Vysoká škola ekonomická v Praze

Kvantitativní ekonomie

Vladimír Holý a Michal Černý

2021

Obsah

Úvod

Tato skripta slouží jako doprovodná literatura k předmětu Kvantitativní ekonomie na Vysoké škole ekonomické v Praze. Obsahem jsou vybrané kvantitativní přístupy zkoumající ekonomickou realitu na mikro i makro úrovni. Skripta nabízí přehled základních statických, lineárních a deterministických ekonomických modelů. Hlavní důraz je kladen na aplikaci samotných výpočtů v matematickém programu MATLAB. Pro přehlednost se v textu objevují různé vsuvky několika typů.

Výklad teorie

Důležité matematické a ekonomické pojmy jsou umístěny do červeného rámečku. Seznam těchto pojmů lze nalézt v dodatku na stránce 142.

Návod k MATLABu

Ve žlutém, rámečku se nacházejí základní instrukce pro práci s MATLABem. Seznam těchto rámečků se nachází na straně 143. V dodatku B jsou pak vypsány základní funkce Matlabu.

Skript

Zelený rámeček ohraničuje skript spustitelný v MATLABu. U levého okraje je vždy zobrazeno číslo řádku příslušného souboru se skriptem. Seznam skriptů je na straně 144.

Poznámka / Doporučená literatura

V šedivém rámečku jsou umístěny poznámky na okraj výkladu. Na konci každé kapitoly také šedivý rámeček obsahuje doporučenou literaturu.

Práce na těchto skriptech byla podpořena v rámci projektu F4/18/2016 Interní rozvojové soutěže Fakulty informatiky a statistiky Vysoké školy ekonomické v Praze.

Kapitola 1

Pravděpodobnostní rozdělení a lineární regrese

V této kapitole se budeme zabývat generováním náhodných dat a následně jednoduchou regresní analýzou. Představíme si normální rozdělení a t-rozdělení a ukážeme si, jak si vykreslit obrázky jejich distribučních funkcí a hustot a jak generovat náhodné veličiny z těchto rozdělení. Dále si zkusíme jednoduchou regresní analýzu nad generovanými daty.

Klíčová slova: *normální rozdělení, t-rozdělení, lineární regrese, regresní přímka, rezidua, t-test.*

1.1 Normální rozdělení

Nejdříve si vedle sebe zobrazíme grafy distribuční funkce $F(x)$ a hustoty $f(x)$ normovaného normálního rozdělení $N(0,1)$. Grafy obou funkcí si necháme vykreslit na intervalu $x \in [-5,5]$. Toho docílíme tak, že si tento interval rozdělíme na několik od sebe stejně vzdálených bodů, vytvoříme si vektor $(-5,-4.9,-4.8,\ldots,5)$. Krok 0,1 volíme jako příklad, v principu můžeme zvolit jakýkoliv dostatečně malý krok, např. 0,01 nebo 0,001. Pro každý bod z tohoto vektoru si spočteme hodnotu funkcí $F(x)$ i $f(x)$ a zaneseme do grafu.

Funkce `plot` a práce s grafy

Vykreslení grafu v MATLABu typicky probíhá v následujících krocích.o

- Funkce `figure` otevře prázdné okno grafu.
- Pokud potřebujeme více grafů v jednom okně, zavoláme funkci `suplot(m,n,p)`, která nám okno grafů rozdělení na `m`×`n` dlaždicí a vybere `p`-tou dlaždici pro vykreslení.
- Hlavní funkcí pro dvojrozměrné grafy je `plot`. Její základní podoba `plot(X,Y)` nám vykreslí graf hodnoty `Y` proti hodnotám `X`. Tato funkce může mít řadu argumentů, které ovlivňují vzhled grafu, jak si ukážeme v průběhu celého textu.
- Pokud to vyžadujeme, funkce `axis([a b c d])` nastaví horiznotální osu na rozsah od `a` do `b` a vertikální osu od `c` do `d`
- Funkce `title(t)` pojmenuje graf a dvojice funkcí `xlabel(t)` a `ylabel(t)` pojmenuje osy.
- Příkaz `hold on` způsobí, že další zavolání funkce `plot` do současného grafu přidá vykreslení. Příkaz `hold off` způsobí, že další zavolání `plot` současný graf smaže.

Pravděpodobnostní rozdělení

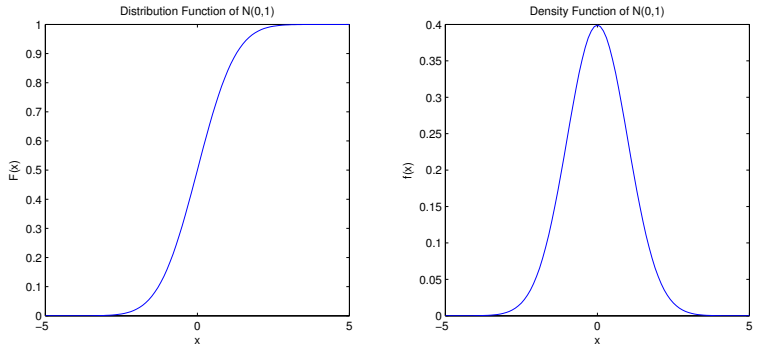
Pro práci s pravděpodobnostními rozděleními obsahuje MATLAB několik funkcí.

- `cdf('name',x,...)` vrátí hodnotu distribuční funkce rozdělení `'name'` v bodě `x`.
- `pdf('name',x,...)` vrátí hodnotu funkce hustoty rozdělení `'name'` v bodě `x`.
- `icdf('name',y,...)` vrátí hodnotu kvantilové funkce (tedy inverzi k distribuční funkci) rozdělení `'name'` v bodě `y`.
- `random('name',...)` vrátí náhodné číslo vygenerované z rozdělení `'name'`.

Dalšími argumenty těchto funkcí jsou parametry konkrétních rozdělení. Následuje přehled často používaných rozdělení s jejich MATLAB jménem a možnými argumenty.

Binomické rozdělení	`'bino'`	`N, p`
χ^2-rozdělení	`'chi'`	`nu`
Exponenciální rozdělení	`'exp'`	`mu`
F-rozdělení	`'f'`	`nu1, nu2`
Geometrické rozdělení	`'geo'`	`p`
Logistické rozdělení	`'logistic'`	`mu, sigma`
Normální rozdělení	`'norm'`	`mu, sigma`
Poissonova rozdělení	`'poiss'`	`lambda`
Studentovo t-rozdělení	`'t'`	`nu`
Spojité rovnoměrné rozdělení	`'unif'`	`a, b`
Diskrétní rovnoměrné rozdělení	`'unid'`	`N`

Např. příkaz `x = random('norm', 0, 2, [10^6, 1])` vytvoří simulovaný výběr z $N(0, 2^2)$ o velikosti 10^6 a vloží jej do sloupcového vektoru `x`.



Obrázek 1.1: Distribuční funkce (vlevo) a hustota (vpravo) normovaného normálního rozdělení.

NormStudent.m: Grafy normálního rozdělení

```
3  figure;
4  subplot(1, 2, 1);
5  xAxis = -5:0.1:5;
6  yAxis = cdf('norm', xAxis, 0, 1);
7  plot(xAxis, yAxis);
8  title('Distribution Function of N(0,1)');
9  xlabel('x');
10 ylabel('F(x)');
11 subplot(1, 2, 2);
12 xAxis = -5:0.1:5;
13 yAxis = pdf('norm', xAxis, 0, 1);
14 plot(xAxis, yAxis);
15 title('Density Function of N(0,1)');
16 xlabel('x');
17 ylabel('f(x)');
```

Nyní si vygenerujeme $n = 30$ náhodných hodnot $\boldsymbol{z} = (z_1, \ldots, z_n)'$ z normálního rozdělení se střední hodnotou $\mu = 0$ a směrodatnou odchylkou $\sigma = 20$.

LinReg.m: Náhodný výběr z normálního rozdělení

```
3  n = 30;
4  z = random('norm', 0, 20, [n 1]); % sloupcovy vektor n x 1
```

Tento náhodný výběr si necháme vykreslit do grafu. Dále si nakreslíme histogram o 6 sloupcích s proloženou hustotou normálního rozdělení. S větším rozsahem náhodného výběru bude histogram více podobný hustotě normálního rozdělení.

LinReg.m: Grafy náhodného výběru

```
6  figure;
7  subplot(1, 2, 1);
8  plot(z, '.b', 'MarkerSize', 20);
9  title('Random Sample from N(0,20)');
10 xlabel('i');
11 ylabel('z_i'); % podtrzitko vysazi i jako dolni index
12 subplot(1, 2, 2);
13 histfit(z, 6, 'norm');
14 title('Histogram of Random Sample from N(0,20)');
```

Specifikace čáry ve funkci `plot`

Ve funkci `plot` můžeme pomocí argumentu `LineSpec` určit tři atributy vykreslování:

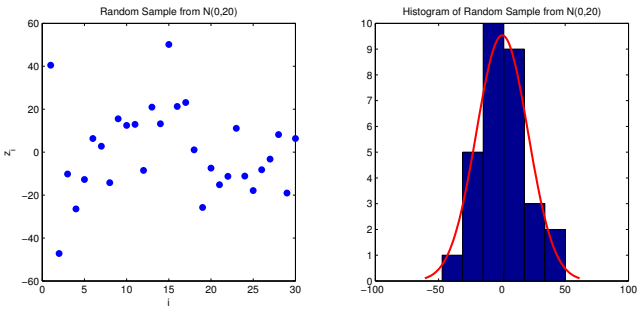
- styl čar spojujících body,
- typ symbolu jednotlivých bodů,
- barvu čar a bodů.

Tyto atributy nastavíme pomocí jednoho řetězce, který je složený ze znaků odpovídajícím jednotlivým hodnotám atributům. Např. `plot(x,y,'-xr')` vykreslí graf jako spojité čáry (symbol `'-'`), s křížky jako body (symbol `'x'`) v červené barvě (symbol `'r'`). Symboly atributů mohou být zadány v libovolném pořadí. Následuje výčet možných hodnot atributů.

	Typy čar		**Typy bodů**
`'-'`	Nepřerušovaná čára	`'+'`	Znaménko plus
`'--'`	Přerušovaná čára	`'o'`	Kruh
`':'`	Tečkovaná čára	`'*'`	Hvězdička
`'-.'`	Čerchovaná čára	`'.'`	Bod
		`'x'`	Křížek
	Druhy barev	`'s'`	Čtverec (square)
`'r'`	Červená (red)	`'d'`	Diamant (diamond)
`'g'`	Zelená (green)	`'^'`	Trojúhelník směřující nahoru
`'b'`	Modrá (blue)	`'v'`	Trojúhelník směřující dolů
`'c'`	Tyrkysová (cyan)	`'>'`	Trojúhelník směřující doprava
`'m'`	Purpurová (magenta)	`'<'`	Trojúhelník směřující doleva
`'y'`	Žlutá (yellow)	`'p'`	Pěticípá hvězda (pentagram)
`'k'`	Černá (black)	`'h'`	Šesticípá hvězda (hexagram)
`'w'`	Bílá (white)		

1.2 Analýza generovaných dat pomocí lineární regrese

Náš náhodný výběr $\boldsymbol{z} = (z_1, \ldots, z_n)'$ teď použijeme pro vytvoření dat, se kterými budeme dále pracovat. Jedná se možná o trochu zvláštní postup, kdy budeme analyzovat data, která si sami vytvoříme, ale nám jde teď především o ilustraci použitých metod než o řešení skutečného empirického problému, kde data sbírá analytik. To je základní princip tzv. simulace. Jde o užitečnou metodu, kdy si sami generujeme



Obrázek 1.2: Náhodný výběr z $N(0, 20)$ (vlevo) a jeho histogram (vpravo).

data, která následně předkládáme statistickým procedurám. Tímto způsobem můžeme zkoumat vlastnosti statistických metod pro generovaná data. To může být velmi užitečné v případě, kdy teoretické odvození vlastností metod je příliš náročné.

Vytvoříme si dva vektory $\boldsymbol{x} = (x_1, \ldots, x_n)'$ a $\boldsymbol{y} = (y_1, \ldots, y_n)'$ dané předpisem

$$\begin{aligned} x_i &= 10 + 0.5(i-1) && \text{pro } i = 1, \ldots, n, \\ y_i &= 3 + 5x_i + z_i && \text{pro } i = 1, \ldots, n. \end{aligned} \tag{1.1}$$

LinReg.m: Generování dat

```
16 x = (10:0.5:24.5)';
17 y = 3 + 5 * x + z;
```

Dále si necháme vykreslit závislost $\boldsymbol{y}$ na $\boldsymbol{x}$, tedy body o souřadnicích (x_i, y_i) pro $i = 1, \ldots, n$.

LinReg.m: Vykreslení dat

```
19 figure;
20 plot(x, y, '.b', 'MarkerSize', 20);
21 title('Data for Linear Regression');
22 xlabel('x');
23 ylabel('y');
```

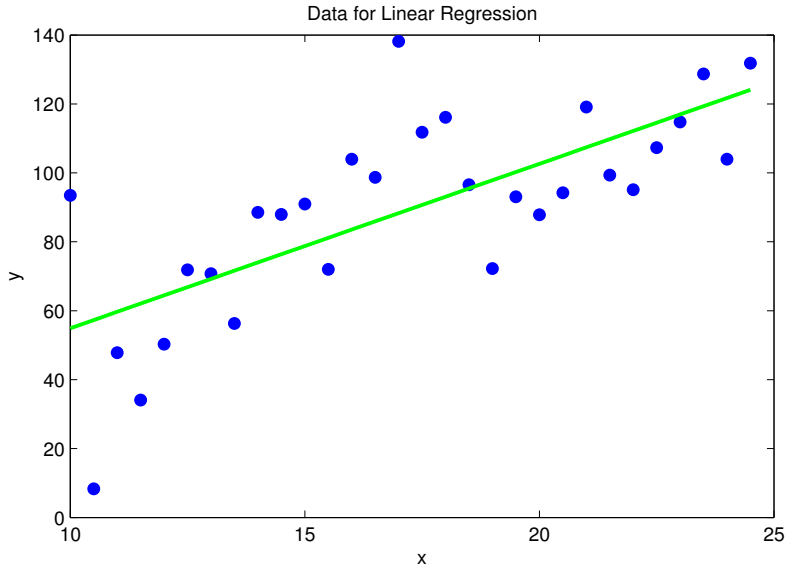
Nyní „zapomeneme“, jakým způsobem jsme data vygenerovali a pokusíme se modelovat závislost $\boldsymbol{y}$ na $\boldsymbol{x}$ pomocí lineární regrese. Zapíšeme si tedy model regresní přímky

$$y_i = \beta_1 + \beta_2 x_i + \varepsilon_i, \qquad \varepsilon_i \overset{iid}{\sim} N(0, \sigma^2) \qquad \text{pro } i = 1, \ldots, n, \tag{1.2}$$

kde β_1 je konstantní koeficient, β_2 je koeficient regresoru $\boldsymbol{x}$ a ε_i jsou nezávislé náhodné veličiny z normálního rozdělení se střední hodnotou $\mu = 0$ a neznámou směrodatnou odchylkou σ. Tento model můžeme zapsat i v obecném maticovém značení jako

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \qquad \boldsymbol{\varepsilon} \sim N(\boldsymbol{0}, \sigma^2 \boldsymbol{I}), \tag{1.3}$$

kde $\boldsymbol{0}$ je vektor nul a $\boldsymbol{I}$ je jednotková matice. Označme k počet parametrů, v našem případě $k = 2$. Matice $\boldsymbol{X}$ má tedy n řádků a k sloupců, kde první sloupec je tvořený samými jedničkami a druhý sloupec je



Obrázek 1.3: Proložení dat regresní přímkou.

tvořený vektorem $\boldsymbol{x}$. Metodou nejmenších čtverců si můžeme spočítat odhad koeficientů $\hat{\boldsymbol{\beta}} = (\hat{\beta}_1, \hat{\beta}_2)'$ pomocí známého vzorce

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{y}. \tag{1.4}$$

Dále si spočteme proložené hodnoty

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\hat{\boldsymbol{\beta}}. \tag{1.5}$$

LinReg.m: Lineární regrese

```
25 k = 2;
26 X = [ones(n, 1) x];
27 betaHat = (X' * X)^(-1) * X' * y
28 yHat = X * betaHat;
```

Regresní přímku si přidáme do grafu dat.

LinReg.m: Přidání regresní přímky

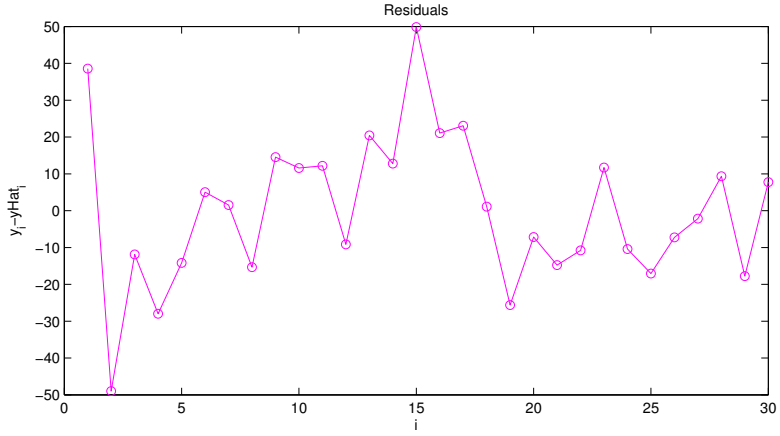
```
30 hold on;
31 plot(x, yHat, 'g', 'LineWidth', 2);
32 hold off;
```

Dalším krokem bude výpočet reziduí modelu

$$\boldsymbol{r} = \boldsymbol{y} - \hat{\boldsymbol{y}} \tag{1.6}$$

a jejich zobrazení v grafu podle indexu i. Pomocí reziduí $\boldsymbol{r} = (r_1, \ldots, r_n)'$ můžeme ještě spočítat statistiku

$$s^2 = \frac{1}{n-k}\sum_{i=1}^{n} r_i^2, \tag{1.7}$$



Obrázek 1.4: Rezidua regresního modelu.

která je nestranným odhadem rozptylu σ^2 náhodných chyb ε_i.

LinReg.m: Výpočet reziduí a odhadu rozptylu

```
34 res = y - yHat;
35 s2 = sum(res.^2) / (n - k)
```

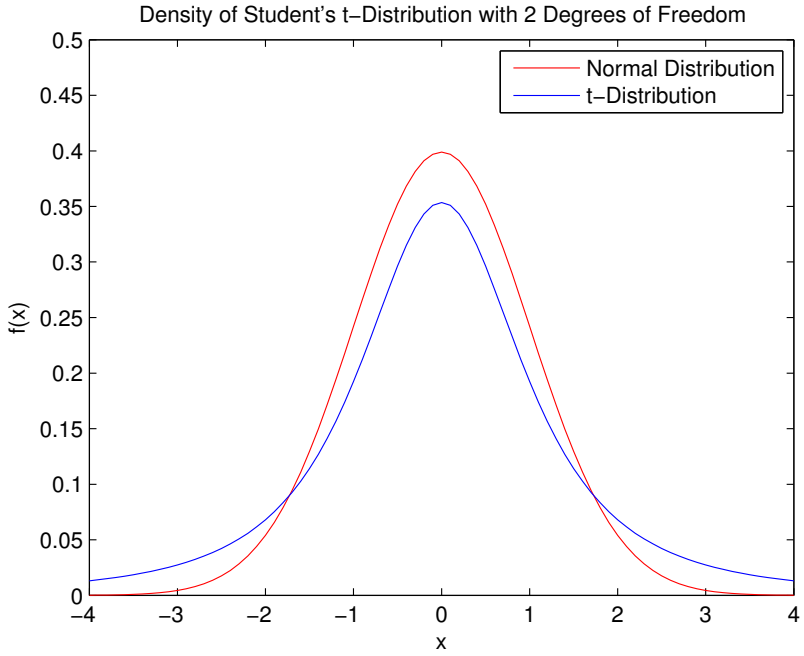
Rezidua si dále můžeme vykreslit do grafu.

LinReg.m: Graf reziduí

```
37 figure;
38 xAxis = 1:n;
39 plot(xAxis, res, 'mo-');
40 title('Residuals');
41 xlabel('i');
42 ylabel('y_i-yHat_i');
```

1.3 t-rozdělení a t-test

Nakonec budeme testovat nulovost regresních parametrů. Než ale přistoupíme k výpočtu samotné testové statistiky, připomeňme si, jak vypadá Studentovo t-rozdělení. Nakreslíme si jeho hustotu pro různé stupně volnosti a ukážeme si, že s rostoucím počtem stupňů volnosti konverguje Studentovo t-rozdělení k normálnímu rozdělení. Graf si vykreslíme v cyklu pro počty stupňů volnosti $df = 1, \ldots, 30$, kdy v každé iteraci vykreslíme hustotu normálního rozdělení a hustotu Studentova t-rozdělení s df stupni volnosti. Na konci každé iterace ještě zastavíme program na půl vteřiny, čímž se nám graf vykreslí jako animace s rostoucím počtem stupňů volnosti.



Obrázek 1.5: Konvergence hustoty t-rozdělení k normálnímu rozdělení.

NormStudent.m: Graf t-rozdělení

```
figure;
xAxis = -4:0.1:4;
for df = 1:30
  plot(xAxis, pdf('norm', xAxis, 0, 1), '-r');
  hold on;
  plot(xAxis, pdf('t', xAxis, df), '-b');
  legend('Normal Distribution', 't-Distribution');
  hold off;
  axis([-4 4 0 0.5]);
  tit = ['Density of Student''s t-Distribution with '];
  tit = [tit, num2str(df), ' Degrees of Freedom'];
  title(tit);
  xlabel('x');
  ylabel('f(x)');
  pause(0.5);
end
```

Přejdeme k odvození t-statistiky. V modelu (1.3) platí

$$\hat{\boldsymbol{\beta}} \sim N(\boldsymbol{\beta}, \boldsymbol{\Omega}), \qquad \boldsymbol{\Omega} = \sigma^2(\boldsymbol{X}'\boldsymbol{X})^{-1}.$$

Potom je

$$\hat{\boldsymbol{\Omega}} := \hat{\sigma}^2(\boldsymbol{X}'\boldsymbol{X})^{-1}$$

nestranný odhad kovarianční matice $\boldsymbol{\Omega}$ estimátoru $\hat{\boldsymbol{\beta}}$. Speciálně, diagonální prvek $\hat{\Omega}_{j,j}$ je odhadem rozptylu $\hat{\beta}_j$, a tudíž $\sqrt{\hat{\Omega}_{j,j}}$ je odhadem směrodatné odchylky $\hat{\beta}_j$. Nyní si již vypočteme testovou t-statistiku

$$T_j = \frac{|\hat{\beta}_j|}{\sqrt{\hat{\Omega}_{j,j}}}. \tag{1.8}$$

Je-li tato testová statistika menší než kvantil $t_{n-k}(1-\frac{\alpha}{2})$ t-rozdělení s $n-k$ stupni volnosti, nezamítáme hypotézu o nulovosti koeficientu β_j na hladině α. Kritický obor, ve kterém zamítáme nulovost koeficientu β_j, je tedy množina

$$W = \left\{T_j \geq t_{n-k}\left(1-\frac{\alpha}{2}\right)\right\}. \tag{1.9}$$

Test budeme provádět na hladině $\alpha = 0.05$. Vypočteme si testovou statistiku T_j a kvantil $t_{n-k}(0.975)$ a zjistíme, zda uvedená nerovnost platí či ne.

