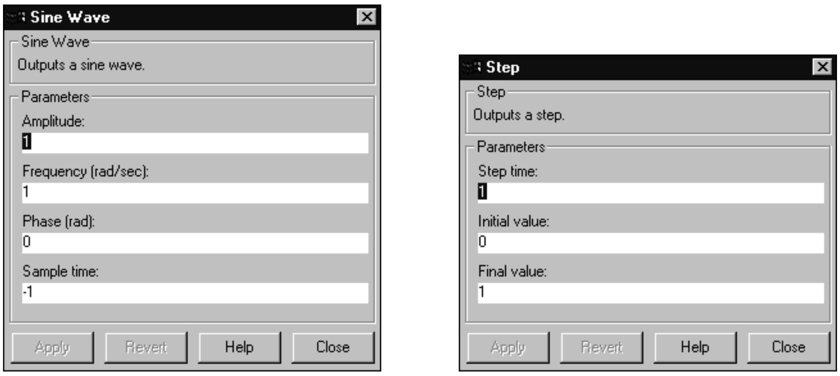


Obrázek 2.3: Okno `sources` v Simulinku

- Dalším blokem je blok `Signal Generator`, po otevření bloku máte možnost definovat generovaný signál: průběh, amplitudu a kmitočet, viz. obrázek: **??**.
- Blok `Ramp` - náběžná hrana, umožňuje nastavení sklonu, hodnoty a času náběhu, obrázek: **??**.
- Blok `Sinwave` - generátor sinusového signálu, umožňuje nastavení amplitudy, kmitočtu a fáze, obrázek: **??**.
- Blok `Step` je blokem jednotkového skoku, lze nastavit: čas skoku, počáteční hodnota, konečná hodnota. Viz. obrázek **??**
- Blok `Chirp` je zdrojem sinového průběhu o počátečním kmitočtu, jehož hodnota se zvyšuje po stanovenou dobu ke kmitočtu konečnému. V okně, obrázek **??** lze nastavit: počáteční kmitočet, konečný kmitočet a dobu změny kmitočtu.
- Blok `Pulse` je zdrojem obdélníkového průběhu. Po rozkliknutí se dá v okně (obrázek: **??**) nastavit: perioda, procento obdélníku z periody, amplituda a čas.
- Blok `Repeating Sequence`, je zdrojem pilového průběhu. V okně nastavení zadáváme čas a výstupní hodnotu, obrázek: **??**.
- Blok `Clock` hodin a `Digital Clock` digitálních hodin, je zdrojem časových impulsů, v okně nastavení, (obrázek: **??**), můžeme nastavit vzorkovací periodu.