LinReg.m: Výpočet t-testu

```
44 alpha = 0.05;
45 omegaHat = s2 * (X' * X)^(-1);
46 tStat = abs(betaHat) ./ diag(omegaHat).^(1/2)
47 tCrit = icdf('t', 1 - alpha / 2, n-k)
48 inW = tStat > tCrit
```

Poznámka

Díky náhodně generovaným datům při každém zavoláním programu samozřejmě dojdeme k různým výsledkům. Můžeme si libovolně upravovat počet pozorování nebo směrodatnou odchylku našeho náhodného výběru. Nižší počet pozorování a vyšší rozptyl náhodné složky dat bude mít tendenci nezamítat nulovost koeficientu β_2. U vyššího počtu pozorování a nižšího rozptylu pak dojde k opačné situaci.

Doporučená literatura

- Anděl, J. 2011. *Základy matematické statistiky*. MatfyzPress. ISBN 978-80-7378-162-0. http://matfyzpress.cz/matematika/14-zaklady-matematicke-statistiky.html
- Cipra, T. 2014. *Finanční ekonometrie*. Ekopress. ISBN 978-80-86929-93-4. https://www.ekopress.cz/titdetail.php?tid=30238
- Greene, W. H. 2018. *Econometric Analysis*. Pearson. ISBN 978-0-13-446136-6. https://www.pearson.com/us/higher-education/program/Greene-Econometric-Analysis-8th-Edition/PGM334862.html
- James, G., Witten, D., Hastie, T., Tibshirani, R. 2013. *An Introduction to Statistical Learning: With Applications in R*. Springer. ISBN 978-1-4614-7137-0. https://doi.org/10.1007/978-1-4614-7138-7
- Wooldridge, J. M. 2019. *Introductory Econometrics: A Modern Approach*. Cengage Learning. ISBN 978-1-337-55886-0. https://www.cengage.co.uk/books/9781337558860/
- Zvára, K. 2019. *Regrese*. MatfyzPress. ISBN 978-80-7378-406-5. http://matfyzpress.cz/matematika/232-regrese-s-cd.html

Kapitola 2

Metoda nejmenších čtverců a metoda LAD

Tradičně se pro odhad koeficientů lineární regrese používá metoda nejmenších čtverců. My si představíme její alternativu - metodu LAD, která je více robustní k odlehlým pozorováním.

Klíčová slova: *lineární regrese, regresní rovina, metoda nejmenších čtverců, metoda LAD, odlehlé pozorování.*

2.1 Data a model lineární regrese

Pomocí funkce `csvread` si načteme data, která budeme chtít analyzovat. Nebudeme se zabývat interpretací dat. Jediné, co potřebujeme vědět, je, že máme k dispozici $n = 20$ pozorování pro proměnné $\boldsymbol{y} = (y_1, \ldots, y_n)'$, $\boldsymbol{x}_1 = (x_{1,1}, \ldots, x_{n,1})'$ a $\boldsymbol{x}_2 = (x_{1,2}, \ldots, x_{n,2})'$. Budeme zkoumat závislost $\boldsymbol{y}$ na $\boldsymbol{x}_1$ a $\boldsymbol{x}_2$ pomocí linerání regrese

$$y_i = \beta_1 + x_{i,1}\beta_2 + x_{i,2}\beta_3 + \varepsilon_i, \qquad i = 1, \ldots, n.$$

Můžeme také použít maticový zápis

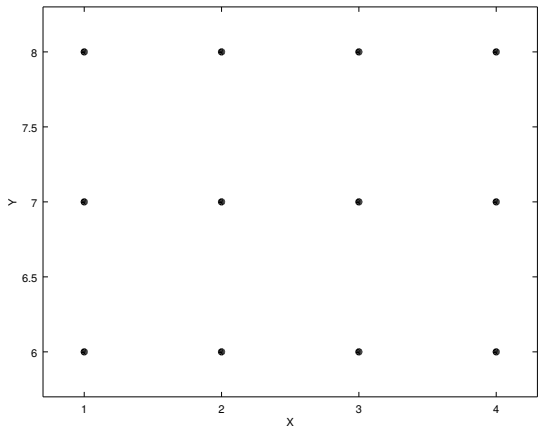
$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \tag{2.1}$$

kde jsme označili $\boldsymbol{X} = (\boldsymbol{e}, \boldsymbol{x}_1, \boldsymbol{x}_2)$, $\boldsymbol{e} = (1, \ldots, 1)'$, $\boldsymbol{\beta} = (\beta_1, \beta_2, \beta_3)'$ a $\boldsymbol{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_n)'$.

LAD.m: Načtení dat

```
3   data = csvread('Data.csv');
4   Y = data(:,1);
5   n = length(Y);
6   X = [ones(n, 1) data(:, 2) data(:, 3)];
```

Načtená data si zobrazíme jako trojrozměrný graf pomocí funkce `plot3`.



Obrázek 2.1: Mřížky vytvořené z horizontálních souřadnic $(1, 2, 3, 4)'$ a vertikálních souřadnic $(6, 7, 8)'$.

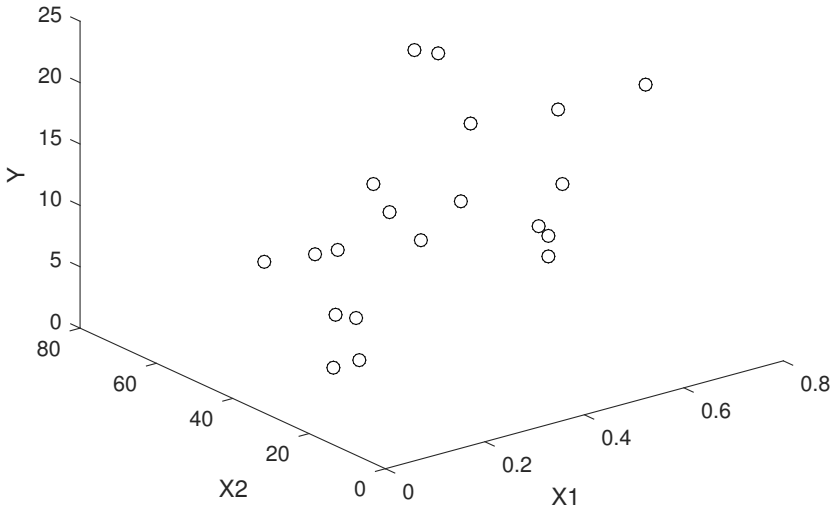
Vizualizace 3D dat

V MATLABu se pro vykreslení trojrozměrnch grafů používají následující funkce.

- Funkce `plot3(x, y, z)` je trojrozměrná obdoba `plot(x, y)`. Lze použít stejné specifikace čar jako u `plot(x, y)`, např. `'r'` pro červenou barvu.
- Funkce `surf(X, Y, Z)` slouží k pro vykreslení trojrozměrného povrchu. Tato funkce může mít jako většina funkcí několik způsobů zápisu. V případě, že funkci dodáme tři stejně velké matice `X`, `Y` a `Z`, funkce vykreslí povrch o výškách daných hodnotami v `Z` o souřadnicích daných hodnotami v `X` a `Y`. Matice `X` a `Y` se zpravidla vytvoří pomocí funkce `meshgrid`.
- Funkce `contour(X, Y, Z)` funguje podobně jako funkce `surf`, jen místo trojrozměrného grafu vykreslí graf dvojrozměrný. Funkce vykreslí vrstevnice (čáry, jejíž body mají stejnou výšku) podle hodnot v `Z`.
- Funkce `meshgrid(x, y)` spočte dvojici matic `[X, Y]`. Matice `X` je vytvořena opakováním vektoru `x` v řádcích. Matice `Y` je vytvořena opakováním vektoru `y` ve sloupcích. Obě matice `x` a `Y` mají počet řádků rovný délce `y` a počet sloupců rovný délce `x`. Tuto funkci lze interpretovat jako vytvoření mřížky v dvourozměrném prostoru. Uvedeme jednoduchý příklad. Horizontálním souřadnicím $(1, 2, 3, 4)'$ a vertikálním souřadnicím $(6, 7, 8)'$ odpovídá mřížka znázorněná na obrázku 2.1. Horizontální souřadnice jednotlivých bodů mřížky si zapíšeme do matice $\boldsymbol{X}$ a vertikální souřadnice do matice $\boldsymbol{Y}$. V našem příkladu tedy máme

$$\boldsymbol{X} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}, \qquad \boldsymbol{Y} = \begin{pmatrix} 6 & 6 & 6 & 6 \\ 7 & 7 & 7 & 7 \\ 8 & 8 & 8 & 8 \end{pmatrix}.$$

  Tyto matice získáme zavoláním příkazu `[X, Y] = meshgrid([1 2 3 4], [6 7 8])`.



Obrázek 2.2: Proměnná $\boldsymbol{y}$ v závislosti na $\boldsymbol{x}_1$ a $\boldsymbol{x}_2$.

LAD.m: Zobrazení dat

```
8   figure(1);
9   plot3(X(:, 2), X(:, 3), Y, 'ok');
10  xlabel('X1');
11  ylabel('X2');
12  zlabel('Y');
13  hold on;
```

2.2 Metoda nejmenších čtverců

Koeficienty $\boldsymbol{\beta}$ z modelu (2.1) odhadneme metodou nejmenších čtverců. K tomu použijeme vzorec (1.4) z kapitoly 1. Tyto odhady označíme jako $\hat{\boldsymbol{\beta}}^{OLS} = (\hat{\beta}_1^{OLS}, \hat{\beta}_2^{OLS}, \hat{\beta}_3^{OLS})'$. Metodu nejmenších čtverců si naprogramujeme jako funkci s argumenty $\boldsymbol{y}$ a $\boldsymbol{X}$.

LAD.m: Metoda nejmenších čtverců

```
15  function betaHat = myOLS(Y, X)
16    betaHat = (X' * X)^(-1) * X' * Y;
17  end
```

Dále si budeme chtít odhadnuté koeficienty zakreslit do grafu. V případě závislosti na jedné proměnné jsme vykreslovali (viz kapitola 1) regresní přímku do dvojrozměrného grafu. V případě závislosti na dvou proměnných teď budeme vykreslovat regresní rovinu do trojrozměrného grafu.

Nejdříve si vytvoříme mřížky pro hodnoty $\boldsymbol{x}_1$ a $\boldsymbol{x}_2$, ve kterých budeme počítat hodnoty regresní plochy. Hodnoty $\boldsymbol{x}_1$ budeme brát z $\boldsymbol{g} = (0, 0.1, \ldots, 0.7)'$, hodnoty $\boldsymbol{x}_2$ z $\boldsymbol{h} = (0, 10, \ldots, 70)'$. Dále si vytvoříme maticové mřížky

$$\boldsymbol{G} = \left(g_{i,j}\right)_{i=1,j=1}^{8,8} = \begin{pmatrix} 0 & 0.1 & \ldots & 0.7 \\ \vdots & & & \vdots \\ 0 & 0.1 & \ldots & 0.7 \end{pmatrix} \quad \text{a} \quad \boldsymbol{H} = \left(h_{i,j}\right)_{i=1,j=1}^{8,8} = \begin{pmatrix} 0 & \ldots & 0 \\ 10 & & 10 \\ \vdots & & \vdots \\ 70 & \ldots & 70 \end{pmatrix},$$

které v MATLABu získáme příkazem `meshgrid`.

LAD.m: Mřížka vysvětlovaných proměnných

```
19  X1grid = 0.0:0.1:0.7;
20  X2grid = 0:10:70;
21  [X1mesh X2mesh] = meshgrid(X1grid, X2grid);
22  [I J] = size(X1mesh);
```

Pro každý bod (i, j) z těchto mřížek si odhadneme vysvětlovanou proměnnou (tedy výšku regresní roviny) jako

$$\hat{y}_{i,j}^{OLS} = \hat{\beta}_1^{OLS} + g_{i,j}\hat{\beta}_2^{OLS} + h_{i,j}\hat{\beta}_3^{OLS}.$$

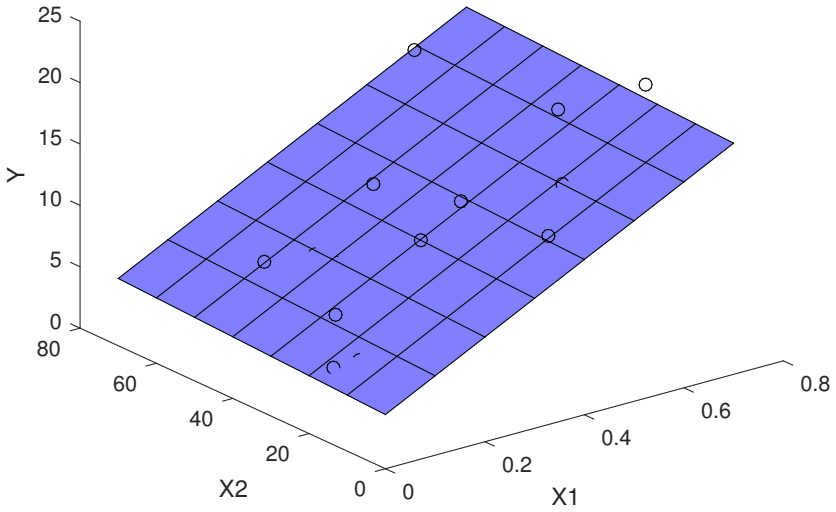
V MATLABu pak regresní rovinu vykreslíme pomocí příkazu `surf`. Rovinu přidáme do stávajícího obrázku s daty.

LAD.m: Odhad regresní roviny metodou nejmenších čtverců

```
24  betaOLS = myOLS(Y, X)
25  YmeshOLS = zeros(I, J);
26  for i = 1:I
27    for j = 1:J
28      YmeshOLS(i, j) = betaOLS(1) ...
29                     + X1mesh(i, j) * betaOLS(2) ...
30                     + X2mesh(i, j) * betaOLS(3);
31    end
32  end
33  figure(1);
34  surf(X1mesh, X2mesh, YmeshOLS,
35       'FaceColor', 'b',  % obarvi povrch na modro
36       'FaceAlpha', 0.5); % nastavi pruhlednost
```

2.3 Metoda LAD

Koeficienty $\boldsymbol{\beta}$ můžeme odhadnout i jiným způsobem než metodou nejmenších čtverců. Další metodou, kterou si představíme, bude tzv. metoda LAD.



Obrázek 2.3: Proměnná $\boldsymbol{y}$ a její odhady metodou nejmenších čtverců v závislosti na $\boldsymbol{x}_1$ a $\boldsymbol{x}_2$.

Metoda LAD

Metoda LAD (Least Absolute Deviations) je statistická a optimalizační technika, která obecně zadanými daty proloží funkci $f_{\boldsymbol{\beta}}$. Metoda hledá závislost proměnné $\boldsymbol{y} = (y_1, \ldots, y_n)'$ na proměnných $\boldsymbol{X} = (x_{i,j})_{i=1,j=1}^{n,m}$ pomocí funkce $f_{\boldsymbol{\beta}}$ s neznámými parametry $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_m)'$. Tyto parametry najdeme optimalizační úlohou

$$\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} \left| y_i - f_{\boldsymbol{\beta}}(x_{i,1}, \ldots, x_{i,m}) \right|. \tag{2.2}$$

Jedná se o minimalizaci součtu absolutních hodnota reziduí. Metoda nejmenších čtverců oproti tomu minimalizuke součet kvadrátů reziduí.

Úloha (2.2) je formulovaná obecně. My budeme řešit úlohu lineární v parametrech tvaru

$$\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} \left| y_i - \beta_1 x_{i,1} + \beta_2 x_{i,2} \cdots + \beta_m x_{i,m} \right|. \tag{2.3}$$

Úloha (2.3) převedeme do tvaru lineárního programování. Absolutní hodnoty v účelové funkci minimalizační úlohy se můžeme zbavit trikem popsaným v následující poznámce.

Poznámka

Nechť máme optimalizační úlohu s absolutní hodnotou v účelové funkci tvaru

$$\min_{z} \quad |z|.$$

Její optimální hodnotu označíme z^*. Dále nechť máme úlohu

$$\begin{aligned} \min_{u,v} \quad & u \\ \text{s.t.} \quad & u \geq v, \\ & u \geq -v. \end{aligned}$$

Její optimální hodnoty označíme u^* a v^*. Ukážeme, že optimální hodnota z^* je rovna v^*. Řešíme dva případy.

- Pokud je v první úloze z kladné, tato úloha se převede na minimalizaci z. Pokud je v druhé úloze v kladné, proměnná u bude zdola omezená v díky první podmínce. Protože u minimalizujeme, optimální u^* bude rovno optimálnímu v^*. Platí tedy $x^* = v^*$.
- Pokud je v první úloze z záporné, tato úloha se převede na minimalizaci $-z$. Pokud je v druhé úloze v záporné, proměnná u bude zdola omezená $-v$ díky druhé podmínce. Protože u minimalizujeme, u^* bude rovno $-v*$. Platí tedy $x^* = v^*$.

Obě úlohy tedy najdou stejnou optimální hodnotu. Přídáním pomocné proměnné u a dvou nerovností do nelineární jsme se zbavili absolutní hodnoty v účelové funkci a vytvořili jsme tak novou úlohu, která již lineární je. Do počáteční úlohy je možné přidat další proměnné a podmínky.

Úlohu (2.3) můžeme převést do tvaru

$$\begin{aligned} \min_{\substack{\beta_1,\ldots,\beta_m,\\ u_1,\ldots,u_n}} \quad & \sum_{i=1}^{n} u_i \\ \text{s.t.} \quad & u_i \geq y_i - \sum_{j=1}^{m} x_{i,j}\beta_j \qquad i = 1,\ldots,n, \\ & u_i \geq -y_i + \sum_{j=1}^{m} x_{i,j}\beta_j \qquad i = 1,\ldots,n, \end{aligned}$$

kterou pak dále převedeme do maticového zápisu

$$\begin{aligned} \min_{\boldsymbol{z}} \quad & \boldsymbol{c}'\boldsymbol{z} \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{z} \leq \boldsymbol{b}, \end{aligned}$$

kde jsme označili vektor proměnných jako $\boldsymbol{z} = (\beta_1,\ldots,\beta_m,u_1,\ldots,u_m)'$ a matici levých stran podmínek, vektor pravých stran podmínek a vektor účelové funkce jako

$$\boldsymbol{A} = \begin{pmatrix} -\boldsymbol{X}_{n\times m} & -\boldsymbol{I}_{n\times n} \\ \boldsymbol{X}_{n\times m} & -\boldsymbol{I}_{n\times n} \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} -\boldsymbol{Y}_{n\times 1} \\ \boldsymbol{Y}_{n\times 1} \end{pmatrix}, \quad \boldsymbol{c} = \begin{pmatrix} \boldsymbol{0}_{m\times 1} \\ \boldsymbol{1}_{n\times 1} \end{pmatrix}, \quad \boldsymbol{I} = \begin{pmatrix} 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & 1 & 0 \\ 0 & 0 & \ldots & 0 & 1 \end{pmatrix}.$$

Maticový zápis použijeme pro solver `linprog` v MATLABu. Metodu LAD si podobně jako metodu nejmenších čtverců naprogramujeme jako funkci.

LAD.m: Metoda LAD

```
38  function betaHat = myLAD(Y, X)
39    [n m] = size(X);
40    c = [zeros(m, 1); ones(n, 1)];
41    A = [-X, -eye(n); X, -eye(n)];
42    b = [-Y; Y];
43    solOpti = linprog(c, A, b);
44    betaHat = solOpti(1:m, 1);
45  end
```

Odhady metodou LAD si označíme jako $\hat{\boldsymbol{\beta}}^{LAD} = (\hat{\beta}_1^{LAD}, \hat{\beta}_2^{LAD}, \hat{\beta}_3^{LAD})'$. Podobně jako v předchozí části dále odhadneme vysvětlovanou proměnnou jako

$$\hat{y}_{i,j}^{LAD} = \hat{\beta}_1^{LAD} + g_{i,j}\hat{\beta}_2^{LAD} + h_{i,j}\hat{\beta}_3^{LAD}.$$

LAD.m: Odhad regresní roviny metodou LAD

```
47  betaLAD = myLAD(Y, X)
48  YmeshLAD = zeros(I, J);
49  for i = 1:I
50    for j = 1:J
51      YmeshLAD(i, j) = betaLAD(1) ...
52                     + X1mesh(i, j) * betaLAD(2) ...
53                     + X2mesh(i, j) * betaLAD(3);
54    end
55  end
56  figure(1);
57  surf(X1mesh, X2mesh, YmeshLAD,
58       'FaceColor', 'r',  % obarvi povrch na cerveno
59       'FaceAlpha', 0.5); % nastavi pruhlednost
```

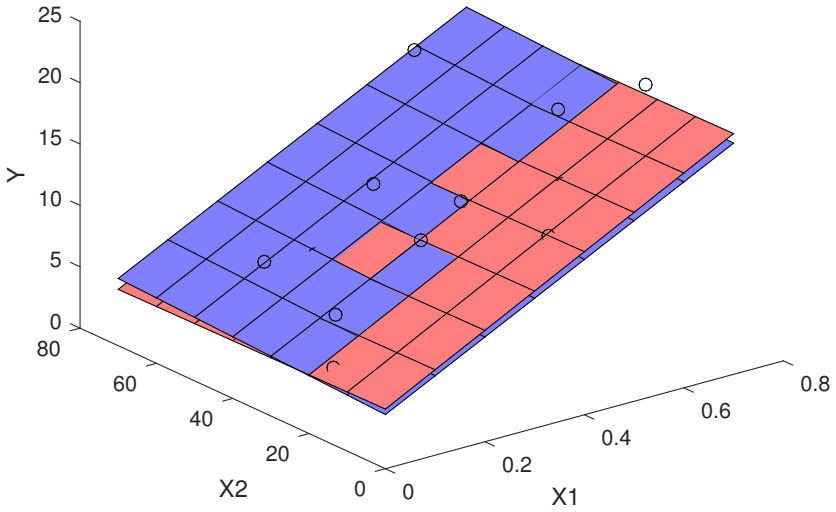
2.4 Citlivost na odlehlá pozorování

K odhadu koeficientů v lineární regresi se převážně používá metoda nejmenších čtverců, ale i metoda LAD má nějaké výhody. Výhodou LAD je robustnost k odlehlým pozorováním. Odlehlé pozorování je takové pozorování, které je velmi vzdálené od ostatních. Může být způsobeno např. chybou měření v daném pozorování. Například, když se při měření výšky lidí k nějakému jedinci zapíše 183 km místo 183 cm. Tato chyba je tak výrazná, že ji lze těžko zachytit náhodnou složkou ε_i.