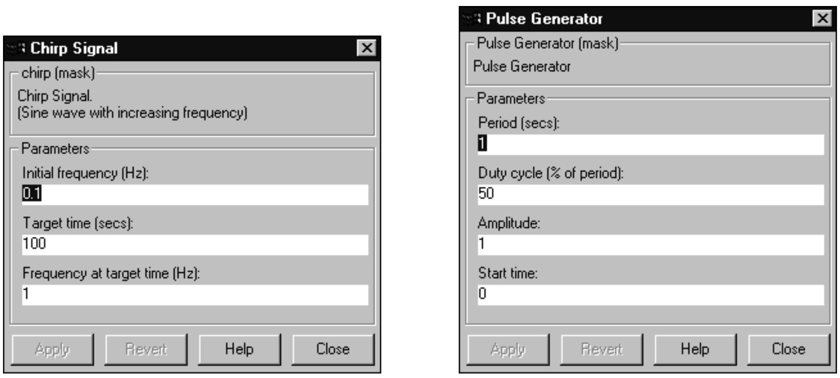


Obrázek 2.4: Okno nastavení bloků `Signal Generator` a `Ramp`

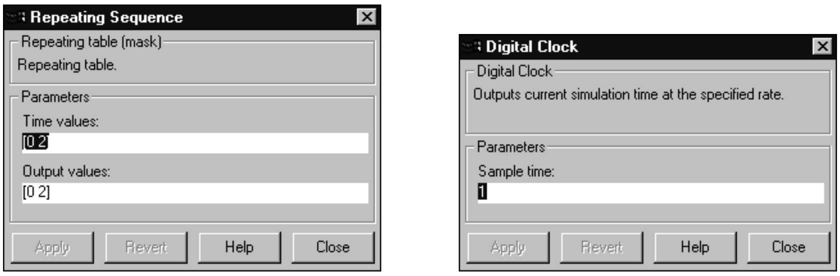


Obrázek 2.5: Okno nastavení bloků `SinWave` a `Step`

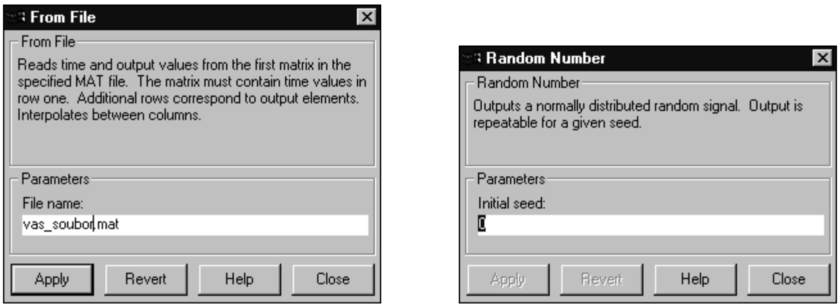
- Blok `From File` nám umožní použít jako zdroj soubor. Možnosti okna nastavení jsou zřejmé z obrázku: **??**
- Okno `Random Number` nám poslouží jako zdroj normálně rozloženého náhodného signálu. Nastavit můžeme počáteční rozsah, obrázek **??**.
- Blok `Band-Limited White Noise` je zdrojem bílého šumu s omezenou šířkou pásma. V okně nastavení lze zadat: výkonovou hustotu, vzorkovací periodu a rozsah, viz. obrázek **??**.



Obrázek 2.6: Okno nastavení bloků `chirp` a `pulse`



Obrázek 2.7: Okno nastavení bloků `Repeating Sequence` a `Digital Clock`



Obrázek 2.8: Okno nastavení bloků `Random Number` a `From File`

2.2 Blok Sinks

Další skupinou bloků v Simulinku jsou bloky `Sinks`, bloky výstupu. Slouží k zobrazení dat v modelu. Viz. obrázek `sinks`

- Prvním blokem v řadě je blok `Scope`, je to vlastně osciloskop, kterým můžeme sledovat časový průběh měřené veličiny. Než se podíváme, jak takový výstup z bloku `Scope` vypadá zkusme si sestavit v okně Simulinku zdroj sinusového signálu a připojme k němu osciloskop. Provedeme to velmi jednoduše, přetáhneme ze skupiny `Sources` blok `Sinwave` do vašeho pokusného Simulinkového okna, potom stejným postupem přeneseme i blok `Scope`. Oba bloky spojíme pomocí myši, obrázek: 2.10.

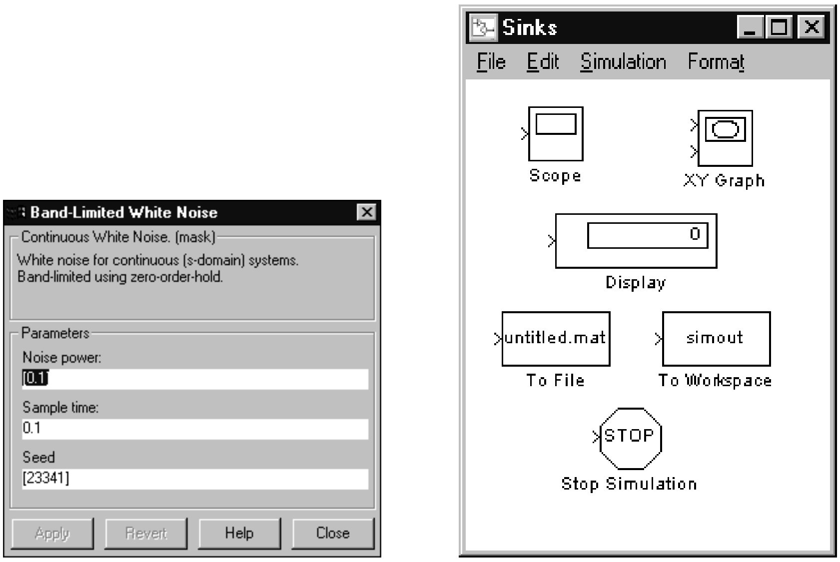
  Rozkliknutím bloku `Sinwave` můžeme provést nastavení. Simulaci spustíme v menu okna ve kterém sestavujeme model `Simulation` → `Start`. Otevřením bloku `Scope`, obrázek 2.11se můžeme podívat na sinusovku.

- Dalším blokem v řadě `Sinks` je blok `XY Graph`, který umožňuje sledovat dva kanály. V okně nastavení, obrázek **??**, můžete definovat rozsahy a periodu vzorkování. V okně Simulinku sestavíme další model.

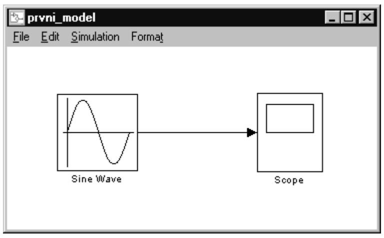
  Příklad 2.1 (lissajousovy křivky)
  Přiveďte na kanál `X` *osciloskopu signál z generátoru sinusového průběhu a na kanál* `Y` *signál z generátoru sinusového průběhu. Pro kanál* `X` *nastavte generátor na kmitočet 1 Hz, amplitudu 1 a fázi 0. Druhý generátor, připojený na kanál* `Y`*, nastavme na kmitočet 1 Hz a fázi* $\pi/2$ *a simulujte. Schéma v simulinku je na obrázku: 2.12.*

  Po spuštění simulace a otevření bloku `XY Graph` *se můžeme podívat, jak vypadá tzv. Lissajousův obrazec , obrázek* **??**.