V této části budeme pro jednoduchost modelovat závislost $\boldsymbol{y}$ pouze na $\boldsymbol{x}_1$, což odpovídá regresnímu modelu

$$y_i = \beta_1 + x_{i,1}\beta_2 + \varepsilon_i, \qquad i = 1, \ldots, n.$$

Pro ilustraci citlivosti na odlehlá pozorování si upravíme vstupní data. Podíváme se, jak se odhlady metodou nejmenších čtverců a metodou LAD budou chovat, pokud hodnotu proměnné y_2 nahradíme nějakým vysokým (a tedy vzdáleným) číslem. Pomocí `for` cyklu dosadíme za y_2 hodnoty od 0 do 40 s krokem jedna. V každé iteraci do grafu vykreslíme regresní přímky.

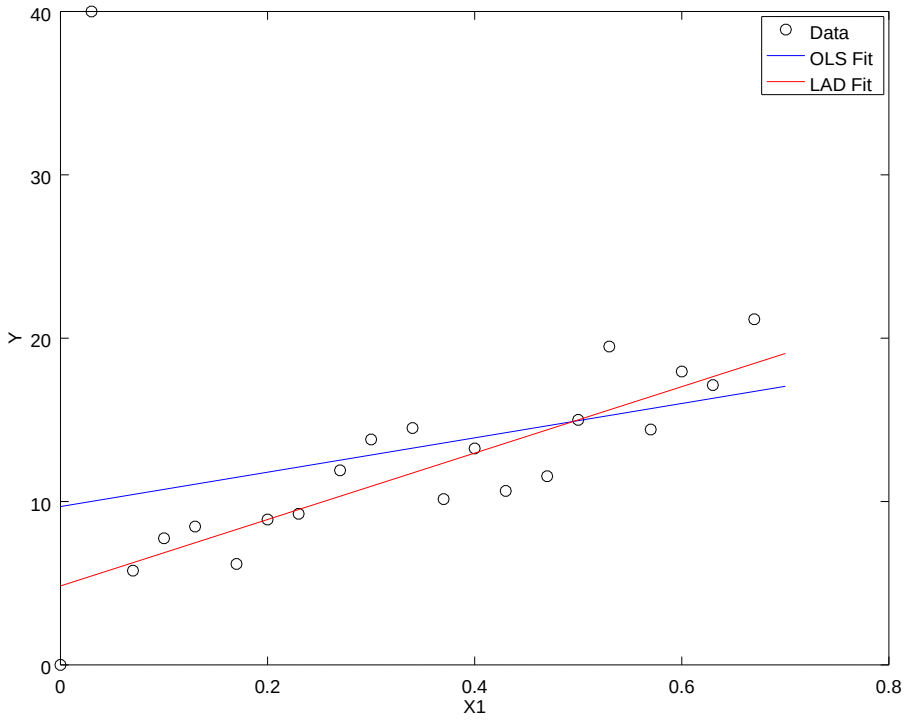


Obrázek 2.4: Proměnná $\boldsymbol{y}$ a její odhady metodou nejmenších čtverců a metodou LAD v závislosti na $\boldsymbol{x}_1$ a $\boldsymbol{x}_2$.

LAD.m: Odhad regresní přímky s odlehlým pozorováním

```
61  figure(2);
62  for obs = 0:40
63    Y = data(:, 1);
64    Y(2) = obs;
65    X = [ones(n, 1) data(:, 2)];
66    betaOLS = myOLS(Y, X);
67    YgridOLS = betaOLS(1) + X1grid * betaOLS(2);
68    betaLAD = myLAD(Y, X);
69    YgridLAD = betaLAD(1) + X1grid * betaLAD(2);
70    plot(X(:,2), Y, 'ok');
71    axis([0.0 0.8 0 40]);
72    hold on;
73    plot(X1grid, YgridOLS, '-b');
74    plot(X1grid, YgridLAD, '-r');
75    xlabel('X1');
76    ylabel('Y');
77    legend('Data','OLS Fit','LAD Fit');
78    hold off;
79    pause(0.1);
80  end
```

Na obrázku 2.5 se odlehlé pozorování s nízkou hodnotou x_1 a vysokou hodnotou y nachází v levé horní části grafu. Vidíme, že metoda nejmenších čtverců silně reaguje na odlehlé pozorování. V extrém-



Obrázek 2.5: Proměnná $\boldsymbol{y}$ a její odhady metodou nejmenších čtverců a metodou LAD v závislosti na $\boldsymbol{x}_1$.

ním studovaném případě $y_2 = 40$ už na první pohled regresní přímka nemodeluje věrně data a nechá se vychýlit odlehlým pozorováním, které mezi data zjevně nezapadá. Naproti tomu odhadnuté koeficienty metodou LAD jsou pro všechny hodnoty y_2 stejné a metoda je tak k odlehlému pozorování robustní. To je způsobeno tím, že metoda LAD dává všem pozorováním stejnou váhu, zatímco metoda nejmenších čtverců klade větší důraz na „vzdálenější“ pozorování.

Poznámka

Robustnost metody LAD k odlehlým pozorováním je velice užitečná vlastnost. Oproti nejmenším čtvercům má ale LAD několik nedostatků. Metoda LAD není stabilní. Může se stát, že malá změna v datech způsobí velkou změnu v řešení (v nalezených koeficientech). Dalším problémem je, že řešení nemusí být jednoznačné.

Doporučená literatura

- Cipra, T. 2014. *Finanční ekonometrie*. Ekopress. ISBN 978-80-86929-93-4. https://www.ekopress.cz/titdetail.php?tid=30238
- Greene, W. H. 2018. *Econometric Analysis*. Pearson. ISBN 978-0-13-446136-6. https://www.pearson.com/us/higher-education/program/Greene-Econometric-Analysis-8th-Edition/PGM334862.html
- James, G., Witten, D., Hastie, T., Tibshirani, R. 2013. *An Introduction to Statistical Learning: With Applications in R*. Springer. ISBN 978-1-4614-7137-0. https://doi.org/10.1007/978-1-4614-7138-7
- Vanderbei, R. J. 2020. *Linear Programming: Foundations and Extensions*. Springer. ISBN 978-3-030-39414-1. https://doi.org/10.1007/978-3-030-39415-8
- Wooldridge, J. M. 2019. *Introductory Econometrics: A Modern Approach*. Cengage Learning. ISBN 978-1-337-55886-0. https://www.cengage.co.uk/books/9781337558860/
- Zvára, K. 2019. *Regrese*. MatfyzPress. ISBN 978-80-7378-406-5. http://matfyzpress.cz/matematika/232-regrese-s-cd.html

Kapitola 3

Základy lineárního programování

V této kapitole si ukážeme základy lineárního programování v MATLABu.

Klíčová slova: *lineární programování, citlivost na změnu dat.*

3.1 Řešení úlohy lineárního programování

Nejdříve se podíváme, jak je úloha lineárního programu definována a jaké nástroje pro její řešení nám Matlab nabízí. Lineární programování budeme používat v mnoha následujících kapitolách.

Lineární programování

Úlohou lineárního programování (LP) je optimalizační úloha tvaru

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \{\boldsymbol{c}'\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}. \tag{3.1}$$

Funkce `linprog`

MATLAB je vybaven toolboxem *Optimization*, kde je k dispozici solver `linprog` pro lineární programování (zdůrazněme, že zde hovoříme výhradně o lineárním programování se spojitými proměnnými; o celočíselném či smíšeném programování budeme hovořit až v kapitole 10). V základní podobě se solver volá ve tvaru

```
xOpt = linprog(c,A,b).
```

Řeší se tak lineární program (LP) tvaru (3.1) a návratovou hodnotou je optimální řešení `xOpt`. Konvencí je, že vektory `c` a `b` se rozumí jako sloupcové; nalezené optimum `xOpt` je rovněž sloupcový vektor.

Poznámka

Jak poznat nepřípustnost (*infeasibility*) či neomezenost (*unboundedness*) LP si vysvětlíme v sekci 7.3 na straně 48.

3.2 Ekvivalentní úpravy lineárního programu

Připomeňme, že (3.1) je zcela obecný tvar LP v tom smyslu, že libovolný LP lze do tvaru (3.1) převést. Vystačíme s několika málo triky:

(a) Je-li cílem řešit LP tvaru $\max\{\boldsymbol{c}\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$, stačí užít vztahu

$$\max\{\boldsymbol{c}'\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\} = -\min\{-\boldsymbol{c}'\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}. \tag{3.2}$$

Maximalizace funkce $f(x)$ je totiž ekvivalentní minimalizaci funkce $-f(x)$. Stačí proto volat `linprog(-c,A,b)`.

(b) Máme-li LP zapsaný ve tvaru s nerovnostmi „$\leq$“ i „$\geq$“, stačí nerovnosti druhého typu přenásobit konstantou -1. Obecně: máme-li řešit lineární program tvaru

$$\min\{\boldsymbol{c}'\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b},\ \boldsymbol{D}\boldsymbol{x} \geq \boldsymbol{h}\}, \tag{3.3}$$

stačí jej přepsat do tvaru

$$\min\left\{\boldsymbol{c}'\boldsymbol{x} \;\middle|\; \begin{pmatrix}\boldsymbol{A}\\ -\boldsymbol{D}\end{pmatrix}\boldsymbol{x} \leq \begin{pmatrix}\boldsymbol{b}\\ -\boldsymbol{h}\end{pmatrix}\right\} \tag{3.4}$$

a volat `linprog(c, [A;-D], [b;-h])`.

(c) Jsou-li mezi omezujícími podmínkami rovnosti, lze postupovat dvojím způsobem. **První způsob** přepíše rovnost $\alpha = \beta$ jako ekvivalentní dvojici nerovností $\alpha \leq \beta$, $-\alpha \leq -\beta$. To znamená: máme-li řešit LP tvaru

$$\min\{\boldsymbol{c}'\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b},\ \boldsymbol{U}\boldsymbol{x} = \boldsymbol{v}\}, \tag{3.5}$$

přepíšeme jej do ekvivalentního tvaru

$$\min\left\{\boldsymbol{c}'\boldsymbol{x} \;\middle|\; \begin{pmatrix}\boldsymbol{A}\\ \boldsymbol{U}\\ -\boldsymbol{U}\end{pmatrix}\boldsymbol{x} \leq \begin{pmatrix}\boldsymbol{b}\\ \boldsymbol{v}\\ -\boldsymbol{v}\end{pmatrix}\right\} \tag{3.6}$$

a voláme `linprog(c, [A;U;-U], [b;v;-v])`. **Druhý způsob** využívá možnosti volat funkci `linprog` v rozšířené syntaxi oproti základnímu tvaru (3.1): příkaz

```
linprog(c, A, b, U, v)
```

přímo řeší LP tvaru (3.5). Oba způsoby jsou ekvivalentní a záleží jen na preferenci uživatele, který způsob zvolí.

Poznámka

Syntaxe příkazu `linprog` je ještě obecnější (pro detaily je vhodné se podívat do dokumentace `doc linprog`). Například je možné pomocí dalších parametrů zadávat horní a/nebo dolní meze proměnných (což se hodí například při práci s nezápornými proměnnými), je možné zadat počáteční nástřel a je možné nastavovat řadu parametrů solveru, například lze volit algoritmus k řešení LP, numerickou citlivost atd. Za zmínku stojí varianta

$$\texttt{linprog(c, A, b, U, v, } \underline{x}\texttt{, } \overline{x}\texttt{)},$$

která řeší LP tvaru

$$\min_{\boldsymbol{x}}\{\boldsymbol{c}'\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b},\ \boldsymbol{U}\boldsymbol{x} = \boldsymbol{v},\ \underline{\boldsymbol{x}} \leq \boldsymbol{x} \leq \overline{\boldsymbol{x}}\}. \tag{3.7}$$

Nicméně my se zde nebudeme těmito možnostmi dále zabývat a vesměs vystačíme jen se základním tvarem (3.1) a (3.5). Budeme-li například potřebovat omezit proměnné shora/zdola, prostě tato omezení zařadíme mezi omezení systému $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ (ale upozorňujeme, že to není jediná možnost). Variantu (3.7) použijeme — kvůli stručnosti zápisu — až v kapitole 10, byť i tam lze vše udělat jen s (3.1) a/nebo (3.5).

3.3 Jednoduchý příklad

Řešme LP

$$\begin{aligned} \max\quad & 3x_1 + 2x_2 \\ \text{s.t.}\quad & x_1 + 2x_2 \leq 2, \\ & 4x_1 + 2x_2 \leq 3, \\ & x_1 + x_2 \geq 1, \\ & x_1, x_2 \geq 0. \end{aligned}$$

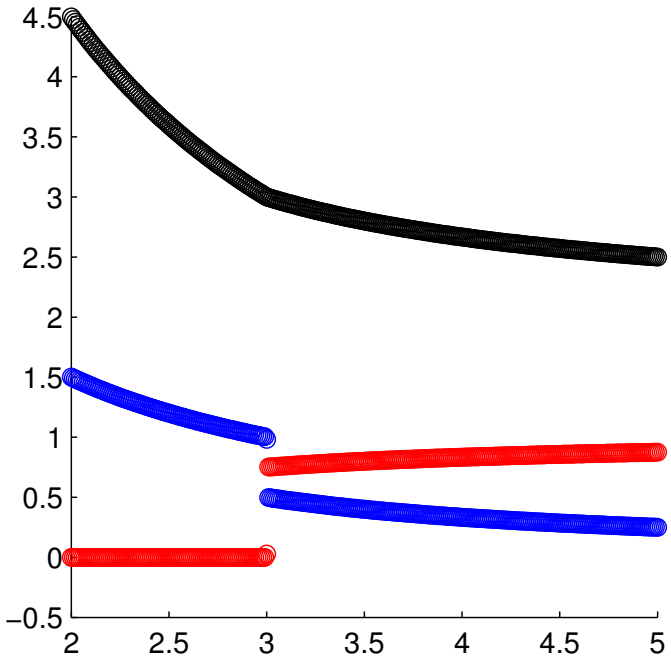
(zde „s.t.“ značí „subject to“, „za omezujících podmínek“; je to běžný žargon). Dle triku (b) z kapitoly 3.2 přenásobíme nerovnosti typu „$\geq$“ konstantou -1 a a dle triku (a) převedeme maximalizaci na minimalizaci:

$$\min \underbrace{(-3,-2)}_{c}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad \text{s.t.} \underbrace{\begin{pmatrix} 1 & 2 \\ 4 & 2 \\ -1 & -1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix}}_{A}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \underbrace{\begin{pmatrix} 2 \\ 3 \\ -1 \\ 0 \\ 0 \end{pmatrix}}_{b}. \tag{3.8}$$

Všimněme si, že podmínkám $x_1, x_2 \geq 0$ odpovídá blok $-\boldsymbol{I}\boldsymbol{x} \leq 0$ v systému nerovností $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$. Zde $\boldsymbol{I}$ je jednotková matice. Můžeme proto například napsat následující skript, pomocí kterého zjistíme, že optimum jest $\boldsymbol{x}^* = \binom{0.333}{0.833}$.

`LinProg.m`: Řešení lineárního programování

```
3  c = [-3; -2];
4  A = [1, 2; 4, 2; -1, -1; -eye(2)];
5  b = [2; 3; -1; zeros(2, 1)];
6  xOpt = linprog(c, A, b)
```



Obrázek 3.1: Citlivost na změnu dat.

3.4 Citlivost na změnu dat

Je poučné si rovněž vyzkoušet jednoduchou analýzu citlivosti na změny dat lineárních programů. V lineárním programování totiž data A, b, c mívají věcný význam a bývá relevantní otázka, jak by se změnilo optimální řešení a/nebo optimální hodnota účelové funkce, kdyby se některý koeficient změnil. Například v případě tzv. dietního problému koeficienty matice A říkají, jaký je obsah živin v potravinách. A často je na místě se ptát, jak by se změnilo optimum, kdyby dodávka potravin měla (mírně) odlišný obsah živin, než je deklarováno.

V tomto příkladu vyjděme z LP (3.8) a položme si otázku, jak by se měnilo optimum x_1^*, x_2^* a optimální hodnota účelové funkce, kdybychom namísto pevné hodnoty $A_{21} = 4$ ji považovali za parametr $A_{21} = \alpha$, kde α probíhá interval (například) $[2, 5]$. Ptáme se vlastně na chování parametrického lineárního programu

$$\left\{ \min_{\boldsymbol{x}\in\mathbb{R}^2} (-3,-2)\begin{pmatrix}x_1\\x_2\end{pmatrix} \quad \text{s.t.} \begin{pmatrix}1 & 2\\ \alpha & 2\\ -1 & -1\\ -1 & 0\\ 0 & -1\end{pmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix} \le \begin{pmatrix}2\\3\\-1\\0\\0\end{pmatrix} \right\}, \qquad \alpha \in [2,5].$$

Pak optima $x_1^*(\alpha), x_2^*(\alpha)$ i optimální hodnota účelové funkce $-cx^*(\alpha)$ jsou funkce α a můžeme si nakreslit graf jejich závislosti na α (α je na horizontální ose). Hodnoty $x_1^*(\alpha)$ kreslíme modře, hodnoty $x_2^*(\alpha)$ kreslíme červeně a optimální hodnotu účelové funkce kreslíme černě.

Všimněme si, že z obrázku je patrná změna báze (bod nespojistosti $x_1^*(\alpha)$ a $x_2^*(\alpha)$) pro $\alpha = 3$. Interpretujeme-li hodnoty α jako *možné scénáře* (například: až nám přivezou potraviny, uvidíme, jaký je skutečný obsah živin α, nyní to ovšem nevíme) a interpretujeme-li účelovou funkci jako ziskovou funkci, z obrázku například zjistíme: budeme-li mít štěstí a nastane-li nejlepší možný scénář $\alpha = 2$, budeme

mít nejvyšší možný zisk 4.5. Zatímco nastane-li nejhorší možný scénář $\alpha = 5$, musíme počítat pouze se ziskem ≈ 2.7. Vlastně jsme „vyřešili“ dva nelineární optimalizační problémy

$$\min_{\alpha\in[2,5]} \min_{\boldsymbol{x}\in\mathbb{R}^2} (-3,-2)\begin{pmatrix}x_1\\x_2\end{pmatrix} \quad \text{s.t.} \begin{pmatrix}1 & 2\\ \alpha & 2\\ -1 & -1\\ -1 & 0\\ 0 & -1\end{pmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix} \leq \begin{pmatrix}2\\3\\-1\\0\\0\end{pmatrix},$$

$$\max_{\alpha\in[2,5]} \min_{\boldsymbol{x}\in\mathbb{R}^2} (-3,-2)\begin{pmatrix}x_1\\x_2\end{pmatrix} \quad \text{s.t.} \begin{pmatrix}1 & 2\\ \alpha & 2\\ -1 & -1\\ -1 & 0\\ 0 & -1\end{pmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix} \leq \begin{pmatrix}2\\3\\-1\\0\\0\end{pmatrix}.$$

Problémy takového typu se v rámci parametrického programování zkoumají často.

Obrázek snadno vygenerujeme skriptem, kde uvnitř `for`-cyklu postupně měníme hodnotu A_{21} v intervalu $[2, 5]$ s rozumně malým krokem (zde volíme krok $1/100$), v každé iteraci voláme solver `linprog` a výsledky vykreslíme do grafu.

LinProg.m: Změna dat

```
8  figure;
9  hold on;
10 for alpha = 2:0.01:5
11   A(2, 1) = alpha;
12   x = linprog(c, A, b);
13   plot(alpha, x(1), 'bo', alpha, x(2), 'ro', ...
14     alpha, -c' * x, 'ko');
15 end
```

Doporučená literatura

- Dupačová, J., Lachout, P. 2011. *Úvod do optimalizace*. MatfyzPress. ISBN 978-80-7378-176-7. http://matfyzpress.cz/matematika/20-uvod-do-optimalizace.html
- Hillier, F. S., Lieberman, G. J. 2018. *Introduction to Operations Research*. McGraw-Hill Education. ISBN 978-0-07-352345-3. http://highered.mheducation.com/sites/0073523453/information{_}center{_}view0/index.html
- Jablonský, J. 2007. *Operační výzkum: Kvantitativní modely pro ekonomické rozhodování*. Professional Publishing. ISBN 978-80-86946-44-3. https://www.researchgate.net/publication/40354299
- Kwon, R. H. 2013. *Introduction to Linear Optimization and Extensions with MATLAB*. CRC Press. ISBN 978-1-4398-6264-3. https://doi.org/10.1201/b13966
- Vanderbei, R. J. 2020. *Linear Programming: Foundations and Extensions*. Springer. ISBN 978-3-030-39414-1. https://doi.org/10.1007/978-3-030-39415-8

Kapitola 4

Maticové hry

V této kapitole se budeme zabývat maticovými hrami. Nejdříve si ukážeme, jak pomocí lineárního programování nalézt nashovské strategie a cenu smíšeného rozšíření maticové hry zadané pomocí výplatní matice. Dále tento postup aplikujeme na několika příkladech. Budeme řešit známou hru Kámen-nůžky-papír, starobylou hazardní hru Morra a hru plukovníka Blotto, v níž se modeluje konflikt dvou armád.