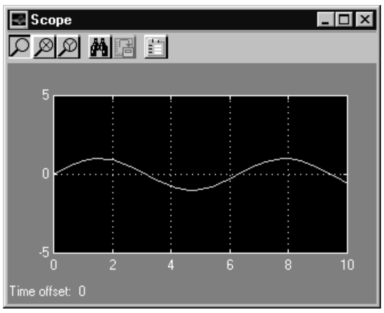
  Zkusme měnit fázi generátoru a sledujme, jak se mění výstup v okně `XY Graph`*. Jsou-li fáze shodné, dostaneme obrázek* **??**



Obrázek 2.9: Okno nastavení bloků `Band-Limited White Noise` a `Sinks`



Obrázek 2.10: První pokus v Simulinku

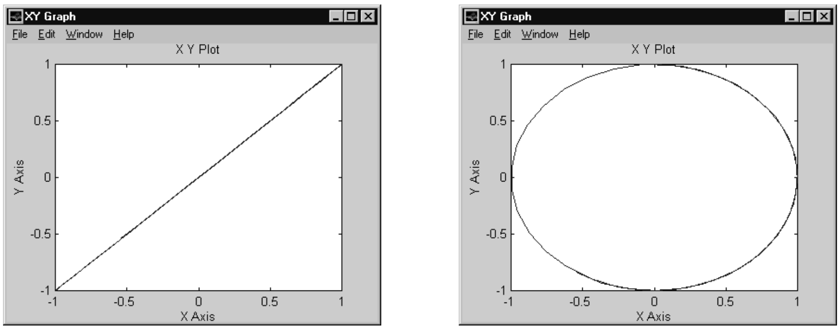


Obrázek 2.11: Otevřený blok `Scope`

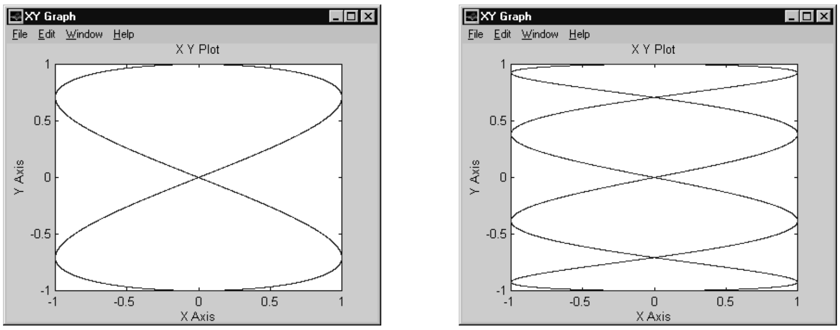
Změníme-li kmitočet jednoho z generátorů dostaneme na výstupu obrazce viz. obrázek **??** **??***. Tato metoda se dá, použít pro porovnání sinuového průběhu neznámého kmitočtu pomocí měřícho generátoru.*

- Blok `Display`nám zobrazí výsledek na displeji. Chceme-li si uložit výsledek naší simulace, použijeme blok `To File`, po otevření tohoto okna můžete definovat soubor, do kterého se nám budou ukládat výstupní data. Nastavení si prohlédněte na obrázku **??**.
- Pomocí bloku `To Workspace` si můžeme uložit výstupní data do pracovní plochy Matlabu. Okno nastavení bloku `To Workspace` je na obrázku **??**.
- Zavedeme-li výstup do bloku `Stop Simulation`, dojde k zastavení simulace.

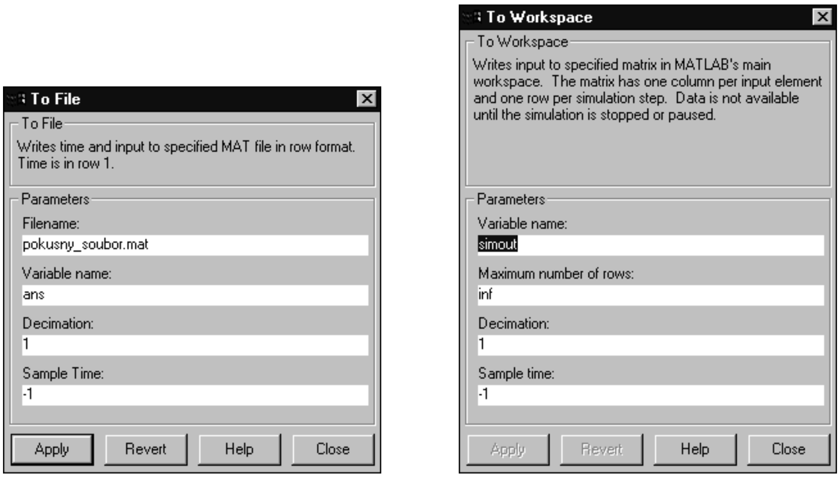
Obrázek 2.12: Model Lissajousových obrazců v Simulinku



Obrázek 2.13: Model v Simulinku a výsledek – Lissajousovy obrazce



Obrázek 2.14: Model v Simulinku a výsledek – Lissajousovy obrazce



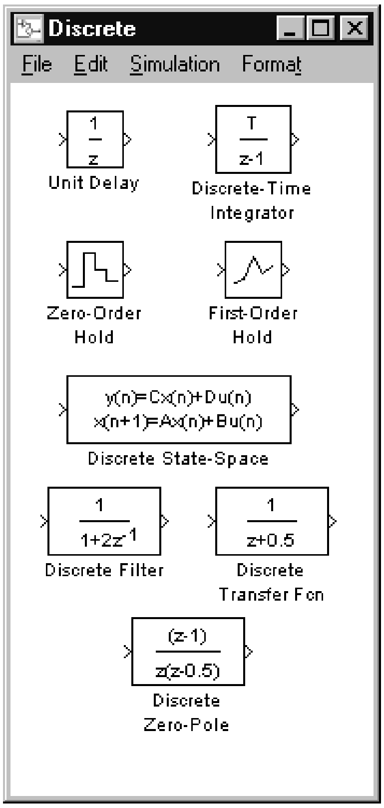
Obrázek 2.15: Okno nastavení bloků `To File` a `To Workspace`

2.3 Blok Discrete

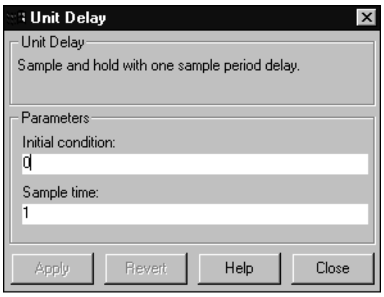
Další skupinou bloků v Simulinku je skupina bloků `Discrete`, obrázek 2.16, které umožňují pracovat s diskrétními veličinami.

- Prvním z řady je blok `Unit Delay`, jednotkové zpoždění. Otevřením okna se dostaneme do nabídky nastavení, obrázek 2.17, kde můžeme nastavit dobu zpoždění signálu a vzorkovací periodu.
- Dalším z řady bloků, které budeme používat je blok `Discrete State-Space`, který nám umožní

zadat diskrétní systém přímo pomocí stavových matic **A, B, C, D,** dále je nutno zadat počáteční podmínky a vzorkovací periodu. Okno pro zadání parametrů systému je na obrázku 2.18.



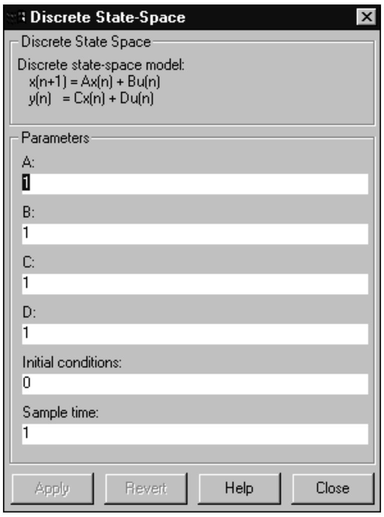
Obrázek 2.16: Okno bloků skupiny `Discrete`



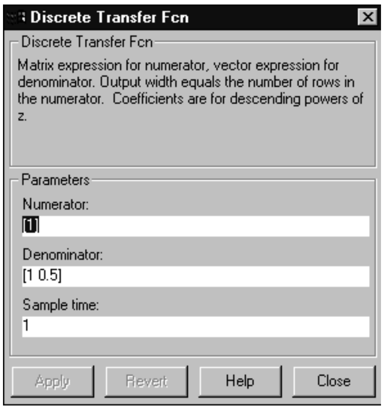
Obrázek 2.17: Okno nastavení bloku `Unit Delay`

- Další blok `Discrete Transfer Fcn` použijeme pro zadání přenosové funkce diskrétního systému , čitatele a jmenovatele přenosové funkce lze zadat po otevření bloku v okně nastavení, viz obrázek 2.19, kde `numerator` a `denominator` jsou vektory čitatele a jmenovatele přenosové funkce v kladných mocninách z.

- Další možností jak v Simulinku definovat diskrétní systém, je jeho zápis do bloku `Discrete Zero-Pole`, kde po otevření okna nastavení můžeme definovat vektory nul (`Zeros`), pólů (`Poles`), zesílení(násobení konstantou) a vzorkovací periodu.



Obrázek 2.18: Okno nastavení bloku `Discrete State-Space`

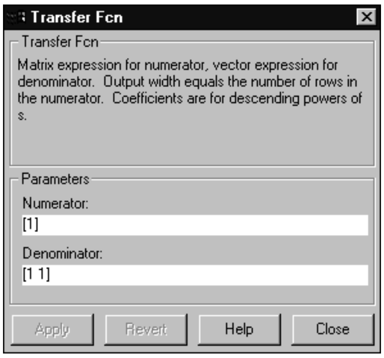


Obrázek 2.19: Okno nastavení bloku `Discrete Transfer Fcn`

2.4 Blok Linear

Množina bloků `Linear`, obrázek **??**, umožňuje sestavení modelu ve spojité oblasti.