Klíčová slova: *teorie her, maticové hry, hra Kámen-nůžky-papír, hra Morra, hra plukovníka Blotto.*

4.1 Řešení maticových her pomocí lineárního programování

V teorii maticových her je třeba řešit lineární programy tvaru

$$\begin{aligned} \max_{\substack{\gamma\in\mathbb{R}\\ \boldsymbol{\xi}\in\mathbb{R}^n}} \quad & \gamma \\ \text{s.t.} \quad & \boldsymbol{P}'\boldsymbol{\xi} \geq \gamma \boldsymbol{e}, \\ & \boldsymbol{e}'\boldsymbol{\xi} = 1, \\ & \boldsymbol{\xi} \geq \boldsymbol{0}, \end{aligned} \tag{4.1}$$

kde $\boldsymbol{P}$ je zadaná matice rozměru $(n \times m)$ a $\boldsymbol{e} = (1, \ldots, 1)'$. (Abychom předešli nejasnostem: γ je skalár — proměnná, která může nabývat libovolných reálných hodnot (i záporných) — a γe je vektor $(\gamma, \ldots, \gamma)'$.) Získáme-li optimum $(\gamma^*, \boldsymbol{\xi}^*)$, pak $\boldsymbol{\xi}^*$ je nashovská strategie pro prvního hráče ve smíšeném rozšíření maticové hry s výplatní maticí $\boldsymbol{P}$ („payoff“ — odtud symbol $\boldsymbol{P}$) a γ^* je cena této hry.

Poznámka

Vyřešíme-li duál k (4.1), získáme nashovskou strategii pro druhého hráče; to zde ovšem nebudeme činit, konstrukci duálu a jeho řešení ponecháváme jako cvičení.

S pomocí triků (a) – (c) z kapitoly 3.2 přepišme LP (4.1) do ekvivalentního tvaru

$$\begin{aligned} \min_{\gamma,\boldsymbol{\xi}} \quad & -\gamma \\ \text{s.t.} \quad & \gamma \boldsymbol{e} - \boldsymbol{P}'\boldsymbol{\xi} \leq \mathbf{0} \\ & \boldsymbol{e}'\boldsymbol{\xi} \leq 1, \\ & -\boldsymbol{e}'\boldsymbol{\xi} \leq -1, \\ & -\boldsymbol{I}\boldsymbol{\xi} \leq \mathbf{0} \end{aligned} \tag{4.2}$$

a uspořádejme proměnné $\gamma, \boldsymbol{\xi}$ do společného vektoru $\boldsymbol{x}$ například v pořadí $\boldsymbol{x} = \binom{\gamma}{\boldsymbol{\xi}}$. Získáme maticový tvar

$$\min_{\boldsymbol{x}=\binom{\gamma}{\boldsymbol{\xi}}} \underbrace{(-1, \mathbf{0}'_{1\times n})}_{\boldsymbol{c}'} \underbrace{\begin{pmatrix}\gamma \\ \boldsymbol{\xi}\end{pmatrix}}_{\boldsymbol{x}} \quad \text{s.t.} \quad \underbrace{\begin{pmatrix} \boldsymbol{e}_{m\times 1} & -\boldsymbol{P}'_{m\times n} \\ 0_{1\times 1} & \boldsymbol{e}'_{1\times n} \\ 0_{1\times 1} & -\boldsymbol{e}'_{1\times n} \\ \mathbf{0}_{n\times 1} & -\boldsymbol{I}_{n\times n} \end{pmatrix}}_{\boldsymbol{A}} \underbrace{\begin{pmatrix}\gamma \\ \boldsymbol{\xi}\end{pmatrix}}_{\boldsymbol{x}} \leq \underbrace{\begin{pmatrix} \mathbf{0}_{m\times 1} \\ 1_{1\times 1} \\ -1_{1\times 1} \\ \mathbf{0}_{n\times 1} \end{pmatrix}}_{\boldsymbol{b}};$$

pro přehlednost jsme explicitně uvedli rozměry vektorů a matic. Převedení na tento tvar a volání solveru `linprog` si napíšeme jako funkci, která dostane na vstup výplatní matici $\boldsymbol{P}$ a vrátí cenu hry γ a vektor nashovských strategií $\boldsymbol{\xi}$; bude to náš „univerzální game-solver“.

`RockPaperScissors.m`: Funkce pro řešení maticových her

```
3  function [gamma, xi] = GameSolver(P)
4    [n, m] = size(P);
5    c = [-1; zeros(n, 1)];
6    A = [ones(m, 1), -P';
7         0, ones(1, n);
8         0, -ones(1, n);
9         zeros(n, 1), -eye(n)];
10   b = [zeros(m, 1); 1; -1; zeros(n, 1)];
11   x = linprog(c, A, b);
12   gamma = x(1);
13   xi = x(2:(n+1));
14 end
```

Poznámka

Lze postupovat i takto: namísto tvaru (3.1) zvlášť napíšeme nerovnosti a zvlášť rovnosti (viz (3.5)), čímž získáme

$$\min_{\gamma,\boldsymbol{\xi}} -\gamma \quad \text{s.t.} \quad \underbrace{\gamma\boldsymbol{e} - \boldsymbol{P}'\boldsymbol{\xi} \leq \mathbf{0},\ -\boldsymbol{I}\boldsymbol{\xi} \leq \mathbf{0}}_{\text{nerovnosti}},\ \underbrace{\boldsymbol{e}'\boldsymbol{\xi} = 1}_{\text{rovnosti}},$$

anebo totéž v maticovém tvaru

$$\min_{\boldsymbol{x}=\binom{\gamma}{\boldsymbol{\xi}}} \underbrace{(-1, \mathbf{0}'_{1\times n})}_{\boldsymbol{c}'} \underbrace{\begin{pmatrix}\gamma\\ \boldsymbol{\xi}\end{pmatrix}}_{\boldsymbol{x}} \quad \text{s.t.} \quad \underbrace{\begin{pmatrix}\boldsymbol{e}_{m\times 1} & -\boldsymbol{P}'_{m\times n}\\ \mathbf{0}_{n\times 1} & -\boldsymbol{I}_{n\times n}\end{pmatrix}}_{\boldsymbol{A}} \underbrace{\begin{pmatrix}\gamma\\ \boldsymbol{\xi}\end{pmatrix}}_{\boldsymbol{x}} \leq \underbrace{\begin{pmatrix}\mathbf{0}_{m\times 1}\\ \mathbf{0}_{n\times 1}\end{pmatrix}}_{\boldsymbol{b}},\quad \underbrace{(0, \boldsymbol{e}'_{1\times n})}_{\boldsymbol{U}} \underbrace{\begin{pmatrix}\gamma\\ \boldsymbol{\xi}\end{pmatrix}}_{\boldsymbol{x}} = \underbrace{(1)}_{\boldsymbol{v}}.$$

Voláme `linprog` ve tvaru (3.5). Funkce by tedy vypadala následovně.

```
[n,m] = size(P);
c = [-1; zeros(n,1)];
A = [ones(m,1), -P'; zeros(n,1), -eye(n)];
b = zeros(m+n,1);
U = [0, ones(1,n)];
v = 1;
x = linprog(c,A,b,U,v);
gamma = x(1);
xi = x(2:n+1);
```

4.2 Hra Kámen-nůžky-papír

Uvažme pro příklad hru Kámen-nůžky-papír s výplatní maticí

$$P = \begin{pmatrix} 0 & 1 & -1\\ -1 & 0 & 1\\ 1 & -1 & 0\end{pmatrix}.$$

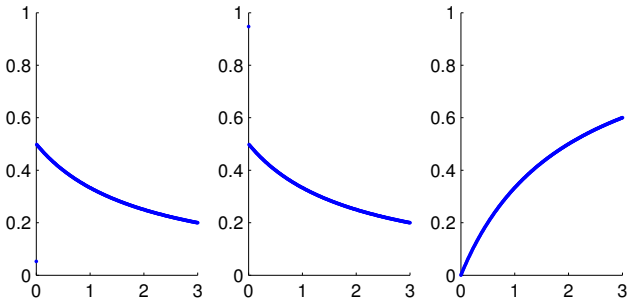
`RockPaperScissors.m`: Řešení hry Kámen-nůžky-papír

```
16 P=[ 0,  1, -1;
17    -1,  0,  1;
18     1, -1,  0];
19 [gamma, xi] = GameSolver(P)
```

Získáme výstup:

```
Cena hry = 1.4211e-014
Strategie = 0.33333    0.33333    0.33333
```

Cena hry vyšla řádu 10^{-14}, což je numerická nula. Kámen-nůžky-papír je symetrická hra, jež má nutně nulovou cenu. Nashovská strategie pro prvního hráče (a ze symetrie hry také pro druhého) je $\boldsymbol{\xi}^* = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})'$; všechny tři strategie — Kámen, nůžky, papír — se mají hrát se stejnou pravděpodobností.



Obrázek 4.1: Závislost ceny na jednotlivých parametrech modifikované hry Kámen-nůžky-papír.

4.3 Vizualizace změny jednoho parametru

I s maticovou hrou si lze „hrát“ podobně jako v kapitole 3.4 a obvykle takové hraní pomáhá při porozumění chování studovaného problému. Zde si položme otázku, jak by se změnily nashovské strategie ve hře Kámen-nůžky-papír, kdyby různé situace byly nestejně oceněné. Uvažme například výplatní matici

$$\boldsymbol{P}(\alpha) = \begin{pmatrix} 0 & \alpha & -1 \\ -\alpha & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix} \tag{4.3}$$

závisející na parametru $\alpha > 0$; řekněme pro příklad, že α probíhá interval $[0, 3]$. Jedná se o modifikaci hry Kámen-nůžky-papír pro situaci, kdy dvojice strategií kámen-nůžky má výplatu α, obecně odlišnou od 1. Jedná se stále o symetrickou hru, takže cena hry je nulová; nicméně nashovské strategie $\xi_1^*(\alpha), \xi_2^*(\alpha), \xi_3^*(\alpha)$ jistě na parametru α závisejí. Jak? Bylo by možné se pokusit odvodit tuto závislost analyticky; my si zde jen numericky nakreslíme obrázek. Stačí volat `GameSolver` uvnitř `for`-cyklu probíhajícího $\alpha \in [0, 3]$ s dostatečně malým krokem (zde zvolíme například $1/100$) a vykreslit výstup.

RockPaperScissors.m: Změna jednoho parametru v Kámen-nůžky-papír

```
figure;
for alpha = 0:0.01:3
  P(1, 2) = alpha; % zmenime hodnoty P12 a P21
  P(2, 1) = -alpha;
  [gamma, xi] = GameSolver(P);
  for i = 1:3 % vizualizace strategie
    % i=1 (=K), 2 (=N), 3 (=P)
    subplot(1, 3, i);
    hold on;
    plot(alpha, xi(i), '.'); % vykresli i-tou
      % strategii proti alpha
    axis([0 3 0 1]); % nastaveni os
  end
end
```

Na vodorovné ose je ve všech třech případech $\alpha \in [0, 3]$. Na prvním obrázku jsou hodnoty $\xi_1^*(\alpha)$ — tedy pravděpodobnosti, s nimiž se má volit strategie Kámen, v závislosti na α. (A analogicky druhý a třetí obrázek zobrazuje Nůžky $\xi_2^*(\alpha)$ a Papír $\xi_3^*(\alpha)$). Je vidět, že při $\alpha = 0$ se strategie Papír nemá hrát

vůbec; její váha (pravděpodobnost) roste s hodnotou α, zatímco váha strategií Kámen a Nůžky klesá, a to stejným tempem.

Poznámka

Nyní, po přehlédnutí techto obrázků, je vhodné se pokusit tuto závislost zdůvodnit analyticky: jednak kvalitativně vysvětlit trend, a jednak se pokusit najít explicitní formuli pro funkce $\xi_i^*(\alpha)$, $i = 1, 2, 3$.

4.4 Vizualizace změny dvou parametrů

Úvahy z předchozího textu můžeme samozřejmě zkombinovat i s dalšími nástroji MATLABu: například s 3D vizualizací. Uvažme modifikaci hry Kámen-nůžky papír s dvěma parametry

$$\boldsymbol{P}(\alpha, \beta) = \begin{pmatrix} 0 & \alpha & -1 \\ -1 & 0 & 1 \\ \beta & -1 & 0 \end{pmatrix}, \quad \alpha \in [-2, 2],\ \beta \in [-2, 2]. \tag{4.4}$$

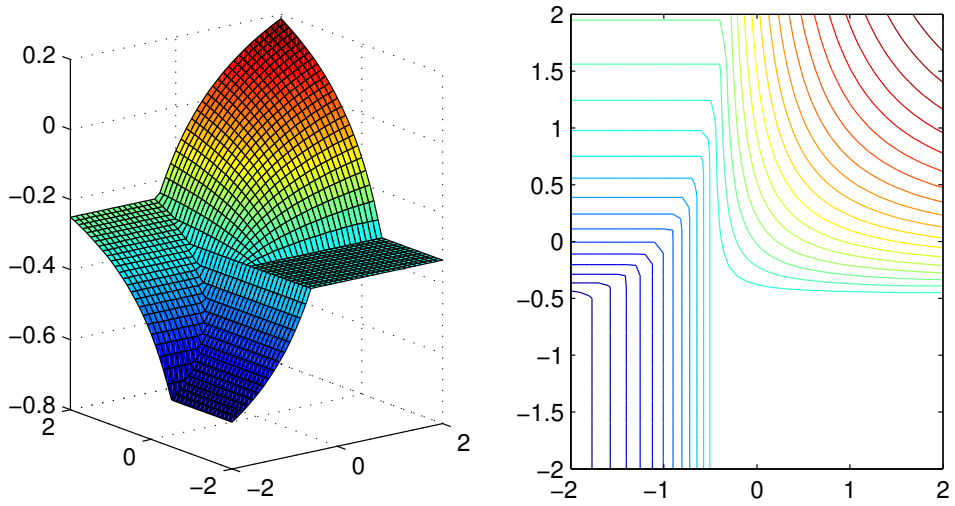
Toto již není symetrická hra; má proto smysl se ptát, jak závisí cena hry γ^* na hodnotách α, β. V levém obrázku vykreslíme cenu hry na svislé ose v závislosti na α, β (horizontální osy) jako trojrozměný graf pomocí `surf`; v pravém obrázku vykreslíme tutéž funkci v prostoru (α, β) pomocí 30 vrstevnic funkcí `contour`. Připravíme si `meshgrid`: pokryjeme čtverec $[-2, 2] \times [-2, 2]$ v (α, β)-prostoru sítí bodů s krokem (například) 0.1; pak v každém bodě sítě spočteme pomocí již vytvořeného `GameSolveru` cenu hry γ. Tu si uložíme v tabulce g. Nakonec vykreslíme hodnoty v g proti bodům sítě.

`RockPaperScissors.m`: Změna dvou parametrů v Kámen-nůžky-papír

```
figure;
[alpha, beta] = meshgrid(-2:0.1:2, -2:0.1:2);
[k, l] = size(alpha);
for i = 1:k
  for j = 1:l
    P(1, 2) = alpha(i, j);
    P(3, 1) = beta(i, j);
    [gamma, xi] = GameSolver(P);
    g(i, j) = gamma;
  end
end
subplot(1, 2, 1);
surf(alpha, beta, g);
subplot(1, 2, 2);
contour(alpha, beta, g, 30);
```

4.5 Hra Morra

Často se stává, že výplatní matice $\boldsymbol{P}$ není zadána explicitně; namísto toho je dána sada pravidel, která umožňují prvek P_{ij} dopočítat na základě (i, j). Tak se činí především tehdy, je-li tento popis kratší a transparentnější než explicitní výčet všech prvků (často hodně velké) výplatní matice. Pak je třeba



Obrázek 4.2: Závislost ceny na dvou parametrech modifikované hry Kámen-nůžky-papír.

napsat skript, který na základě pravidel matici $\boldsymbol{P}$ sestaví. Taková je situace u hry Morra, kterou se budeme zabývat v této části.

Pravidla hry Morra jsou následující. Každý ze dvou hráčů tajně ukáže jeden až n prstů (většinou se hraje tříprstá Morra, tedy $n = 3$). Dále oba hráči hádají, kolik prstů ukázal protivník. Pokud oba hráči uhádnou, kolik prstů ukazuje protivník, nebo to oba neuhádnou, nic se neděje (výplata obou hráčů je 0). Pokud ovšem jeden z hráčů uhádl a druhý ne, pak první vyhrál částku rovnající se součtu zdvižených prstů obou hráčů. Každý hráč může ukázat 1 až n prstů a zároveň hádat 1 až n prstů, má tedy celkem n^2 možných strategií.

Matici $\boldsymbol{P}$ sestavíme pomocí funkce pro obecné n. Nashovské strategie a cenu hry pak nalezneme pomocí našeho solveru `GameSolver`.

Morra.m: Výplatní matice hry Morra

```
function X = morra(fingers)
  X = zeros(fingers^2, fingers^2);
  for showA = 1:fingers % pocet prstu, ktere ukazuje A
    for guessA = 1:fingers % pocet prstu, ktere hada A
      for showB = 1:fingers % pocet prstu, ktere ukazuje B
        for guessB = 1:fingers % pocet prstu, ktere hada B
          i = fingers * (showA - 1) + guessA; % radek
          j = fingers * (showB - 1) + guessB; % sloupec
          if (guessA == showB) && (guessB ~= showA)
            X(i, j) = showA + showB; % hrac A vyhral
          end
          if (guessA ~= showB) && (guessB == showA)
            X(i, j) = -showA - showB; % hrac B vyhral
          end
        end
      end
    end
  end
end
```

Morra.m: Řešení hry Morra

```
P = morra(3)
[gamma, xi] = GameSolver(P)
```

Výplatní matice má pro $n = 3$ tvar

$$\boldsymbol{P} = \begin{array}{c} \begin{matrix} (1,1) & (1,2) & (1,3) & (2,1) & (2,2) & (2,3) & (3,1) & (3,2) & (3,3) \end{matrix} \\ \begin{pmatrix} 0 & 2 & 2 & -3 & 0 & 0 & -4 & 0 & 0 \\ -2 & 0 & 0 & 0 & 3 & 3 & -4 & 0 & 0 \\ -2 & 0 & 0 & -3 & 0 & 0 & 0 & 4 & 4 \\ 3 & 0 & 3 & 0 & -4 & 0 & 0 & -5 & 0 \\ 0 & -3 & 0 & 4 & 0 & 4 & 0 & -5 & 0 \\ 0 & -3 & 0 & 0 & -4 & 0 & 5 & 0 & 5 \\ 4 & 4 & 0 & 0 & 0 & -5 & 0 & 0 & -6 \\ 0 & 0 & -4 & 5 & 5 & 0 & 0 & 0 & -6 \\ 0 & 0 & -4 & 0 & 0 & -5 & 6 & 6 & 0 \end{pmatrix} \end{array} \begin{matrix} (1,1) \\ (1,2) \\ (1,3) \\ (2,1) \\ (2,2) \\ (2,3) \\ (3,1) \\ (3,2) \\ (3,3) \end{matrix} ,$$

kde jsme zápisem (i, j) označili strategii, ve které hráč ukazuje i prstů a hádá j prstů.

4.6 Hra plukovníka Blotto

Další maticovou hrou pro dva hráče je hra plukovníka Blotto. Hráč A má k dispozici daný počet pluků n_A a hráč B má k dispozici počet pluků n_B (obecně se mohou počty pluků u hráčů lišit). Oba hráči spolu bojují na dvou polích, které si můžeme označit bitva 1 a bitva 2. Každý z hráčů se rozhodne, kolik pluků pošle do obou bitev. Do každé bitvy musí poslat nejméně 1 pluk a bojovat musí všechny jeho

pluky. Armádu lze rozdělit jen na celé pluky. Bitvy se vyhodnocují zvlášť a výsledná výplata je součtem výplat obou bitev. Pokud se na poli potkají armády se stejným počtem pluků, navzájem se zničí a výplata obou hráčů je 0. Pokud jeden z hráčů má větší počet pluků, zničí armádu druhého a získá výplatu rovnou počtu zničených pluků. Druhý hráč má pak výplatu rovnou minus počet zničených pluků.

Matici $\boldsymbol{P}$ sestavíme pomocí funkce s parametry velikost armády hráče A a velikost armády hráče B. Nashovské strategie a cenu hry pak opět nalezneme pomocí našeho solveru `GameSolver`.

`Blotto.m`: Výplatní matice hry plukovníka Blotto

```
16 function X = blotto(armyA, armyB)
17   X = zeros(armyA - 1, armyB - 1);
18   for battle1A = 1:(armyA-1) % pocet pluku ...
19     % hrace A v bitve 1
20     for battle1B = 1:(armyB-1) % pocet pluku ...
21       % hrace B v bitve 1
22       battle2A = armyA - battle1A; % pocet pluku ...
23         % hrace A v bitve 2
24       battle2B = armyB - battle1B; % pocet pluku ...
25         % hrace B v bitve 2
26       victory1 = (battle1A > battle1B) * battle1B - ...
27         (battle1A < battle1B) * battle1A; % v pripade ...
28         % vitezstvi A pocet znicenych pluku B, ...
29         % v pripade vitezstvi B minus znicenych ...
30         % pluku A v bitve 1
31       victory2 = (battle2A > battle2B) * battle2B - ...
32         (battle2A < battle2B) * battle2A; % v pripade ...
33         % vitezstvi A pocet znicenych pluku B, ...
34         % v pripade vitezstvi B minus znicenych ...
35         % pluku A v bitve 2
36       X(battle1A, battle1B) = victory1 + victory2;
37     end
38   end
39 end
```

`Blotto.m`: Řešení hry plukovníka Blotto

```
41 P=blotto(6, 5)
42 [gamma, xi] = GameSolver(P)
```

Výplatní matice má pro $n_A = 6$ a $n_B = 5$ tvar

$$\boldsymbol{P} = \begin{array}{c} \begin{matrix} (4,1) & (3,2) & (2,3) & (1,4) \end{matrix} \\ \begin{pmatrix} 4 & 2 & 1 & 0 \\ 1 & 3 & 0 & -1 \\ -2 & 2 & 2 & -2 \\ -1 & 0 & 3 & 1 \\ 0 & 1 & 2 & 4 \end{pmatrix} \end{array} \begin{matrix} (5,1) \\ (4,2) \\ (3,3) \\ (2,4) \\ (1,5) \end{matrix},$$

kde jsme zápisem (i, j) označili strategii, ve které hráč pošle i pluků do první bitvy a j pluků do druhé bitvy.

Doporučená literatura

- Anděl, J. 2007. *Matematika náhody*. MatfyzPress. ISBN 978-80-7378-004-3. http://matfyzpress.cz/matematika/12-matematika-nahody.html
- Anděl, J. 2018. *Statistické úlohy, historky a paradoxy*. MatfyzPress. ISBN 978-80-7378-360-0. http://matfyzpress.cz/vsechny-tituly/178-statisticke-ulohy-historky-a-paradoxy.html
- Fudenberg, D., Tirole, J. 1991. *Game Theory*. MIT Press. ISBN 978-0-262-06141-4. https://mitpress.mit.edu/books/game-theory
- Vanderbei, R. J. 2020. *Linear Programming: Foundations and Extensions*. Springer. ISBN 978-3-030-39414-1. https://doi.org/10.1007/978-3-030-39415-8
- Neumann, J., Morgenstern, O. 2004. *Theory of Games and Economic Behavior: 60th Anniversary Commemorative Edition*. Princeton University Press. ISBN 978-0-691-11993-9. https://press.princeton.edu/books/paperback/9780691130613/theory-of-games-and-economic-behavior

Kapitola 5

Toky v sítích

Budeme se zabývat úlohu o tocích v síti, ve které budeme hledat maximální možných objem, který systémem potrubí může protéci. Tento problém se dá formulovat jako úloha lineárního programování.

Klíčová slova: *toky v sítích, lineární programování.*

5.1 Řešení pomocí lineárního programování

Úloha maximálního toku v síti

Buď dána síť s množinou vrcholů $V = \{1, \ldots, n\}$ a množinou (orientovaných) hran E s nezáporným ohodnocením (tzv. *kapacitou*) k_e ($e \in E$). Připomeňme, že v síti je rozlišena dvojice speciálních vrcholů, tzv. *zdroj* (vrchol s nulovým vstupním stupněm) a *cíl* (vrchol s nulovým výstupním stupněm). Řekněme, že zdrojem je vrchol 1 a cílem je vrchol n. Cílem je najít maximální tok mezi zdrojem a cílem: jedná se o ohodnocení hran x_e ($e \in E$) takové, že

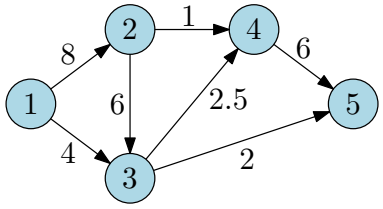
(i) pro každou hranu e jest $0 \leq x_e \leq k_e$ („kapacity nelze překročit“, tzv. *kapacitní podmínky*);

(ii) pro každý vrchol v různý od zdroje a cíle platí $\sum_{e \in \text{in}(v)} x_e = \sum_{e \in \text{out}(v)} x_e$ („součet přítoků je stejný jako součet odtoků — ve vrcholu v se nic neztrácí“, tzv. *tokové podmínky*);

(iii) veličina $\sum_{e \in \text{in}(n)} x_e$ je maximální („celkový přítok do cíle je maximální možný“).

Symbolem $\text{in}(v)$ jsme označili množinu hran vstupujících do vrcholu v a symbolem $\text{out}(v)$ jsme označili množinu hran vystupujících z vrcholu v.

Podmínky (i) – (iii) jsou lineární v x_e, a tak je snadné tento problém zformulovat jako lineární



Obrázek 5.1: Příklad možné sítě, kde vrchol 1 je zdroj a vrchol $n = 5$ je cíl.

program. Za každou hranu $e \in E$ zařadíme proměnnou x_e a můžeme psát

$$\max_{x_e, e\in E} \sum_{e\in \mathrm{in}(n)} x_e \text{ s.t. } \underbrace{0 \le x_e \le k_e \ (\forall e \in E)}_{(\star)}, \underbrace{\sum_{e\in \mathrm{in}(v)} x_e = \sum_{e\in \mathrm{out}(v)} x_e \ (\forall v \in \{2,\dots,n-1\})}_{(\dagger)}. \tag{5.1}$$

Podmínky $(\star)$ jsou kapacitní omezení (i) a podmínky $(\dagger)$ jsou tokové podmínky (ii).

Uvažme příklad sítě na obrázku:

Proměnné x_e $(e \in E)$ uspořádejme do vektoru $\boldsymbol{x}$ např. v pořadí $(x_{12}, x_{13}, x_{23}, x_{24}, x_{45}, x_{34}, x_{35})'$. Zde x_{ij} odpovídá hraně (i, j). Analogicky zaveďme vektor kapacit $\boldsymbol{k} = (8, 4, 6, 1, 6, 2.5, 2)'$. Lineární program (5.1) pak získá tvar vhodný pro volání solveru `linprog` v syntaxi (3.5):

$$\max \overbrace{(0\,0\,0\,0\,1\,0\,1)}^{\boldsymbol{c}'} \begin{pmatrix} x_{12}\\ x_{13}\\ x_{23}\\ x_{24}\\ x_{45}\\ x_{34}\\ x_{35} \end{pmatrix} \text{ s.t. } \underbrace{\overbrace{\begin{pmatrix} \boldsymbol{I}_{7\times 7}\\ -\boldsymbol{I}_{7\times 7} \end{pmatrix}}^{\boldsymbol{A}} \begin{pmatrix} x_{12}\\ x_{13}\\ x_{23}\\ x_{24}\\ x_{45}\\ x_{34}\\ x_{35} \end{pmatrix} \le \overbrace{\begin{pmatrix} \boldsymbol{k}_{7\times 1}\\ \boldsymbol{0}_{7\times 1} \end{pmatrix}}^{\boldsymbol{b}}}_{(\star)}, \underbrace{\overbrace{\begin{pmatrix} -1 & 0 & 1 & 1 & 0 & 0 & 0\\ 0 & -1 & -1 & 0 & 0 & 1 & 1\\ 0 & 0 & 0 & -1 & 1 & -1 & 0 \end{pmatrix}}^{\boldsymbol{U}} \begin{pmatrix} x_{12}\\ x_{13}\\ x_{23}\\ x_{24}\\ x_{45}\\ x_{34}\\ x_{35} \end{pmatrix} = \overbrace{\begin{pmatrix} 0\\ 0\\ 0 \end{pmatrix}}^{\boldsymbol{v}}}_{(\dagger)} \tag{5.2}$$

Opět, podmínky $(\star)$ jsou kapacitní omezení (i) a podmínky $(\dagger)$ jsou tokové podmínky (ii).

Matice a vektory $\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{v}$ sestavíme snadno. Nechť $m = |E|$ značí počet hran. Matice $\boldsymbol{U}$ vypadá velmi podobně jako incidenční matice grafu G. Připomeňme, že *incidenční matice* $\boldsymbol{Z}$ je matice rozměru $n \times m$, řádky jsou indexovány vrcholy $v = 1, \dots, n$ a sloupce jsou indexovány hranami $e \in E$ a platí

$$Z_{ve} = \begin{cases} 1, & \text{vystupuje-li hrana } e \text{ z vrcholu } v,\\ -1, & \text{vstupuje-li hrana } e \text{ do vrcholu } v,\\ 0 & \text{jinak.} \end{cases} \tag{5.3}$$

V našem příkladu jest

$$\boldsymbol{Z} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0\\ -1 & 0 & 1 & 1 & 0 & 0 & 0\\ 0 & -1 & -1 & 0 & 0 & 1 & 1\\ 0 & 0 & 0 & -1 & 1 & -1 & 0\\ 0 & 0 & 0 & 0 & -1 & 0 & -1 \end{pmatrix}.$$

Matice $\boldsymbol{U}$ vznikne z matice $\boldsymbol{Z}$ smazáním prvního a posledního řádku; skutečně, tokové podmínky (ii) platí pro všechny vrcholy *kromě* zdroje (vrchol 1) a cíle (poslední vrchol v pořadí). A není náhodou, že vektor $\boldsymbol{c}'$ je (až na znaménko) roven poslednímu řádku matice $\boldsymbol{Z}$; skutečně, poslední řádek matice $\boldsymbol{Z}$ je charakteristický vektor hran vstupujících do cíle. Sestrojíme-li tedy incidenční matici $\boldsymbol{Z}$, snadno z ní získáme $\boldsymbol{U}$ a $\boldsymbol{c}$.

Je třeba zvolit vhodnou reprezentaci dat: jedna z přirozených reprezentací je uspořádat hrany do matice rozměru $m \times 2$, kde řádky odpovídají hranám a v řádku je uvedeno číslo počátečního a koncového vrcholu. Ve stejném pořadí zapišme kapacity ve vektoru $\boldsymbol{k}$.

Flow.m: Zadání dat.

```
3  n = 5; % pocet vrcholu
4  e = [1, 2; 1, 3; 2, 3; 2, 4; 4, 5; 3, 4; 3, 5]; % hrany
5  k = [   8;    4;    6;    1;    6;  2.5;    2]; % kapacity
6  m = length(k); % delka vektoru kapacit = pocet hran
```

Incidenční matici $\boldsymbol{Z}$ vytvoříme ve dvou krocích: nejprve si připravíme nulovou matici $\mathbf{0}_{n\times m}$ a poté `for`-cyklem přes hrany projdeme postupně sloupce $\boldsymbol{Z}$ a vyplníme do nich ± 1 podle předpisu (5.3).

Flow.m: Incidenční matice

```
8  Z = zeros(n, m);
9  for i = 1:m
10 Z(e(i, 1), i) = 1;
11 Z(e(i, 2), i) = -1;
12 end
```

Nyní již můžeme snadno definovat $\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{U}, \boldsymbol{v}$ dle (5.2) a volat solver `linprog` v syntaxi (3.5).

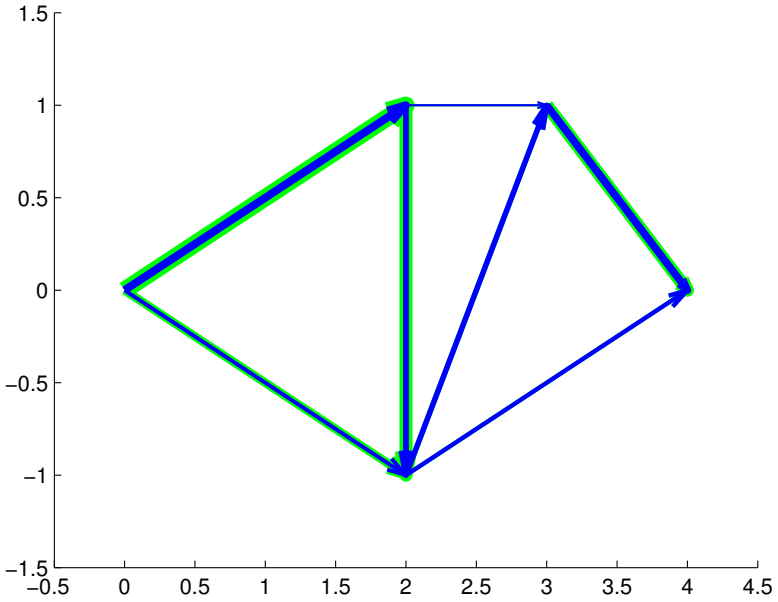
Flow.m: Řešení

```
14 A = [eye(m); -eye(m)];
15 b = [k; zeros(m, 1)];
16 c = -(Z(n, :))'; % posledni radek incidencni matice
17 U = Z(2:n-1, :); % incidencni matice bez prvniho ...
18   % a posledniho radku
19 v = zeros(n-2, 1);
20 x = linprog(-c, A, b, U, v) % piseme -c, uloha je totiz
21   % maximalizacni
```

Optimální $\boldsymbol{x}^*$ je nalezený maximální tok. (Upozorňujeme, že obecně nemusí být jednoznačný.)

5.2 Kresba obrázku

MATLAB disponuje knihovnou pro práci s grafy a jejich pokročilou vizualizaci. Její popis přesahuje rámec tohoto textu (nicméně je užitečné se podívat např. na `doc graph` pro základní informace o neorientovaných grafech a `doc digraph` pro základní informace o orientovaných grafech). Naším cílem je ukázat jednoduchý skript bez použití grafových nástrojů, kterým si lze síť a nalezený maximální tok nakreslit jen s pomocí vizualizace hran šipkami. Šipku bychom si mohli nakreslit sami pomocí funkce `plot` (šipka je koneckonců jen trojice úseček); nicméně elegantnější bude použít funkci `quiver`.



Obrázek 5.2: Hrany sítě.

Funkce `quiver`

Funkce

```
quiver(x, y, u, v, 0),
```

nakreslí šipku z bodu (x, y) do bodu $(x + u, y + v)$. Pátý argument bude u nás vždy nula; funkce `quiver` totiž ještě pracuje s „natahováním“ šipek (což se v některých aplikacích hodí), ale zde tuto funkci vypínáme. Funkce `quiver` používá stejné formátovací konvence jako `plot`, pročež můžeme analogicky nastavit barvu či sílu šipky. Například

```
quiver(1, 1, 2, 3, 0, 'r', 'LineWidth', 5)
```

nakreslí silnou červenou šipku z bodu $(1, 1)$ do bodu $(3, 4)$; síla šipky je 5 bodů.

Naším cílem bude nakreslit následující obrázek:

Je zde nakreslená síť (5.1). Zelená barva odpovídá kapacitám hran; síla zelené šipky je úměrná kapacitě. Modrými šipkami kreslíme nalezený maximální tok. Síla modré šipky odpovídá velikosti optimálního toku hranou. Například hrana $(1, 2)$ má kapacitu 8, což kreslíme zelenou šipkou síly 8 bodů; optimální tok touto hranou je jen 5, a tak silnou zelenou šipku částečně (ale ne zcela) překryjeme modrou šipkou o síle 5 bodů. Odtud je patrné, že kapacita této hrany není vyčerpána (zelená šipka je silnější než modrá). Jiným příkladem je hrana $(2, 4)$; ta má kapacitu 1. Zelená šipka síly 1 ovšem není viditelná, neboť je plně překryta modrou šipkou síly 1. To znamená, že kapacita této hrany je při optimálním toku zcela využita. Totéž platí pro hrany $(3, 4)$ a $(3, 5)$.

Je třeba říci, na jakých souřadnicích (ξ_i, y_i) v rovině mají být umístěny vrcholy $i = 1, \ldots, n\ (= 5)$. (Horizontální souřadnici raději značíme ξ, neboť symbol x už jsme použili pro optimální tok.) Tyto souřadnice pak použijeme jako počáteční a koncové body šipek. Aby byl náš obrázek opticky podobný

obrázku (5.1), zvolme například

$$(\xi_1, y_1) = (0,0),\ \ (\xi_2, y_2) = (2,1),\ \ (\xi_3, y_3) = (2,-1),\ \ (\xi_4, y_4) = (3,1),\ \ (\xi_5, y_5) = (4,0).$$

Flow.m: Souřadnice vrcholů.

```
23 xi = [0; 2; 2; 3; 4];
24 y  = [0; 1;-1; 1; 0];
```

Nyní se hodí vytvořit pomocnou funkci `PlotNet`, která pomocí `for`-cyklu postupně vykreslí hrany jako šipky, a to v síle dané vektorem `weights` a barvou `color`. Tuto funkci pak zavoláme dvakrát; nejprve pro vykreslení zelených šipek, kde `weights` jsou kapacity, a podruhé pro vykreslení modrých šipek, kde `weights` jsou velikosti optimálního toku hranami.

Flow.m: Funkce pro kresleni hran

```
26 function PlotNet(weights, color)
27   for i = 1:m % i probiha mnozinou hran
28   quiver(xi(e(i, 1)), y(e(i, 1)), ...
29          xi(e(i, 2)) - xi(e(i, 1)), ...
30          y(e(i, 2)) - y(e(i, 1)), ...
31          0, color, 'Linewidth', weights(i));
32   end
33 end
```

Všimněme si, že `xi(e(i,1))` a `y(e(i,1))` jsou rovinné souřadnice počátečního vrcholu i-té hrany a `xi(e(i,2))` a `y(e(i,2))` jsou rovinné souřadnice koncového vrcholu i-té hrany. Kreslíme tedy šipku z bodu `xi(e(i,1))` a `y(e(i,1))` a ve funkci `quiver` klademe $u =$ `xi(e(i,2)) − xi(e(i,1))` a $v =$ `y(e(i,2)) − y(e(i,1))`.

Nyní stačí volat `PlotNet(k,'g')` pro vykreslení zelených šipek a `PlotNet(x,'b')` pro vykreslení modrých šipek (zde `x` je nalezený optimální tok pomocí solveru `linprog`).

Flow.m: Souřadnice vrcholů.

```
35 figure;
36 hold on;
37 PlotNet(k, 'g'); % kapacity k zelene
38 PlotNet(x, 'b'); % optimalni tok modre
```

Poznámka

S modelem si lze „hrát“ obvyklým způsobem. Kdybychom například připustili, že tok hranou může být oběma směry (a model má vybrat ten směr, který povede k maximalizaci přítoku do cíle), stačí v (5.1) nahradit podmínku $0 \le x_e \le k_e$ podmínkou $-k_e \le x_e \le k_e$. Pak v (5.2) bude $\boldsymbol{b} = \binom{\boldsymbol{k}}{-\boldsymbol{k}}$, a tedy stačí ve skriptu psát `b = [k; -k]`. Vše ostatní zůstává beze změny (pochopitelně by bylo třeba upravit vizualizační skript, aby kreslil modré šipky správným směrem podle znamének ve vektoru optimálního toku $\boldsymbol{x}^*$; to ponecháváme jako cvičení).

Analogicky lze uvážit tzv. multikomoditní toky, toky s dvojitě oceněnými hranami — kapacitou a přepravními náklady —, toky s více zdroji a/nebo více cíli, toky s úbytky ve vnitřních vrcholech atd. Je poučné coby cvičení tyto lineární programy zformulovat a napsat je jako skripty.

Doporučená literatura

- Bertsekas, D. P. 1998. *Network Optimization: Continuous and Discrete Methods*. Athena Scientific. ISBN 978-1-886529-02-1. http://www.athenasc.com/netbook.html
- Hillier, F. S., Lieberman, G. J. 2018. *Introduction to Operations Research*. McGraw-Hill Education. ISBN 978-0-07-352345-3. http://highered.mheducation.com/sites/0073523453/information{_}center{_}view0/index.html
- Jablonský, J. 2007. *Operační výzkum: Kvantitativní modely pro ekonomické rozhodování*. Professional Publishing. ISBN 978-80-86946-44-3. https://www.researchgate.net/publication/40354299
- Kwon, R. H. 2013. *Introduction to Linear Optimization and Extensions with MATLAB*. CRC Press. ISBN 978-1-4398-6264-3. https://doi.org/10.1201/b13966
- Vanderbei, R. J. 2020. *Linear Programming: Foundations and Extensions*. Springer. ISBN 978-3-030-39414-1. https://doi.org/10.1007/978-3-030-39415-8

Kapitola 6

Metoda kritické cesty

Výpočtem délky projektu, který se skládá z několika dílčích úloh, se zabývá metoda CPM. Kromě této deterministické metody si ukážeme i simulace pro případ, kdy délky dílčích úloh jsou náhodné veličiny.

Klíčová slova: *CPM, PERT, simulace.*

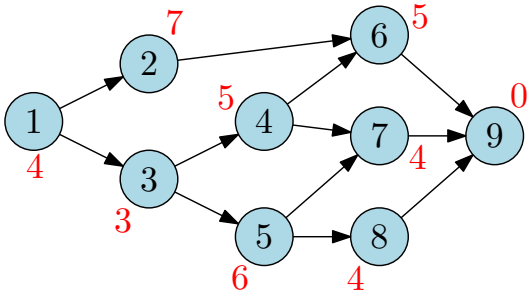
6.1 Výpočet délky projektu pro dané doby dílčích úloh

Metoda CPM

Metoda CPM (metoda kritické cesty, *Critical Path Method*) je jednoduchý postup v řízení projektů. Projekt sestává z dílčích úkolů, indexovaných $1, \ldots, n$, a pro úkol i je zadána doba trvání d_i. Dále je zadána *precedence*: pro některé dvojice úkolů (i, j) je řečeno, že úkol j nemůže začít před dokončením úkolu i; jinak mohou úkoly běžet paralelně. Je přirozené tuto situaci reprezentovat pomocí orientovaného grafu $G = (V, E)$: vrcholy $V = \{1, \ldots, n\}$ nechť představují úkoly a nechť dvojice (i, j) tvoří hranu ($(i, j) \in E$), právě když úkol j nemůže začít před dokončením úkolu i. Cílem je zjistit dobu trvání celého projektu. Předpokládejme, že graf G je acyklický (jinak by projekt nešlo dokončit nikdy); nutně tudíž má vrchol nulového vstupního stupně a vrchol nulového výstupního stupně. Budeme bez újmy na obecnosti dále předpokládat, že vrchol (úkol) s nulovým výstupním stupněm je jediný, že je to vrchol n a že platí $d_n = 0$; snadno se nahlédne, že toho lze dosáhnout vždy. Úkol s pořadovým číslem n pak vlastně představuje pouhý formální úkol, totiž ukončení projektu, který trvá nulovou dobu.

Nechť x_i představuje časový okamžik, kdy nejdříve může úkol x_i začít. Dobu trvání projektu lze zjistit pomocí lineárního programu

$$\min_{x_1,\ldots,x_n} x_n \quad \text{s.t.} \quad \underbrace{x_j \geq x_i + d_i \ (\forall (i,j) \in E)}_{(\star)}, \quad x_i \geq 0 \ (\forall i \in V). \tag{6.1}$$



Obrázek 6.1: Reprezentace projektu pomocí vrcholů a hran grafu.

Poznámka

Jde to spočítat mnohem jednodušeji než pomocí LP — po krátké úvaze se snadno přijde na přímočarý algoritmus, ovšem nám zde jde o cvičení na LP v MATLABu, a je to takto navíc zřejmě nejjednodušší k programování.

Podmínky $(\star)$ říkají: je-li (i,j) hrana, pak začátek x_j úkolu j může nastat nejdříve poté, co je dokončen úkol i (to je okamžik $x_i + d_i$). Minimalizujeme x_n — to je okamžik, kdy projekt vyvrcholí posledním úkolem, a tedy také doba trvání celého projektu (připomeňme konvenci $d_n = 0$).