- Prvním blokem je blok `Gain`, který provádí násobení konstantou. Potřebnou konstantu lze zadat po otevření bloku v okně nastavení.
- Blok `Sum`, sumátor nám umožní sečíst (odečíst) námi definovaný počet vstupů.
- Blok `Integrator` je důležitým blokem při sestavování modelů spojitých systémů, po otevření okna nastavení lze definovat počáteční podmínky.
- Blok `Transfer Fcn` umožňuje nasimulovat spojitý systém pomocí čitatele a jmenovatele přenosové funkce spojitého systému.Do pole `Numerator` zadejme koeficienty polynomu čitatele a do pole `Denominator` koeficienty polynomu jmenovatele. Viz. obrázek 2.20.
- Blok `State-Space` umožňuje zadat spojitý systém přímo pomocí stavových matic **A, B, C, D**. Do okna nastaveni je nutno vložit také počáteční podmínky. Okno pro zadání parametrů systému je na obrázku **??**. Pomocí bloku `Zero-Pole` lze zadat systém, podobně jako tomu bylo u systému diskrétního, pomocí pólů a nul.
- Konečně blok `Dot Product` vrací skalární součin.

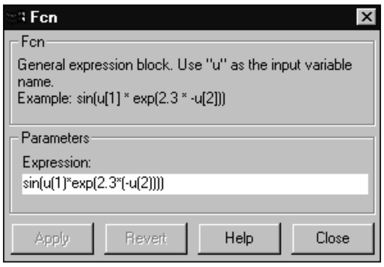


Obrázek 2.20: Okno nastavení bloku `Transfer Fcn`

2.5 Blok Nonlinear

Z množiny bloků `Nonlinear`, vybereme blok

- `Fcn`, který nám umožní definovat si vlastní funkci, obrázek 2.21.



Obrázek 2.21: Okno nastavení vlastní funkce `Fcn`

- Podobným blokem je blok `Elementary Math`, který nám umožní provádět různé základní matematické funkce (trigonometrické, exponenciální a hyperbolické). Požadovanou funkci si vybereme v okně nastavení.

  Další bloky v této skupině umožňují např. logické operace:

- `Logical Operator`, relační operace,
- blok `Relational Operator`, funkci absolutní hodnoty,
- blok `Abs`, blok `Sign` vrací funkci signum.
- Blok `Switch` nám umožní přepínat vstupní signály 1 a 3 podle definované hodnoty řídícího signálu 2. , kterou zadáme v okně nastavení na výstup. Má-li řídící signál na vstupu 2. hodnotu větší nebo rovnu zadané hodnotě je přepínač v poloze 1., jinak se přepne do polohy 3.

2.6 Bloky Connections

Další, neméně důležitou, skupinou bloků je skupina `Connections` - spojení. V našich modelech budeme zpravidla používat bloky:

- `Mux` - multiplexor(blok se dvěma a více vstupy a jedním výstupem, postupně periodicky přepíná jednotlivé vstupy na výstup) a

- `Demux` - demultiplexor (blok, který postupně periodicky přepíná vstup na dva a více výstupů). Po otevření okna nastavení lze u bloků nastavit počet vstupů resp. výstupů.

2.7 Bloky Blocksets and Toolboxes

Poslední množinou bloků jsou bloky `Blocksets and Toolboxes` a v nich skupina

- `Simulink Extras`. Skupina `Additional Sinks` obsahuje spektrální analyzatory. Viz obrázek **??** které vrací kmitočtové charakteristiky daných signálů.

Kapitola 3

Modelování sytémů v Simulinku

V předchozí kapitole jsme se seznámili se základními bloky v Simulinku. Zde si ukážeme, jak lze sestavit jednoduché modely diferenciálních rovnic, diferenčních rovnic a jejich aplikace na praktických příkladech.

Chceme-li sestavit nový model, otevřeme v Simulinku nové okno. Bloky, potřebné pro sestavení modelu přesuneme myší nebo zkopírujeme z příslušné skupiny. Potřebujeme-li blok otočit, označíme ho a použijeme klávesovou zkratku <CTRL>+<R>. Bloky se spojují kliknutím na výstup spojovaného bloku a tažením ke vstupu bloku následujícího. Po označení lze se spojnicí manipulovat, resp. ji vymazat.

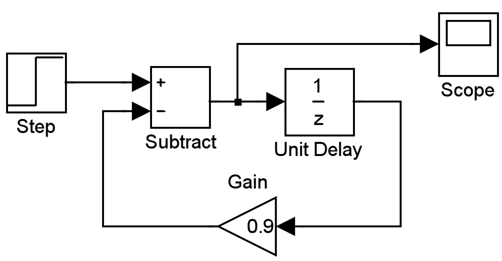
3.1 Modelování diskrétních systémů

Podstatou modelování diskrétních systémů je blok Unit delay, který zpožďuje vstupní signál o jeden časový interval. Pokud na jeho vstup je připojen signál $y(n)$, na jeho výstupu bude signál $y(n-1)$ s tím, že do bloku Unit delay lze zapsat počáteční podmínku $y(1)$. Modelování spočívá ve vyjádření proměnné $y(n)$ jako funkce proměnných $y(n-1),\dots y(n-m)$ a $u(n),\dots u(n-m)$, kde m je řád systému. Zpožděné vstupy a výstupy lze realizovat skládáním bloků Unit delay. Zpožděné vstupy a výstupy plus aktuální vstup $f(u(n)\dots)$ je možno vyjádřit $f(y_{(m-1)},\dots)$ a modelovat tak velkou třídu diskrétních systémů.

Ukažme si metodiku diskrétního modelování na jednoduchém příkladu tvorby modelu pro rovnici

$$y(n) = -0,9y(n-1) + u(n)$$

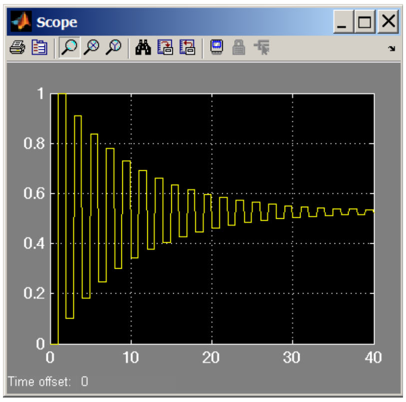
pro $y(1) = 10$ a $u(n) = \mathbf{1}(n)$, kde $\mathbf{1}(n)$ je jednotkový skok. Zapojení v Simulinku, popisující tuto rovnici, uvádíme na obrázku 3.1, výsledek simulace je na obrázku 3.2.



Obrázek 3.1: Schéma modelu v simulinku

Příklad 3.1 (Numerický výpočet odmocniny)
Ověřte, že nelineární diskrétní systém popsaný diferenční rovnicí:

$$y(n) = \frac{1}{2}\left[y(n-1) + \frac{x(n-1)}{y(n-1)}\right]$$



Obrázek 3.2: Výsledek simulace

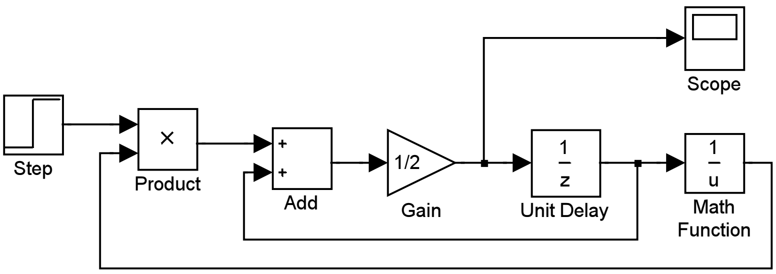
vrací po několika krocích numerický výpočet odmocniny vstupního signálu. Modelujte tuto diferenční rovnici v Simulinku.

Řešení

Odmocnina z čísla 10 je s přesností na 10 desetinných míst rovna $\sqrt{10} = 3,16227766017$. *Nechť pro všechna n platí* $x(0) \equiv x(n) = 10$, *postupným dosazováním pro jakoukoli počáteční podmínku* $y(1) \neq 0$ *dostáváme postupně:*

$$
\begin{array}{ll}
y(1) = 3 & y^2(1) = 9 \\
y(2) = 3,165 & y^2(2) = 10,017225 \\
y(3) = 3,162278 & y^2(3) = 10,00000214928 \\
y(4) = 3,1622776601 & y^2(4) = 9,999999999568 \\
\vdots & \vdots
\end{array}
$$

Blokové schéma v Simulinku je zobrazeno na obrázku 3.3. Počáteční podmínku $y(1) \neq 0$ *je možno zadat jako parametr do bloku* `Unit delay`. *Do bloku* `Fcn` *je možno zapsat jakoukoli funkci v proměnné u, kde u může být i vektor. Na výstupu bloku* `Fcn`*je realizovaná zadaná funkce, např. v našem případě* $\frac{1}{u}$.

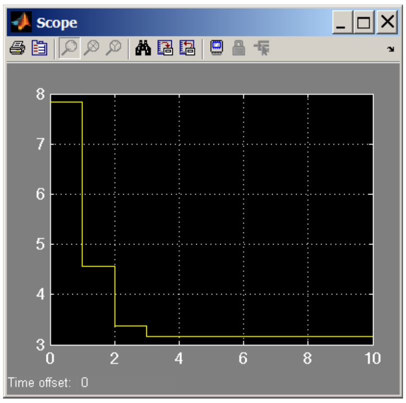


Obrázek 3.3: Schéma modelu v Simulinku

Výsledek simulace je ukázán na obrázku 3.4. Jako počáteční podmínka bylo zvoleno $y(1) = 15$.

3.2 Modelování spojitých systémů

Podstatou modelování spojitých systémů je blok `Integrator`, který integruje vstupní signál. Pokud je na jeho vstup připojen signál $\frac{dy(t)}{dt}$, na výstupu bude signál $y(t)$ s tím, že do bloku `Integrator` lze zapsat



Obrázek 3.4: Konvergence numerického výpočtu druhé odmocniny

počáteční podmínku $y(0)$. Modelování spočívá ve vyjádření proměnné $\frac{d^{(m)}y(t)}{dt^{(m)}}$ jako funkce proměnných

$$\frac{d^{(m)}y(t)}{dt^{(m)}} = f\left(u(t), \frac{d^{(m-1)}y(t)}{dt^{(m-1)}}, \ldots, y(t)\right),$$

kde n je řád systému. Derivované vstupy a výstupy lze realizovat skládáním bloků `Integrator`. Derivované vstupy a výstupy plus aktuální vstup $u(t)$ je možno zpracovat a vypočítat tak velkou třídu spojitých systémů.