Přepišme (6.1) do tvaru

$$\min_{x_1,\ldots,x_n} x_n \;\text{ s.t. }\; x_i - x_j \le -d_i \;(\forall (i,j) \in E), \quad -x_i \le 0 \;(\forall i \in V);$$

odtud je již přímočarý krok k tvaru

$$\min_{\boldsymbol{x}=(x_1,\ldots,x_n)'} (\mathbf{0}_{1\times(n-1)}\ 1)'\boldsymbol{x} \;\text{ s.t. }\; \boldsymbol{Z}'\boldsymbol{x} \le -\boldsymbol{\delta},\ -\boldsymbol{I}_{n\times n}\boldsymbol{x} \le \mathbf{0}_{n\times 1},$$

kde $\boldsymbol{Z}$ je incidenční matice grafu G daná předpisem (5.3) z kapitoly 5, m označuje počet hran a $\boldsymbol{\delta}$ je vektor dob trvání úkolů definovaný předpisem

$$\delta_e = d_i, \quad \text{jestliže hrana } e \text{ začíná ve vrcholu } i \;\; (e \in E). \tag{6.2}$$

Složky vektoru $\boldsymbol{\delta}$ jsou indexovány hranami ve stejném pořadí jako sloupce incidenční matice $\boldsymbol{Z}$. Pišme konečně

$$\min_{\boldsymbol{x}} \underbrace{(\mathbf{0}_{1\times(n-1)}1)'}_{\boldsymbol{c}'}\boldsymbol{x} \;\text{ s.t. }\; \underbrace{\begin{pmatrix}\boldsymbol{Z}' \\ -\boldsymbol{I}_{n\times n}\end{pmatrix}}_{\boldsymbol{A}}\boldsymbol{x} \le \underbrace{\begin{pmatrix}-\boldsymbol{\delta} \\ \mathbf{0}_{n\times 1}\end{pmatrix}}_{\boldsymbol{b}}; \tag{6.3}$$

toto je již vhodný tvar pro solver `linprog`.

Pro účely skriptu zvolíme stejnou reprezentaci grafu G jako v kapitole 5, totiž výčet hran v matici rozměru $m \times 2$, kde v k-tém řádku je zapsán počáteční a koncový vrchol k-té hrany. Pro příklad uvažme graf (projekt) z následujícího obrázku; černě jsou uvedeny indexy vrcholů (úkolů) i a červeně jejich doby trvání d_i.

Nejprve reprezetujme data stejně jako v kapitole 5.

CPM.m: Zadání dat.

```
3  d = [4; 7; 3; 5; 6; 5; 4; 4; 0]; % doby trvani
4  e = [1, 2; 1, 3; 2, 6; 3, 4; 3, 5; 4, 6; 4, 7; 5, 7; ...
5       5, 8; 6, 9; 7, 9; 8, 9]; % vycet hran
6  n = length(d); % pocet vrcholu = pocet dob trvani
7  m = length(e); % pocet hran = pocet radku matice e
```

Sestrojme incidenční matici $\boldsymbol{Z}$ (opět, stejně jako v kapitole 5).

CPM.m: Incidenční matice

```
9  Z = zeros(n, m);
10 for i = 1:m
11   Z(e(i, 1), i) =  1;
12   Z(e(i, 2), i) = -1;
13 end
```

Nyní jsme připraveni vyřešit LP (6.3).

CPM.m: Řešení

```
15 c = [zeros(n - 1, 1); 1];
16 A = [Z'; -eye(n)];
17 delta = d(e(:, 1));
18 b = [-delta; zeros(n)];
19 x = linprog(c, A, b);
20 disp(x(n)); % x_n je posledni ukol, a tedy doba ...
21             % trvani celeho projektu
```

Skript ohlásí, že doba trvání projektu je 17; prostým pohledem na graf G v obrázku 6.1 se snadno ověří, že je tomu skutečně tak.

Je vhodné si podrobně rozmyslet, proč instrukce `delta = d(e(:,1))` skutečně vytvoří vektor $\boldsymbol{\delta}$ splňující (6.2) — připomeňme, že `e(:,1)` je vektor indexů počátečních vrcholů jednotlivých hran.

6.2 Výpočet délky projektu pro náhodné doby dílčích úloh

Řekněme, že doby trvání úkolů d_i nejsou přesně známé; modelujme je jako náhodné veličiny, jejichž rozdělení (jak se zde předpokládá) známe. Pak i doba trvání projektu je náhodná veličina; pochopitelně nás zajímá její rozdělení a jeho charakteristiky (střední hodnota, medián, rozptyl, případně vyšší momenty, kvantily atd.). V padesátých létech byla vyvniuta tzv. metoda PERT (*Program Evaluation and Review Technique*), která se snaží rozdělení doby trvání projektu aproximovat pomocí normálního rozdělení, ovšem za velmi restriktivních předpokladů a s vážným rizikem velké chyby aproximace. (Snadno se nahlédne, že obecně lze těžko očekávat, že by doba trvání projektu měla mít například symetrické rozdělení; už z tohoto základního náhledu je patrné, že aproximace normálním — a tedy symetrickým — rozdělením může být silně zavádějící, může třeba podhodnocovat riziko výrazného prodloužení projektu.) Metoda PERT je dosti hrubá heuristika, jež v podstatě nemá význam v okamžiku, kdy je snadné rozdělení doby trvání projektu nasimulovat. Simulaci si nyní ukažme.

Vytvořme si pomocnou funkci `time = SolveCPM(d,e)`, kde shrneme, co jsme dosud vytvořili — funkce dostane na vstup vektor d dob trvání úkolů a seznam e hran grafu G a vrátím nám `time`, dobu trvání projektu. Tato funkce poslouží coby CPM-solver.

SimulCPM.m: Solver CPM

```
3  function time = SolveCPM(d, e)
4    % time = doba trvani projektu zadaneho ...
5    % hranami e a dobami ukolu d
6    n = length(d); % pocet vrcholu
7    m = length(e); % pocet hran
8    Z = zeros(n, m); % incidencni matice Z
9    for i = 1:m
10     Z(e(i, 1), i) = 1;
11     Z(e(i, 2), i) = -1;
12   end
13   c = [zeros(n - 1, 1); 1]; % priprava na volani linprog
14   A = [Z'; -eye(n)];
15   b = [-d(e(:, 1)); zeros(n, 1)];
16   x = linprog(c, A, b);
17   time = x(n); % x n je posledni ukol, ...
18     % a tedy doba trvani celeho projektu
19 end
```

Nyní budeme simulovat doby trvání úkolů z jejich distribucí a opakovaně volat `SolveCPM`. Simulaci zopakujeme (řekněme) 10^5-krát; doby trvání zaznamenáme do pomocného vektoru $\boldsymbol{h}$, z nějž vykreslíme histogram (to je simulací získaný odhad skutečné distribuce doby trvání projektu) a pro ilustraci napočteme některé charakteristiky. Protože simulace chvíli trvá, je vhodné skript animovat — postupně vykreslovat empirickou distribuci (histogram) a nechat uživatele sledovat, jak proces konverguje.

V našem příkladu řekněme, že doby trvání jsou nezávislé, rovnoměrně rozdělené náhodné veličiny na intervalu $[d_i - 2, d_i + 3]$, kde $i = 1, \ldots, 8$ jsou úkoly z obrázku 6.1 a d_i jsou v obrázku 6.1 červeně uvedné hodnoty. (Připomínáme, že podle přijaté konvence vždy jest $d_9 = 0$.)

Poznámka

Rovnoměrné rozdělení jsme zvolili jen pro příklad; při simulaci lze použít kterékoliv rozdělení, třeba β-rozdělení (jak je zvykem v PERTu) či třeba empirické rozdělení dob trvání úkolů získané z historických dat. Stejně tak lze užívat i závislé náhodné veličiny; to by se hodilo např. tehdy, provádějí-li různé úkoly titíž pracovníci, anebo více úkolů je ovlivněno společným faktorem, třeba nepříznivým počasím při stavebních pracích.

SimulCPM.m: Zadání vstupních dat

```
21 d = [4; 7; 3; 5; 6; 5; 4; 4; 0]; % doby trvani ukolu
22 e = [1, 2; 1, 3; 2, 6; 3 ,4; 3, 5; 4, 6; 4, 7; 5, 7; ...
23   5, 8; 6, 9; 7, 9; 8, 9];
```

SimulCPM.m: Vlastní simulace

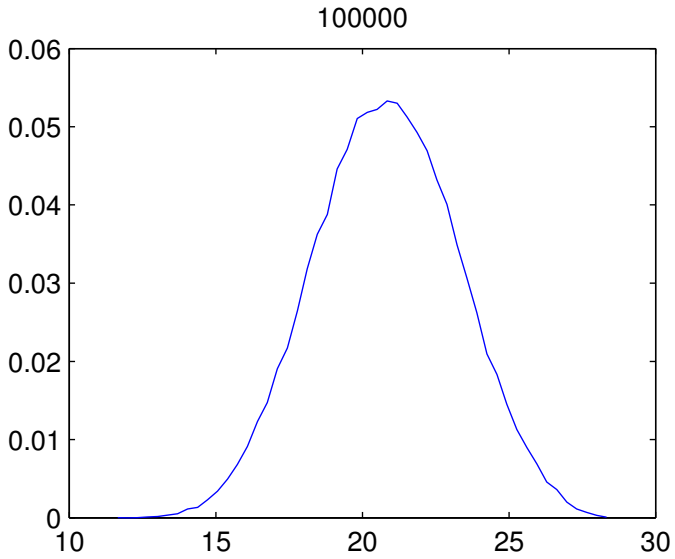
```
figure;
n = length(d); % pocet vrcholu (ukolu)
h = []; % zde budeme uchovavat simulovane doby trvani
for j = 1:10^5 % pocet simulaci
  d1 = d + [random('unif', -2, 3, [n - 1, 1]); 0];
  % d1 jsou puvodni doby d plus nahodna chyba ...
    % z Unif(-2,3) [krome posledniho ...
    % ukolu, ta je vody 0]
  time = SolveCPM(d1, e); % spocti dobu trvani ...
    % projektu pri dobach ukolu d1
  h = [h; time]; % do vektoru h uloz hodnotu time
  [freq, bins] = hist(h, 50); % z dosud spoctenych ...
    % hodnot h spocti histogram s 50 prihradkami ...
    % freq jsou absolutni cetnosti a bins jsou ...
    % stredy prihradek
  plot(bins, freq ./ length(h)); % vykresli ...
    % (relativni) cetnosti proti stredum prihradek
  title(j); % v hlavicce obrazku vypis cislo iterace ...
    % (at vime, jak jsme daleko)
  pause(0.0001); % formalni pauza, refresh
end
```

SimulCPM.m: Pár charakteristik na závěr

```
disp(['Prumer = ',        num2str(mean(h))]);
disp(['Max = ',           num2str(max(h))]);
disp(['Min = ',           num2str(min(h))]);
disp(['Smerodatna odchylka = ', num2str(var(h).^0.5)]);
disp(['Median = ',        num2str(median(h))]);
disp(['90% kvantil = ',   num2str(quantile(h, 0.9))]);
disp(['95% kvantil = ',   num2str(quantile(h, 0.95))]);
disp(['99% kvantil = ',   num2str(quantile(h, 0.99))]);
```

Výsledkem skriptu je simulované rozdělení doby trvání projektu a několik jeho základních charakteristik:

```
Prumer = 20.8116
Max = 28.5031
Min = 12.0171
Smerodatna odchylka = 2.4402
Median = 20.8179
90% kvantil = 23.9856
95% kvantil = 24.8517
99% kvantil = 26.2794
```



Obrázek 6.2: Simulovaná distribuce doby trvání projektu.

6.3 Analýza simulované distribuce

Ačkoliv simulovaná distribuce se může na první pohled jevit podobná normálnímu rozdělení (jak ji aproximuje metoda PERT), není tomu tak; již na druhý pohled je patrné sešikmení (jinými slovy: je zde větší tendence k prodlužování doby trvání projektu než k jeho rychlému dokončení). Můžeme zkusit otestovat shodu s normálním rozdělením např. Jarque-Berovým testem.

Poznámka

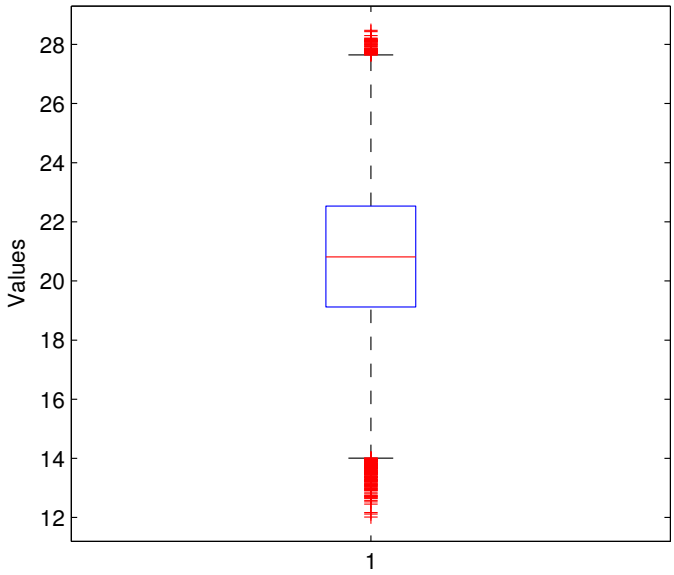
Připomeňme, že kouzlo J.-B. testu na shodu distribucí spočívá v následujícím. Další běžné testy — např. χ^2-test či Kolmogorov-Smirnovův test (v MATLABu jako `kstest`) — dokáží testovat shodu empirické (v našem případě: simulované) distribuce s $N(\mu, \sigma^2)$ *při daných hodnotách* μ *a* σ^2. Ty ale nikdo nezná a jejich odhady pomocí výběrového průměru a rozptylu jsou zatíženy další chybou. Naproti tomu J.-B. test má za nulovou hypotézu, že pozorovaná empirická (simulovaná) distribuce má špičatost rovnu 0 a šikmost rovnu 0. To jsou vlastnosti právě normálního rozdělení. J.-B. test tudíž nevyžaduje znalost μ ani σ^2, protože pro jejich libovolné hodnoty je vždy šikmost i špičatost normálního rozdělení rovna 0.

`SimulCPM.m`: Test normality

```
56  [rozhodnuti, pvalue] = jbtest(h)
```

Výsledná hodnota `rozhodnuti` je 1, jestliže test na 5% hladině zamítá hypotézu o normalitě (a je 0, nezamítá-li), a `pvalue` je dosažená hladina testu. V našem případě je `pvalue` nerozlišitelně blízko nule; J.-B. test tedy hypotézu o shodě simulací získané distribuce s normálním rozdělením dokonce zamítá „zcela radikálně“.

Konečně je možné užít i další běžné nástroje pro vizualizaci dat, například nakreslit boxplot.



Obrázek 6.3: Boxplot.

SimulCPM.m: Vykreslení boxplotu

```
58 figure;
59 boxplot(h)
```

Doporučená literatura

- Hillier, F. S., Lieberman, G. J. 2018. *Introduction to Operations Research*. McGraw-Hill Education. ISBN 978-0-07-352345-3. http://highered.mheducation.com/sites/0073523453/information{_}center{_}view0/index.html
- Jablonský, J. 2007. *Operační výzkum: Kvantitativní modely pro ekonomické rozhodování*. Professional Publishing. ISBN 978-80-86946-44-3. https://www.researchgate.net/publication/40354299
- Vanhoucke, M. 2013. *Project Management with Dynamic Scheduling: Baseline Scheduling, Risk Analysis and Project Control*. Springer. ISBN 978-3-642-40437-5. https://doi.org/10.1007/978-3-642-40438-2

Kapitola 7

Von Neumannův růstový model

V této kapitole si představíme Von Neumannův růstový model, který následně budeme řešit pomocí lineárního programování.

Klíčová slova: *Von Neumannův růstový model, binární vyhledávání, lineární programování.*

7.1 Základní problém

Uvažme von Neumannův růstový model s nezápornou maticí vstupů $\boldsymbol{A} \in \mathbb{R}^{n\times m}$ (input matrix) a nezápornou maticí výstupů $\boldsymbol{B} \in \mathbb{R}^{n\times m}$ (output matrix). Víme, že von Neumannovo optimální tempo růstu γ^* a von Neumannovy optimální intenzity $\boldsymbol{x}^*$ producentů (aktivit) se získají jako řešení optimalizačního problému

$$\begin{aligned} \max_{\substack{\gamma\in\mathbb{R}\\ x\in\mathbb{R}^n}} \quad & \gamma \\ \text{s.t.} \quad & \boldsymbol{B}'x \geq \gamma \boldsymbol{A}'\boldsymbol{x}, \\ & \boldsymbol{x} \geq \boldsymbol{0}, \\ & \boldsymbol{x} \neq \boldsymbol{0}. \end{aligned} \tag{7.1}$$

Připomeňme, že (7.1) se také nazývá *primární von Neumannův problém* či *von Neumannův problém technologické expanze*.

7.2 Formulace jako úloha lineárního programování

Zřejmě je (7.1) nelineární optimalizační problém: jednak se zde vyskytuje součin proměnných $\gamma \cdot x_i$ a jednak zde máme podmínku $\boldsymbol{x} \neq \boldsymbol{0}$. Součiny proměnných a omezující podmínky typu „$\neq$“ jsou v LP zakázány (a obecně jsou takové problémy **NP**-těžké).

Naším cílem je sestavit funkci `[gamma, x] = vonNeumann(A,B)`, které uživatel na vstup zadá dvojici input-output matic $(\boldsymbol{A}, \boldsymbol{B})$ a funkce spočte optimální tempo růstu `gamma` a optimální vektor intenzit `x`. Musíme si ovšem poradit právě s nelinearitou optimalizačního problému (7.1). Lze na to jít

různě; jednou z možností je reformulovat (7.1) jako tzv. zobecněný lineárně-frakcionální program (*generalized linear-fractional programming problem*, GLFP); jde o typ nelineárních optimalizačních problémů, které jsou pomocí metod vnitřního bodu řešitelné zhruba stejně efektivně jako lineární programy. Touto cestou se nevydáme (vyžadovalo by to totiž příliš mnoho teorie) a ukážeme, jak problém (7.1) řešit s užitím LP.

Snadno se nahlédne, že systém nerovností $\boldsymbol{B}'\boldsymbol{x} \geq \gamma \boldsymbol{A}'\boldsymbol{x}$ je homogenní v $\boldsymbol{x}$ (to znamená: je-li $\boldsymbol{x}$ řešením, pak je také $\alpha\boldsymbol{x}$ řešením pro libovolné $\alpha > 0$). A díky podmínkám $\boldsymbol{x} \geq \boldsymbol{0}$, $\boldsymbol{x} \neq \boldsymbol{0}$ můžeme bez újmy na obecnosti zafixovat z nekonečně mnoha řešení jedno konkrétní, například to, jehož složky se nasčítají na 1. Pišme proto

$$\max_{\substack{\gamma\in\mathbb{R}\\ \boldsymbol{x}\in\mathbb{R}^n}} \gamma \;\text{ s.t. }\; \boldsymbol{B}'\boldsymbol{x} \geq \gamma \boldsymbol{A}'\boldsymbol{x},\;\; \boldsymbol{x} \geq \boldsymbol{0},\;\; \boldsymbol{e}'\boldsymbol{x} = 1, \tag{7.2}$$

kde $e = (1, \ldots, 1)'$.

Z teorie víme, že problém (7.1) má vždy optimum s $\gamma^* > 0$ (za velmi mírných a přirozených předpokladů na input-output matice $(\boldsymbol{A}, \boldsymbol{B})$). Pohleďme nyní na „lineární program“

$$(LP_\gamma) \quad \max_{\boldsymbol{x}\in\mathbb{R}^n} \boldsymbol{0}'\boldsymbol{x} \;\text{ s.t. }\; \underbrace{\boldsymbol{B}'\boldsymbol{x} \geq \gamma \boldsymbol{A}'\boldsymbol{x},\;\; \boldsymbol{x} \geq \boldsymbol{0},\;\; \boldsymbol{e}'\boldsymbol{x} = 1}_{(\star)},$$

kde $\gamma > 0$ je *parametr*, nikoliv proměnná modelu. Pak se skutečně jedná o lineární program! Jen účelová funkce je zde identická nula — ale nám nepůjde o její maximalizaci či minimalizaci, to by bylo absurdní. Půjde nám o následující: pomocí solveru `linprog` chceme jen rozhodnout, při dané hodnotě γ, o *přípustnosti či nepřípustnosti* systému lineárních omezení $(\star)$. Proto je účelová funkce irelevantní a můžeme ji zvolit jakkoliv, třeba jako identickou nulu. Idea pro následující postup je tato: pro velké hodnoty γ je systém $(\star)$ jistě nepřípustný, zatímco pro malé hodnoty γ je přípustný; optimální hodnotu γ^* (a jí odpovídající optimální intenzity $\boldsymbol{x}^*$) budeme hledat „někde mezi“. Učiníme tak zanedlouho metodou půlení intervalu.

7.3 Testování (ne)přípustnosti lineárního programu

Užijeme syntaxi

```
[x, optval, flag] = linprog(c,A,b,U,v);
```
(7.3)

zde solver `linprog` řeší lineární program tvaru $\min_{\boldsymbol{x}}\{\boldsymbol{c}'\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{U}\boldsymbol{x} = \boldsymbol{v}\}$ a vrací optimální řešení `x`, optimální hodnotu účelové funkce `optval` a — pro nás nyní důležitou — hodnotu `flag`, která informuje o způsobu ukončení výpočtu solveru. Hodnota `flag` $= 1$ znamená, že bylo nalezeno optimum, a `flag` $= -2$ znamená, že problém je nepřípustný. (Pro úplnost dodejme, že `flag` $= -3$ znamená neomezenost, která ovšem v našem konkrétním případě nemůže nastat; další hodnoty `flag` značí vesměs numerické problémy při řešení rozsáhlých úloh LP či překročení povoleného počtu iterací. Detaily podá `help linprog`.)