Ukažme si postup při modelování spojitého systému na jednoduchém příkladu:

Příklad 3.2 (O vlcích a ovcích)
Označme si:
x...počet lovených kořistí, zde počet ovcí
y...počet dravců, zde počet vlků

Počet narozených ovcí v časovém intervalu $\langle t; t+\Delta t\rangle$ lze vyjádřit rovnicí:

$$\Delta x_n(t) = k_1 x(t)\Delta t, \tag{3.1}$$

kde k_1 popisuje schopnost ovcí se rozmnožovat. Počet zahubených kořistí v časovém intervalu $\langle t; t+\Delta t\rangle$ lze vyjádřit rovnicí:

$$\Delta x_m(t) = k_2 x(t) y(t)\Delta t, \tag{3.2}$$

kde k_2 je relativní kořistění vlků na ovcích, součin $x(t)y(t)$ je počet setkání ovce s vlkem. Celková změna počtu ovcí je potom:

$$\begin{aligned} \Delta x(t) &= \Delta x_n(t) - \Delta && (3.3)\\ x_m(t) &= [k_1 x_(t) - k_2 x(t) y(t)]\Delta t && (3.4) \end{aligned}$$

Vynásobíme-li celou rovnici $1/\Delta t$ dostaneme:

$$\frac{\Delta x(t)}{\Delta t} = k_1 x(t) - k_2 x(t) y(t) \tag{3.5}$$

Položíme-li $\Delta t \to 0$, potom dostaneme:

$$\frac{dx(t)}{dt} = k_1 x(t) - k_2 x(t) y(t) \tag{3.6}$$

Podobně postupujeme i u vlků. Počet narozených vlků v časovém intervalu $\langle t; t+\Delta t\rangle$ lze vyjádřit rovnicí:

$$\Delta y_n(t) = k_3 k_2 y(t) x(t) \Delta t, \tag{3.7}$$

kde k_3 je výživnost ovcí (účinnost přeměny energie z biomasy).

$$\Delta y_m(t) = k_4 y(t) \Delta t \tag{3.8}$$

je počet pošlých vlků v časovém intervalu $\langle t; t+\Delta t\rangle$, kde k_4 je úmrtnost vlků. Stejnými úpravami, jako u rovnic ovcí dostaneme

$$\frac{dy(t)}{dt} = k_3 k_2 y(t) x(t) - k_4 y(t). \tag{3.9}$$

Experimentálně již byly na tomto modelu vyzkoušeny konstanty:

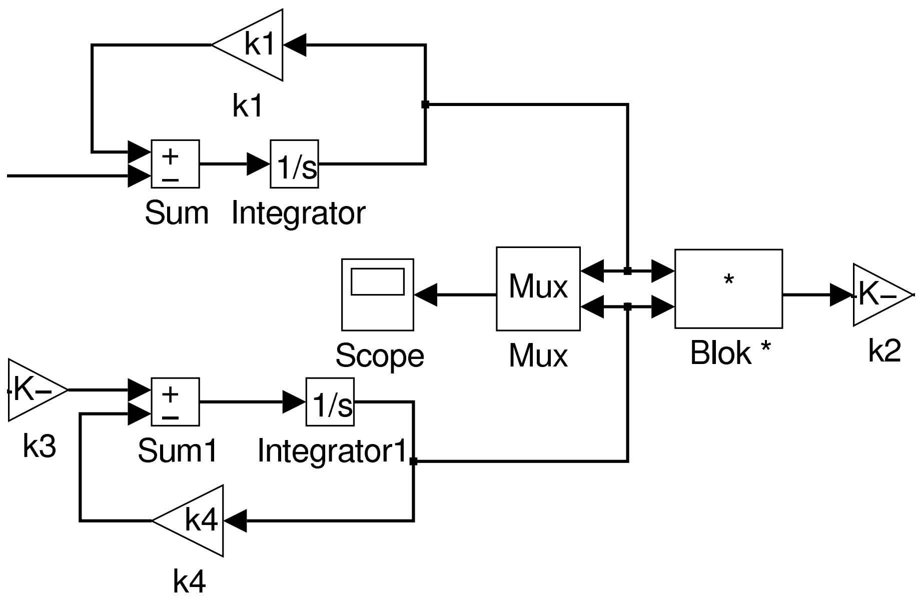
k_1=*0,4*

k_2=*0,017*

k_3=*0,7*

k_4=*1,2*

Počáteční podmínky: Počet ovcí 100, počet vlků 5.

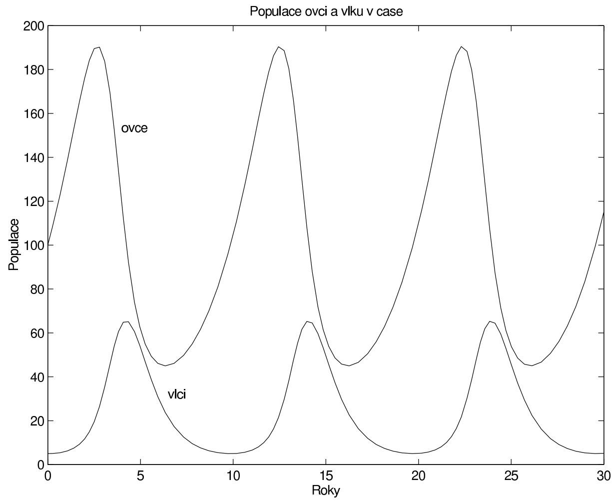


Obrázek 3.5: Blokové schéma v Simulinku

Na obrázku 3.6je grafické znázornění počtu ovcí a vlků v čase. Simulace ukazuje vzájemnou závislost populace vlků a ovcí. Přemnoží-li se ovce, vlci mají dostatek potravy a množí se. S přibývajícím počtem vlků zase klesá počet ovcí. Vlci začnou mít nedostatek potravy a vymírají. Ovce mají příznivé podmínky se množit a jejich populace roste. Cyklus se opakuje.

Příklad 3.3 (Nabídka – poptávka)

Vyjděme z ekonomického modelu, kdy nabídka v n-tém časovém intervalu $o(n)$ je přímo úměrná ceně produktu v minulém časovém intervalu $c(n-1)$ a naopak poptávka v n-tém časovém intervalu $d(n)$ je přímo úměrná současné záporné ceně produktu $y(n)$. Tento model je realistický, neboť čím byla v minulosti větší cena, tím je dnes větší nabídka a naopak, čím je dnes větší cena, tím klesá i poptávka. Zavedeme-li proměnnou $u(n)$,



Obrázek 3.6: Vývoj populace ovcí a vlků v průběhu 30 let

která charakterizuje počet vyrobených kusů produktu v čase n a podmínku, že nabídka v čase n se rovná poptávce v čase n, můžeme zapsat následující tři diferenční rovnice:

$$\begin{aligned} o(n) &= c\cdot y(n-1)+a\cdot u(n),\\ d(n) &= -e\cdot y(n)+b\cdot u(n),\\ o(n) &= d(n). \end{aligned} \tag{3.10}$$

Úkol:

Pro ekonomický model definovaný soustavou (3.10) namodelujte vývoj ceny $y(n)$, nabídky $o(n)$ a poptávky $d(n)$ v Simulinku pro hodnoty parametrů $a = 5, b = 225, c = 5, e = 5,5$.

Řešení:

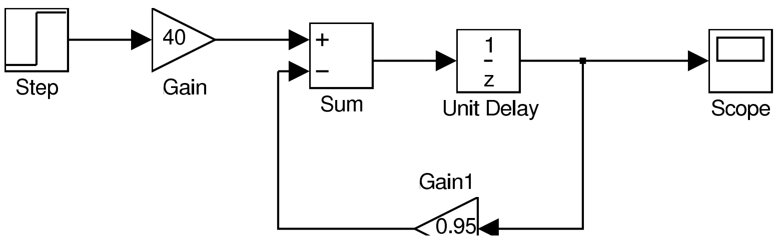
Vyloučením nabídky a poptávky ze soustavy rovnic získáme jedinou diferenční rovnici pro časový vývoj ceny $y(n)$,

$$y(n)+\left(\frac{c}{e}\right)\cdot y(n-1)=\frac{b-a}{e}\cdot u(n), \tag{3.11}$$

a tuto rovnici dále upravíme na

$$y(n)=\frac{b-a}{e}\cdot u(n)-\left(\frac{c}{e}\right)\cdot y(n-1). \tag{3.12}$$

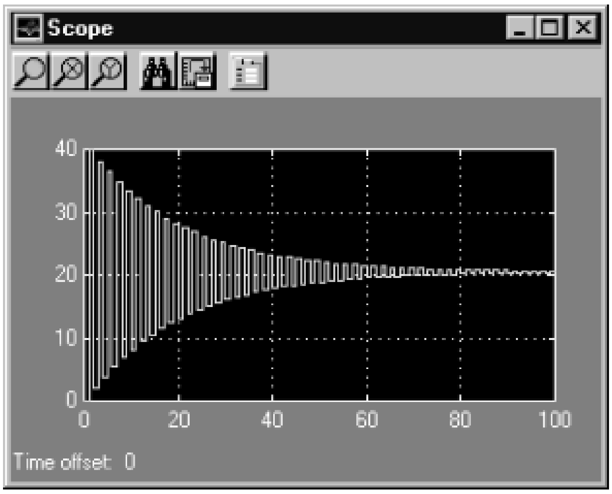
Použitím bloku jednotkového zpoždění `Unit delay`*, zpožďujícího signál $y(n)$ o jeden krok, je možno vytvořit v Simulinku jednoduše model celé rovnice tak, jak je ukázáno na obrázku 3.7.*



Obrázek 3.7: Schéma modelu v Simulinku

Vývoj ceny $y(n)$ ukazuje obrázek 3.8, na kterém je možno pozorovat pozvolnou konvergenci ceny k teoretickému ekvilibru, jež je pro nekonečný čas rovno

$$\lim y(n)_{n\to\infty}=\frac{b-a}{d}\cdot k_2.$$



Obrázek 3.8: Časový vývoj ceny