Chceme-li otestovat přípustnost (LP_γ) (při zadané hodnotě parametru γ), pišme (LP_γ) v ekvivalentním tvaru

$$\max_{\boldsymbol{x}\in\mathbb{R}^n} \boldsymbol{0}'\boldsymbol{x} \;\text{ s.t. }\; (\gamma\boldsymbol{A}' - \boldsymbol{B}')\boldsymbol{x} \leq \boldsymbol{0},\;\; -\boldsymbol{x} \leq \boldsymbol{0},\;\; \boldsymbol{e}'\boldsymbol{x} = 1;$$

pak už okamžitě vidíme

$$\max_{\boldsymbol{x}\in\mathbb{R}^n} \underbrace{\boldsymbol{0}'}_{\boldsymbol{c}'}\boldsymbol{x} \;\text{ s.t. }\; \underbrace{\begin{pmatrix}\gamma\boldsymbol{A}' - \boldsymbol{B}'\\ -\boldsymbol{I}\end{pmatrix}}_{\boldsymbol{D}}\boldsymbol{x} \leq \underbrace{\begin{pmatrix}0\\0\end{pmatrix}}_{\boldsymbol{b}},\;\; \underbrace{\boldsymbol{e}'}_{\boldsymbol{U}}\boldsymbol{x} = \underbrace{1}_{\boldsymbol{v}} \tag{7.4}$$

a voláme `[x, optval, flag] = linprog(c,D,b,U,v)`. Je-li `flag` $= 1$, pak je problém přípustný; jiank je nepřípustný (pomiňme zde možné další hodnoty `flag` z titulu numerických problémů). Pro stručnost volejme solver `linprog` v kompaktní formě

```
  [x, optval, flag] = ...
linprog(zeros(n,1), [gamma.*A' - B'; -eye(n)], zeros(m+n,1), ones(1,n), 1).
```

c D b U v

7.4 Binární vyhledávání (půlení intervalu)

Algoritmus binárního vyhledávání se skládá z následujících kroků:

1. Zvolme velkou hodnotu $\overline{\gamma}$, a to tak, aby $(LP_{\overline{\gamma}})$ byl jistě nepřípustný; pro naše účely jistě postačí zvolit například $\overline{\gamma} = 10^4$. Zvolme také malou hodnotu $\underline{\gamma}$ (například $\underline{\gamma} = 10^{-8}$) tak, aby $(LP_{\underline{\gamma}})$ byl jistě přípustný.

2. Pohleďme doprostřed intervalu $[\underline{\gamma}, \overline{\gamma}]$: položme

$$\gamma' = \tfrac{1}{2}(\underline{\gamma} + \overline{\gamma}). \tag{7.5}$$

3. Otestujme přípustnost $(LP_{\gamma'})$.

4. Nyní:

   a) je-li $(LP_{\gamma'})$ přípustný, musí optimální γ^* ležet v intervalu $[\gamma', \overline{\gamma}]$. Položíme proto $\underline{\gamma} := \gamma'$ a pokačujeme Krokem 2 s novým, nyní již na polovinu zúženým intervalem $[\underline{\gamma}, \overline{\gamma}]$;

   b) je-li ovšem $(LP_{\gamma'})$ nepřípustný, musí optimální γ^* ležet v intervalu $[\underline{\gamma}, \gamma']$. Položíme proto $\overline{\gamma} := \gamma'$ a pokračujeme Krokem 2 s novým, nyní již na polovinu zúženým intervalem $[\underline{\gamma}, \overline{\gamma}]$.

Postup opakujeme, dokud se nedosáhne zanedbatelně malé chyby, řekněme dokud není $\overline{\gamma} - \underline{\gamma} < 10^{-8}$. Takto jsme (sice jen přibližně, ale s libovolně malou chybou) nalezli optimální tempo růstu γ^* (a samozřejmě také jemu odpovídající vektor optimálních intenzit $\boldsymbol{x}^*$). Všimněme si, že interval $[\underline{\gamma}, \overline{\gamma}]$ se zužuje geometrickou řadou; požadované přesnosti tudíž dosáhneme velice rychle (přibližně za $\log_2 \frac{10^4}{10^{-8}} \approx 40$ iterací). Celý skript může vypadat například následujícím způsobem.

VonNeumann.m: Binární vyhledávání.

```
function [gamma, x] = solveVonNeumann(A, B)
  gammaUp = 10^4;   % pocatecni hodnota
  gammaLow = 10^-8; % pocatecni hodnota
  while(gammaUp - gammaLow >= 10^-8)
    % hlavni cyklus binarniho vyhledavani
    % dokud je interval prilis siroky
    gammaPrime = mean([gammaUp, gammaLow]);
      % prostredek intervalu
    [x0, optval, flag] = linprog(zeros(n, 1), ...
      [gammaPrime .* A' - B'; -eye(n)], ...
      zeros(m + n, 1), ones(1, n), 1);
    % test pripustnosti
    if flag == 1              % problem je pripustny
      gammaLow = gammaPrime; % posun dolni mez nahoru
      x = x0;                % pamatuj si pripustne ...
                               % intenzity
      gamma = gammaPrime;    % pamatuj si pripustne ...
                               % tempo rustu
    else                      % problem je nepripustny
      gammaUp = gammaPrime;  % posun horni mez dolu
    end
  end
  % vystupni hodnotou je posledni zapamatovane...
    % pripustne (gamma,x)
end
```

Poznámka

Půlením intervalu lze také analogicky řešit duální von Neumannův problém

$$\min_{\substack{\beta\in\mathbb{R}\\ \boldsymbol{y}\in\mathbb{R}^m}} \beta \text{ s.t. } \boldsymbol{B}\boldsymbol{y} \leq \beta\boldsymbol{A}\boldsymbol{y}, \ \boldsymbol{y} \geq \boldsymbol{0}, \ \boldsymbol{y} \neq \boldsymbol{0},$$

jehož optimum $(\beta^*, \boldsymbol{y}^*)$ dává stínové ceny $\boldsymbol{y}^*$ produktů a optimální profit-faktor (optimální úrokovou sazbu) β^*. Detaily ponecháme k vypracování jako cvičení.

Kapitola 8

Dopravní problém

Dopravní problém je klasická úloha, ve které jde o minimalizaci ceny přepravy zboží. Jedná se o úlohu lineárního programování.

Klíčová slova: *dopravní problém, lineární programování.*

8.1 Řešení pomocí lineárního programování

Dopravní problém

Dopravní problém je následující úloha: je dán seznam m dodavatelů jisté komodity a n odběratelů této komodity. Úkolem je přepravit komoditu od dodavatelů k odběratelům s tím, že je dán vektor $\boldsymbol{p} = (p_1, \ldots, p_n)'$ požadavků (odběratel i chce odebrat právě množství p_i) a vektor $\boldsymbol{k} = (k_1, \ldots, k_m)'$ kapacit dodavatelů (dodavatel j disponuje jen omezeným množstvím k_j). Dále je zadána nezáporná matice $\boldsymbol{C} = (c_{ij}) \in \mathbb{R}^{n \times m}$ jednotkových cen přeprav — c_{ij} je cena přepravy jedné tuny komodity od dodavatele j k odběrateli i. Cílem je zajist přepravu, která uspokojí požadavky odběratelů, nepřekročí kapacity dodavatelů a bude co nejlevnější. (Všimněme si, že úloha má řešení právě tehdy, když $\boldsymbol{e}'\boldsymbol{k} \geq \boldsymbol{e}'\boldsymbol{p}$; tuto podmínku je snadné ověřit.)

Problém se snadno zformuluje jako LP s proměnnými x_{ij} = objem přepravy mezi dodavatelem j a odběratelem i:

$$\min_{x_{ij}} \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} \quad \text{s.t.} \quad \underbrace{\sum_{j=1}^{m} x_{ij} = p_i \ (\forall i = 1, \ldots, n),}_{(\star)} \quad \underbrace{\sum_{i=1}^{n} x_{ij} \leq k_j \ (\forall j = 1, \ldots, m),}_{(\dagger)} \quad x_{ij} \geq 0 \, (\forall i, j). \tag{8.1}$$

Podmínky $(\star)$ uspokojí požadavky odběratelů a podmínky $(\dagger)$ zajistí, že se nepřekročí kapacity dodavatelů.

Naším cílem zde je napsat funkci `X = DopravniProblem(C, k, p)`, kde uživatel na vstup zadá data $\boldsymbol{C}, \boldsymbol{k}, \boldsymbol{p}$ a obdrží matici `X` $= (x^*_{ij})$ optimálních převozů. Zvolíme-li na chvíli jako příklad $m = 3$

dodavatele a $n = 4$ odběratele, můžeme (8.1) přepsat do podoby, kde „naskládáme“ proměnné a ceny do jednoho vektoru po sloupcích

$$\boldsymbol{x} = (x_{11}, x_{21}, x_{31}, x_{41},\ x_{12}, x_{22}, x_{32}, x_{42},\ x_{13}, x_{23}, x_{33}, x_{43})', \tag{8.2}$$
$$\boldsymbol{c} = (c_{11}, c_{21}, c_{31}, c_{41},\quad c_{12}, c_{22}, c_{32}, c_{42},\quad c_{13}, c_{23}, c_{33}, c_{43})' \tag{8.3}$$

a obdržíme (8.1) ve tvaru

$$\min_{\boldsymbol{x}} \boldsymbol{c}'\boldsymbol{x} \ \text{ s.t. } \underbrace{\begin{pmatrix} 1 & & & & 1 & & & & 1 & & & \\ & 1 & & & & 1 & & & & 1 & & \\ & & 1 & & & & 1 & & & & 1 & \\ & & & 1 & & & & 1 & & & & 1 \end{pmatrix}}_{\boldsymbol{A}_1} \boldsymbol{x} = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{pmatrix},$$
$$\underbrace{\begin{pmatrix} 1 & 1 & 1 & 1 & & & & & & & & \\ & & & & 1 & 1 & 1 & 1 & & & & \\ & & & & & & & & 1 & 1 & 1 & 1 \end{pmatrix}}_{\boldsymbol{A}_2} \boldsymbol{x} \le \begin{pmatrix} k_1 \\ k_2 \\ k_3 \end{pmatrix},$$
$$-\boldsymbol{x} \le \boldsymbol{0}.$$

(Prázdné prvky v maticích $\boldsymbol{A}_1, \boldsymbol{A}_2$ jsou nuly.) Pro obecné m, n uvážíme matice $\boldsymbol{A}_1, \boldsymbol{A}_2$ analogické struktury a získáme (8.1) v podobě

$$\min_{\boldsymbol{x}} \boldsymbol{c}'\boldsymbol{x} \ \text{ s.t. } \underbrace{(\boldsymbol{I}_{n\times n} \quad \boldsymbol{I}_{n\times n} \quad \cdots \quad \boldsymbol{I}_{n\times n})}_{\boldsymbol{A}_1} \boldsymbol{x} = \boldsymbol{p}, \quad \underbrace{\begin{pmatrix} \boldsymbol{e}'_{1\times n} & & & \\ & \boldsymbol{e}'_{1\times n} & & \\ & & \ddots & \\ & & & \boldsymbol{e}'_{1\times n} \end{pmatrix}}_{\boldsymbol{A}_2} \boldsymbol{x} \le \boldsymbol{k}, \quad -\boldsymbol{x} \le \boldsymbol{0}. \tag{8.4}$$

(Aby nedošlo k nedorozumění: $\boldsymbol{e}'_{1\times m}$ je řádkový vektor m jedniček.) K vyřešení dopravního problému zřejmě stačí volat `linprog(c, [A2; -eye(m * n)], [k; zeros(m * n,1)], A1, p)`. Jak lze vytvořit vektor $\boldsymbol{c}$ a matice $\boldsymbol{A}_1$, $\boldsymbol{A}_2$? Jedna možnost je pochopitelně pomocí `for`-cyklů, například

```
c  = [];  for j=1:m; C  = [C; c(:,j)]; end;
A1 = [];  for j=1:m; A1 = [A1, eye(n)]; end;
A2 = [];  for j=1:m; A2 = [A2, zeros(j-1,n); zeros(n*(j-1),1), ones(1,n)]; end;
```

Existují ale i elegantnější způsoby. Příkaz `c = reshape(C, [m * n, 1])` učiní přesně operaci „přeskládání“ matice $\boldsymbol{C}$ do vektoru rozměru $mn \times 1$ podle (8.3). Matice $\boldsymbol{A}_1$ a $\boldsymbol{A}_2$ lze jednoduše vytvořit pomocí tzv. Kroneckerova součinu. Jsou-li dány matice $\boldsymbol{U} = (u_{ij}) \in \mathbb{R}^{k_1\times\ell_1}$ a $\boldsymbol{V} \in \mathbb{R}^{k_2\times\ell_2}$, pak *Kroneckerův součin* $\boldsymbol{U} \otimes \boldsymbol{V}$ je matice rozměru $k_1k_2 \times \ell_1\ell_2$ s bloky

$$\boldsymbol{U} \otimes \boldsymbol{V} = \begin{pmatrix} u_{11}\boldsymbol{V} & u_{12}\boldsymbol{V} & \cdots & u_{1,\ell_1}\boldsymbol{V} \\ u_{21}\boldsymbol{V} & u_{22}\boldsymbol{V} & \cdots & u_{2,\ell_1}\boldsymbol{V} \\ \vdots & \vdots & \ddots & \vdots \\ u_{k_1,1}\boldsymbol{V} & u_{k_1,2}\boldsymbol{V} & \cdots & u_{k_1,\ell_1}\boldsymbol{V} \end{pmatrix},$$

například

$$\begin{pmatrix} 1 & 2 & 3 & 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 2 & 0 & 3 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 2 & 0 & 3 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

V MATLABu se Kroneckerův součin počítá pomocí funkce `kron`. Z (8.4) je patrno, že

$$\boldsymbol{A}_1 = \boldsymbol{e}'_{1\times m} \otimes \boldsymbol{I}_{n\times n}, \quad \boldsymbol{A}_2 = \boldsymbol{I}_{m\times m} \otimes \boldsymbol{e}'_{1\times n},$$

a tedy můžeme matice $\boldsymbol{A}_1$, $\boldsymbol{A}_2$ vytvořit stručným příkazem

$$A_1 = \texttt{kron(ones(1,}m\texttt{), eye(}n\texttt{))}, \quad A_2 = \texttt{kron(eye(}m\texttt{), ones(1,}n\texttt{))}.$$

Nyní již máme všechny ingredience a můžeme napsat jednoduchý solver pro dopravní problém. Uživatel zadá (sloupcové) vektory $\boldsymbol{k}, \boldsymbol{p}$ a matici $\boldsymbol{C}$ a obdrží matici $\boldsymbol{X} = (\boldsymbol{x}_{ij}^*)$ s optimální přepravou.

Transport.m: Solver pro dopravní problém

```
3  function X = SolveTP(k, p, C)
4    [n,m] = size(C);
5    c = reshape(C,[m*n, 1]);
6    A1 = kron(ones(1,m), eye(n));
7    A2 = kron(eye(m), ones(1,n));
8    x = linprog(c, [A2; -eye(m*n)], [k; zeros(m*n,1)], ...
9      A1, p);
10   X = reshape(x,[n,m]);
11 end
```

Příkaz 10 jen „přeskládá“ získané optimální řešení `x` tvaru (8.2) do matice `X` tvaru $n \times m$.

Poznámka

Stejně — jako v dalších kapitolách — i zde je vhodné si jako cvičení rozmyslet, jak by vypadalo řešení rozličných modifikací dopravního problému. Například: řekneme, že instance $(\boldsymbol{k}, \boldsymbol{p}, \boldsymbol{C})$ dopravního problému vykazuje tzv. *more-for-less paradox*, jestliže je možné zvýšením kapacit (některých) dodavatelů a adekvátním zvýšením požadavků (některých) odběratelů dosáhnout přepravy celkově většího množství *a přitom snížit celkové přepravní náklady*. Zdali more-for-less paradox nastává, lze zjistit pomocí vhodného LP — jako cvičení je možné rozšířit skript `SolveTP`, aby uživateli podal tuto informaci. Podobně je možné rozšířit skript o vizualizaci výsledné optimální přepravy jako v kapitole 5; umístění dodavatelů a odběratelů lze vykreslit coby body v rovině a mezi nimi zobrazit šipky, jejichž síla odpovídá transportovanému množství. A konečně zájemce se může podívat na toolbox *mapping* — pomocí funkcí `worldmap` a `geoshow` lze vykreslit (například) mapu konkrétního státu, a pak je možné toky přepravy zobrazovat na reálné mapě. (Kreslení map ovšem přesahuje rámec tohoto textu.)

Doporučená literatura

- Dupačová, J., Lachout, P. 2011. *Úvod do optimalizace*. MatfyzPress. ISBN 978-80-7378-176-7. http://matfyzpress.cz/matematika/20-uvod-do-optimalizace.html
- Hillier, F. S., Lieberman, G. J. 2018. *Introduction to Operations Research*. McGraw-Hill Education. ISBN 978-0-07-352345-3. http://highered.mheducation.com/sites/0073523453/information{_}center{_}view0/index.html
- Jablonský, J. 2007. *Operační výzkum: Kvantitativní modely pro ekonomické rozhodování*. Professional Publishing. ISBN 978-80-86946-44-3. https://www.researchgate.net/publication/40354299
- Kwon, R. H. 2013. *Introduction to Linear Optimization and Extensions with MATLAB*. CRC Press. ISBN 978-1-4398-6264-3. https://doi.org/10.1201/b13966
- Vanderbei, R. J. 2020. *Linear Programming: Foundations and Extensions*. Springer. ISBN 978-3-030-39414-1. https://doi.org/10.1007/978-3-030-39415-8

Kapitola 9

Analýza obalu dat

Metoda analýzy obalu dat slouží k ohodnocení efektivity sledovaných jednotek pomocí různých vstupů a výstupů. V této kapitole si ukážeme jak takovou úlohu řešit s využitím lineárního programování.

Klíčová slova: *data envelopment analysis (DEA), efektivnost, rankovací metoda, lineární programování.*

9.1 Formulace problému

DEA je oblíbená rankovací metoda. Existuje v nepřeberném množství variant a modifikací; my zde ukážeme jen jednoduchý základní model.

Data Envelopment Analysis

Je dán systém p jednotek (tzv. *decision making unit, DMU*), z nichž každá spotřebovává jisté vstupy a generuje jisté výstupy, a jednotky jsou homogenní v tom smyslu, že generují stejné typy výstupů při stejných typech vstupů. Jednotkou může být v konkrétní aplikaci takřka cokoliv, o čem lze rozumně prohlásit, že spotřebovává vstupy a transformuje je na výstupy. Může jít o podniky ve stejném odvětví (vstupy jsou například vybavenost pracovní silou a kapitálem, výstupy jsou výrobky několika druhů); může jít o nemocnice (vstupem jsou např. počty lékařů, sester a vybavenost technikou; výstupy jsou objemy hospitalizací, počty operací, ambulatní výkony atd.), může jít o lidi (u učitele může být vstupem mzda, úroveň kvalifikace a délka praxe, výstupy mohou být objem vědeckých výkonů a objem pedagogických výkonů); může jít o státy či regiony (vstupem je např. struktura pracovní síly, vybavenost infrastrukturou apod., výstupem je HDP).

Cílem DEA je stanovit *relativní srovnání* (ranking) daného systému jednotek, u nichž známe vstupy a výstupy. Potíž je, že vstupy i výstupy jsou různého druhu; je-li jednotkou telekomunikační společnost, jejími výstupy může být objem datových služeb a objem hlasových služeb *a není jasné, podle kterého kritéria jednotky srovnávat* — jedna jednotka může být „lepší“ v datových službách, jiná v hlasových službách, a není jasné, jak nesrovnatelná kritéria sloučit a udělat jednoznačné porovnání.

V rámci DEA metodiky se definuje *efektivita* jednotky jako *vážený součet výstupů ku váženému součtu vstupů*. Označme $\boldsymbol{a}_k \geq \mathbf{0}$ vektor výstupů a $\boldsymbol{b}_k \geq \mathbf{0}$ vektor vstupů k-té jednotky ($k = 1, \dots, p$); ty jsou uživatelem zadány. Efektivita k-té jednotky je podle definice

$$\mathrm{ef}_k = \frac{\boldsymbol{a}_k' \boldsymbol{v}}{\boldsymbol{b}_k' \boldsymbol{w}}, \tag{9.1}$$

kde $\boldsymbol{v} \geq \mathbf{0}$ a $\boldsymbol{w} \geq \mathbf{0}$ jsou vektory vah (později váhy omezíme podmínkou zajišťující $\mathrm{ef}_k \in [0, 1]$). Vše by bylo jednoduché, kdyby váhy $\boldsymbol{v}, \boldsymbol{w}$ byly známé — kdyby byl někdo třeba schopen říci, že jeden gigabyte přenesených dat je ekvivalentní deseti hlasovým spojením; kdyby byl někdo schopen říci, že jedna transplantace je ekvivalentní deseti ambulatním vyšetřením; nebo kdyby byl někdo schopen říci, že učitel vychovávající tři doktorandy je ekvivalentní učiteli publikujícímu jednu vědeckou monografii. Pak by nebylo co řešit: prostě bychom jednotky mohli srovat podle efektivit (9.1) spočtených se známými váhami. Zde přichází hlavní idea DEA metodiky. Váhy jsou neznámé; dovolme tedy jednotce $k \in \{1, \dots, p\}$, ať si *zvolí váhy* $\boldsymbol{v}, \boldsymbol{w}$ *sama, a to tak, aby to pro ni bylo co nejvýhodnější* (aby dosáhla co nejvyšší efektivity). Nicméně poté musí své váhy předložit ostatním jednotkám; pokud by některá jiná jednotka dosáhla při těchto vahách lepšího efektivitního poměru, prohlásíme jednotku k za *DEA-neefektivní* ($\mathrm{ef}_k < 1$); jinak ji prohlásíme za *DEA-efektivní* ($\mathrm{ef}_k = 1$).

Vektory zadaných výstupů a vstupů uspořádejme do matic

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{a}_1' \\ \boldsymbol{a}_2' \\ \vdots \\ \boldsymbol{a}_p' \end{pmatrix} \in \mathbb{R}^{p \times n}, \quad \boldsymbol{B} = \begin{pmatrix} \boldsymbol{b}_1' \\ \boldsymbol{b}_2' \\ \vdots \\ \boldsymbol{b}_p' \end{pmatrix} \in \mathbb{R}^{p \times m}$$

(dvojici $\boldsymbol{B}, \boldsymbol{A}$ se také říká *input-output matice*). Symbolem n jsme označili počet výstupů a symbolem m jsme označili počet vstupů. Ideu DEA — ať si k-tá jednotka zvolí své váhy, jak nejlépe dokáže — snadno napíšeme jako (nelineární) optimalizační problém

$$\max_{\substack{\boldsymbol{v} \in \mathbb{R}^n \\ \boldsymbol{w} \in \mathbb{R}^m}} \frac{\boldsymbol{a}_k' \boldsymbol{v}}{\boldsymbol{b}_k' \boldsymbol{w}} \quad \text{s.t.} \quad 0 \overset{(\star)}{\leq} \frac{\boldsymbol{a}_\ell' \boldsymbol{v}}{\boldsymbol{b}_\ell' \boldsymbol{w}} \overset{(\dagger)}{\leq} 1 \ \ (\forall \ell = 1, \dots, p), \quad \boldsymbol{v}, \boldsymbol{w} \geq \mathbf{0}. \tag{9.2}$$

Omezující podmínky (†) vlastně formalizují test, kdy se váhy $\boldsymbol{v}, \boldsymbol{w}$ volené jednotkou k předkládají jednotkám $\ell = 1, \dots, p$ a všechny efektivity se normalizují na škálu $[0, 1]$. Výsledný DEA-ranking se získá tak, že každá jednotka $k = 1, \dots, p$ vyřeší problém (9.2) a jednotky se pak seřadí podle dosažených efektivit.

9.2 Linearizace úlohy

Problém (9.2) se snadno zlinearizuje. Omezení (⋆) jsou redundantní, čísla $\boldsymbol{a}'_\ell \boldsymbol{v}, \boldsymbol{b}'_\ell \boldsymbol{w}$ jsou totiž nezáporná. Díky nezápornosti $\boldsymbol{b}'_\ell w$ můžeme (†) přepsat do lineárního tvaru

$$\boldsymbol{a}'_\ell \boldsymbol{v} \le \boldsymbol{b}'_\ell \boldsymbol{w}.$$

Všimněme si, že váhy nejsou nikdy jednoznačné — hodnota výrazu $\frac{\boldsymbol{a}'_k \boldsymbol{v}}{\boldsymbol{b}'_k \boldsymbol{w}}$ se nezmění, nahradíme-li váhy $\boldsymbol{v}, \boldsymbol{w}$ váhami $\alpha\boldsymbol{v}, \alpha\boldsymbol{w}$ pro libovolné $\alpha > 0$. Proto můžeme bez újmy na obecnosti váhy standardizovat; hodí se omezit se jen na váhy splňující $\boldsymbol{b}'_k \boldsymbol{w} = 1$. Tím získáme lineární program

$$(\mathrm{DEA}_k) \quad \max_{\substack{\boldsymbol{v}\in\mathbb{R}^n \\ \boldsymbol{w}\in\mathbb{R}^m}} \boldsymbol{a}'_k \boldsymbol{v} \text{ s.t. } \boldsymbol{a}'_\ell \boldsymbol{v} \le \boldsymbol{b}'_\ell \boldsymbol{w} \ \ (\forall \ell = 1, \dots, p), \ \ \boldsymbol{b}'_k \boldsymbol{w} = 1, \ \ \boldsymbol{v}, \boldsymbol{w} \ge \boldsymbol{0}. \tag{9.3}$$

Nyní se vyřeší řada LP $(\mathrm{DEA}_1), \dots, (\mathrm{DEA}_p)$ a jejich optimální účelové hodnoty (= efektivity) dají výsledný ranking.

Přepišme (9.3) do maticového tvaru vhodného pro solver `linprog(c, D, b, U, z)`:

$$\min_{\boldsymbol{x}=\binom{\boldsymbol{v}}{\boldsymbol{w}}} \underbrace{\begin{pmatrix} -\boldsymbol{a}'_k & \boldsymbol{0}_{1\times m} \end{pmatrix}}_{\boldsymbol{c}'} \underbrace{\begin{pmatrix} \boldsymbol{v} \\ \boldsymbol{w} \end{pmatrix}}_{\boldsymbol{x}} \text{ s.t. } \underbrace{\begin{pmatrix} \boldsymbol{A} & -\boldsymbol{B} \\ -\boldsymbol{I}_{n\times n} & \boldsymbol{0}_{n\times m} \\ \boldsymbol{0}_{m\times n} & -\boldsymbol{I}_{m\times m} \end{pmatrix}}_{\boldsymbol{D}} \underbrace{\begin{pmatrix} \boldsymbol{v} \\ \boldsymbol{w} \end{pmatrix}}_{\boldsymbol{x}} \le \underbrace{\begin{pmatrix} \boldsymbol{0}_{p\times 1} \\ \boldsymbol{0}_{n\times 1} \\ \boldsymbol{0}_{m\times 1} \end{pmatrix}}_{\boldsymbol{b}}, \ \underbrace{\begin{pmatrix} \boldsymbol{0}_{1\times n} & \boldsymbol{b}'_k \end{pmatrix}}_{\boldsymbol{U}} \underbrace{\begin{pmatrix} \boldsymbol{v} \\ \boldsymbol{w} \end{pmatrix}}_{\boldsymbol{x}} = \underbrace{1}_{\boldsymbol{z}}. \tag{9.4}$$

Napišme skript, kde uživatel na vstup zadá matice výstupů a vstupů $(\boldsymbol{A}, \boldsymbol{B})$ a výsledkem bude vektor `ef` efektivit jednotek. Ve `for`-cyklu budeme řešit lineární program (DEA_k) pro $k = 1, \dots, p$. Všimněme si v (9.4), že $\boldsymbol{D}, \boldsymbol{b}$ nezávisí na k; můžeme si tedy $\boldsymbol{D}, \boldsymbol{b}$ připravit ještě před `for`-cyklem.

DEA.m: Solver pro DEA

```
3  function ef = SolveDEA(A, B)
4    [p, n] = size(A);
5    [p, m] = size(B);
6    D = [A, -B; -eye(n + m)];
7    b = zeros(p + n + m,1);
8    for k = 1:p
9      c = [-A(k, :), zeros(1, m)]';
10     U = [zeros(1, n), B(k, :)];
11     x = linprog(c, D, b, U, 1);
12     ef(k) = A(k, :) * x(1:n);
13     % ulozili jsme optimalni hodnotu ucelove funkce, ...
14       % A(k, :) * x(1:n) je hodnota ucel. funkce, ...
15       % x(1:n) jsou optimalni vahy v
16   end
17 end
```

Poznámka

Často je rozumné předpokládat, že data $(\boldsymbol{A}, \boldsymbol{B})$ pro DEA analýzu jsou náhodné veličiny: počty telefonních hovorů, počty vyšetření pacientů či počty vychovaných studentů totiž často lze chápat jako výsledek jistého náhodného procesu (například to, kolik přijde pacientů do nemocnice, je do jisté míry náhoda). Formálně: předpokládejme, že data $(\boldsymbol{A}, \boldsymbol{B})$ jsou náhodné veličiny se známou distribucí (např. empirickou distribucí získanou měřením počtu příchozích pacientů v několika obdobích). Pak i efektivity jsou náhodné veličiny a je rozumné nasimulovat jejich rozdělení podobně jako v kapitolách 3.4 a 6. Tato simulace nám podá informaci například o *robustnosti* či *stabilitě* DEA-rankingu. Zůstane-li totiž jednotka DEA-efektivní i poté, co (náhodně) perturbujeme data — a modeluje-li tato perturbace, co se v realitě skutečně může stát —, ukazuje to, že její DEA-efektivita zůstává stabilní, i když se vnější podmínky mění. To je kvalitativní rozdíl oproti jednotce, která je sice při daných datech $(\boldsymbol{A}, \boldsymbol{B})$ DEA-efektivní, ovšem při náhodných perturbacích dat vykazuje její DEA-efektivita velký rozptyl.

Doporučená literatura

- Coelli, T. J., Prasada Rao, D. S., O'Donnell, C. J., Battese, G. E. 2005. *An Introduction to Efficiency and Productivity Analysis*. Springer. ISBN 978-0-387-24265-1. https://doi.org/10.1007/b136381
- Cooper, W. W., Seiford, L. M., Tone, K. 2007. *Data Envelopment Analysis: A Comprehensive Text with Models, Applications, References and DEA-Solver Software*. Springer. ISBN 978-0-387-45281-4. https://doi.org/10.1007/978-0-387-45283-8
- Fried, H. O., Knox Lovell, C. A., Schmidt, S. S. 2008. *The Measurement of Productive Efficiency and Productivity Change*. Oxford University Press. ISBN 978-019987028-8. https://doi.org/10.1093/acprof:oso/9780195183528.001.0001
- Jablonský, J., Dlouhý, M. 2016. *Modely hodnocení efektivnosti a alokace zdrojů*. Professional Publishing. ISBN 978-80-7431-155-0. http://www.profespubl.cz/titulka/modely-hodnoceni-efektivnosti-a-alokace-zdroju/
- Ray, S. C. 2004. *Data Envelopment Analysis: Theory and Techniques for Economics and Operations Research*. Cambridge University Press. ISBN 978-0-521-80256-7. https://doi.org/10.1017/cbo9780511606731

Kapitola 10

Celočíselné lineární programování

V této kapitole se budeme zabývat celočíselným programováním. Naprogramujeme si vlastní řešitel založený na algoritmu větvení a mezí, který pak aplikujeme na jednoduchou úlohu batohu.

Klíčová slova: *celočíselné programování, metoda větvení a mezí, branch and bound, úloha batohu.*

10.1 Celočíselné programování v MATLABu

Až dosud jsme se zabývali lineárním programováním se spojitými proměnnými ($\boldsymbol{x} \in \mathbb{R}^n$). MATLAB disponuje i solverem pro celočíselné a smíšené lineární programování (*ILP*, Integer Linear Programming; *MILP*, Mixed Integer Linear Programming).

Funkce `intlinprog`

Solver pro celočíselné a smíšené lineární programování má základní syntaxi analogickou solveru `linprog`:

```
intlinprog(c, I, A, b, U, v),
```

která řeší problém

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \boldsymbol{c}'\boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \\ & \boldsymbol{U}\boldsymbol{x} = \boldsymbol{v}, \\ & x_i \in \begin{cases} \mathbb{Z}, & i \in I, \\ \mathbb{R}, & i \notin I. \end{cases} \end{aligned}$$

Tedy: I je množina (vektor) indexů proměnných, které mají nabývat celočíselných hodnot. Mají-li být všechny proměnné celočíselné, stačí volit `I = [1:n]'`, kde n je počet proměnných.

Poznámka

Jako cvičení nyní doporučujeme vyzkoušet `intlinprog` na běžných problémech formulovatelných jako ILP, například problém batohu, problém obchodního cestujícího, job scheduling či různé další typy přiřazovacích problémů.

Poznámka

Připomeňme, že narozdíl od (spojitého) lineárního programování je celočíselné lineární programování obecně **NP**-těžké; není tedy snadné řešit rozsáhlejší instance. Byť je solver v MATLABu dobrý, patrně se nemůže srovnávat s CPLEXem a dalším podobným software (ovšem toto tvrzení by by bylo třeba potvrdit řádnou srovnávací studií). Potřebuje-li uživatel řešit reálné problémy ILP, vždy se doporučuje využít raději CPLEX či další speciální software.

10.2 Úvod k B&B algoritmu

My si v této kapitole zkusíme vytvořit vlastní jednoduchý solver pro ILP založený na metodě Branch-and-Bound (B&B). Pro řešení praktických problémů je to čirý nerozum: nikdy se nám totiž nepodaří naprogramovat něco, co by alespoň vzdáleně dokázalo konkurovat profesionálnímu software, a rozhodně se nic takového nedoporučuje. Psát vlastní solver má smysl jen tehdy, chce-li si uživatel vyzkoušet vlastnosti algoritmu a „hrát“ si s ním — chce-li jeho práci vizualizovat, nebo chce-li například zkoušet porovnávat efektivitu různých strategií volby indexu pro B&B-branching a tak podobně.

Připomeňme, že B&B algoritmus pro ILP je následující procedura, která instanci ILP redukuje na výpočet řady spojitých lineárních programů. Z technických důvodů bude na tomto místě pro nás jednodušší řešit ILP ve tvaru

$$\min_{\boldsymbol{x}\in\mathbb{Z}^n} \boldsymbol{c}'\boldsymbol{x} \text{ s.t. } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b},\ \boldsymbol{U}\boldsymbol{x} = \boldsymbol{v},\ \underline{\boldsymbol{x}} \leq \boldsymbol{x} \leq \overline{\boldsymbol{x}}. \tag{10.1}$$

Napíšeme skript

$$\texttt{x = MyBaB(}c\texttt{, }A\texttt{, }b\texttt{, }U\texttt{, }v\texttt{, }\underline{x}\texttt{, }\overline{x}\texttt{)},$$

který tuto úlohu řeší a vrací optimální řešení jako vektor `x`. Všimněme si, že díky existenci dolních a horních mezí $\underline{\boldsymbol{x}}, \overline{\boldsymbol{x}}$ se nemusíme zabývat neomezeností (to je jen proto, aby náš skript byl co nejjednodušší — nicméně jako cvičení nechť čtenář rozpracuje problém v plné obecnosti).

Jako podprogram budeme užívat standardní solver `linprog`, tentokrát v syntaxi

$$\texttt{[x, ov, flag] = linprog(c, A, b, U, v, }\underline{x}\texttt{, }\overline{x}\texttt{)},$$

který řeší LP

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \boldsymbol{c}'\boldsymbol{x} \text{ s.t. } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b},\ \boldsymbol{U}\boldsymbol{x} = \boldsymbol{v},\ \underline{\boldsymbol{x}} \leq \boldsymbol{x} \leq \overline{\boldsymbol{x}} \tag{10.2}$$

(to je tzv. *relaxace* problému (10.1)). Výsledkem výpočtu je optimální řešení `x`, optimální hodnota účelové funkce `ov` a informace o způsobu ukončení výpočtu `flag`. Připomeňme, že hodnota $\texttt{flag} = 1$ znamená, že bylo nalezeno optimum. Protože v našem případě nemůže dojít k neomezenému případu, hodnoty `flag` různé od jedné ztotožníme s nepřípustností (abstrahujeme zde od toho, že některé hodnoty `flag` mohou signalizovat numerické problémy).

10.3 B&B algoritmus

Během výpočtu budeme udržovat seznam $\mathfrak{L}$ lineárních programů. Seznam $\mathfrak{L}$ je na začátku prázdný.

Vstup: data $\boldsymbol{c}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{U}, \boldsymbol{v}, \underline{\boldsymbol{x}}, \overline{\boldsymbol{x}}$ problému (10.1).
Krok 1. Je-li LP (10.2) nepřípustný, skončíme — pak je totiž nepřípustný také ILP (10.1). Je-li LP (10.2) přípustný, vložíme jej do seznamu $\mathfrak{L}$.
Krok 2. Je-li seznam $\mathfrak{L}$ prázdný, skončíme — problém (10.1) je nepřípustný.
Krok 3. Je-li seznam $\mathfrak{L}$ neprázdný, nalezneme v něm ten lineární program $\{\min_{\boldsymbol{x}\in\mathbb{R}} \boldsymbol{c}'\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b},\ \boldsymbol{U}\boldsymbol{x} = \boldsymbol{v},\ \underline{\boldsymbol{x}}^0 \leq \boldsymbol{x} \leq \overline{\boldsymbol{x}}^0\}$, který má mezi všemi LP v seznamu $\mathfrak{L}$ nejmenší optimální hodnotu účelové funkce. Říkejme mu (LP^0).
Krok 4. Odstraňme (LP^0) ze seznamu $\mathfrak{L}$.
Krok 5. Nechť $\boldsymbol{x}^*$ je optimální řešení (LP^0). Je-li $\boldsymbol{x}^*$ celočíselné, skončíme — nalezli jsme optimum (10.1).
Krok 6. Nechť i^0 je libovolný index takový, že $\boldsymbol{x}^*_{i^0}$ není celočíselné. Zaveďme lineární programy

$$(LP') \quad \min_{\boldsymbol{x}\in\mathbb{R}^n} \boldsymbol{c}'\boldsymbol{x} \text{ s.t. } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b},\ \boldsymbol{U}\boldsymbol{x} = \boldsymbol{v},\ \underline{\boldsymbol{x}}^0 \leq \boldsymbol{x} \leq \overline{\boldsymbol{x}}^0,\ x_{i^0} \geq \lceil x^*_{i^0} \rceil, \tag{10.3}$$

$$(LP'') \quad \min_{\boldsymbol{x}\in\mathbb{R}^n} \boldsymbol{c}'\boldsymbol{x} \text{ s.t. } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b},\ \boldsymbol{U}\boldsymbol{x} = \boldsymbol{v},\ \underline{\boldsymbol{x}}^0 \leq \boldsymbol{x} \leq \overline{\boldsymbol{x}}^0,\ x_{i^0} \leq \lfloor x^*_{i^0} \rfloor. \tag{10.4}$$

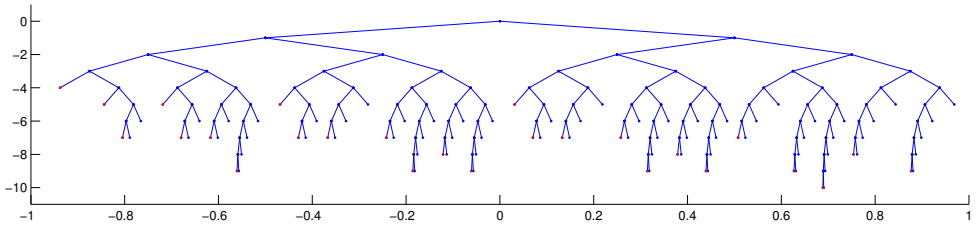
Zde $\lfloor \xi \rfloor = \max\{z \in \mathbb{Z} \mid z \leq \xi\}$ značí dolní celou část čísla ξ (funkce `floor`) a $\lceil \xi \rceil = \min\{z \in \mathbb{Z} \mid z \geq \xi\}$ značí horní celou část čísla ξ (funkce `ceil`).
Krok 7. Jestliže je (LP') přípustný, přidejme jej do seznamu $\mathfrak{L}$.
Krok 8. Jestliže je (LP'') přípustný, přidejme jej do seznamu $\mathfrak{L}$.
Krok 9. Pokračujeme Krokem 2.

Poznámka

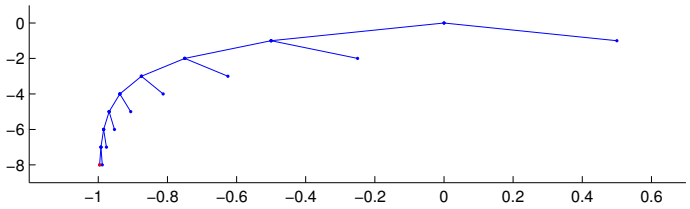
Konstrukci (10.3) a (10.4) lze udělat tak, že se jen zvýší dolní mez $\underline{x}^0_{i^0}$ na hodnotu $\lceil x^*_{i^0} \rceil$ (v případě (10.3)) a sníží se horní mez $\overline{x}^0_{i^0}$ na hodnotu $\lfloor x^*_{i^0} \rfloor$ (v případě (10.4)). To znamená, že všechny lineární programy v seznamu $\mathfrak{L}$ mají shodná data $\boldsymbol{c}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{U}, \boldsymbol{v}$ a liší se jen v mezích $\underline{\boldsymbol{x}}^0, \overline{\boldsymbol{x}}^0$ (a pochopitelně také v optimálních řešeních $\boldsymbol{x}^*$ a optimálních hodnotách účelové funkce $\boldsymbol{c}'\boldsymbol{x}^*$). Proto můžeme reprezentovat LP v seznamu $\mathfrak{L}$ jen pomocí čtveřice údajů $(\underline{\boldsymbol{x}}^0, \overline{\boldsymbol{x}}^0, \boldsymbol{x}^*, \boldsymbol{c}'\boldsymbol{x}^*)$. Konkrétně: uděláme to tak, že $\underline{\boldsymbol{x}}^0$ budeme uchovávat jako sloupce speciální matice `LLB` („List of Lower Bounds“), $\overline{\boldsymbol{x}}^0$ budeme uchovávat jako sloupce speciální matice `LUB` („List of Upper Bounds“), $\boldsymbol{x}^*$ budeme uchovávat jako sloupce speciální matice `Lx` a hodnoty účelové funkce $\boldsymbol{c}'\boldsymbol{x}^*$ budeme uchovávat jako prvky vektoru `Lov` („List of optimal values“).

Poznámka

Kroky 4 a 7–8 se nazývají *branching* (*větvení*) — nahrazujeme totiž (LP^0) dvojicí (LP'), (LP''). Protože v Kroku 1 začínáme s jediným LP v seznamu $\mathfrak{L}$, lze postupné větvení kreslit jako strom, tzv. *branching tree*. Ten si později ve skriptu také nakreslíme. Vrcholem ve stromě je (LP^0). Vrchol má dva syny, levého a pravého; je-li (LP') přípustný, je levým synem s modrou barvou, a je-li (LP') nepřípustný, je levým synem (a listem) s červenou barvou. Je-li (LP'') přípustný, je pravým synem s modrou barvou, a je-li (LP'') nepřípustný, je pravým synem (a listem) s červenou barvou. Kořenem je LP (10.2). Strom jistě může mít i modré listy — to jsou přípustné LP, které se ovšem dále nevětví (a jejichž zpracování tudíž B&B algoritmus „ušetřil“). Následující obrázky ilustrují dva příklady, jak konkrétně mohou stromy vypadat; druhý obrázek ukazuje příklad výpočtu, který je „velice úsporný“.



Obrázek 10.1: Příklad rozvětveného stromu.



Obrázek 10.2: Příklad úsporného stromu.

Poznámka

Výběr indexu i^0 (tzv. *branching index*) v Kroku 6 nemusí být jednoznačný. Skutečně, vektor $\boldsymbol{x}^*$ má často více neceločíselných složek. V principu můžeme vybrat libovolnou z nich. Metoda výběru i^0, je-li více možností, se nazývá *branching strategy* a různými volbami lze získat různé varianty B&B; dokonce stojí za to empiricky testovat, které strategie fungují „lépe“ a „hůře“ (to ponecháme jako cvičení). My vybereme i^0 takto: spočteme vektor $\boldsymbol{x}' = \boldsymbol{x}^* - \texttt{round}(\boldsymbol{x}^*)$ (funkce `round` zaokrouhluje složky vektoru na nejbližší celé číslo). Vektor $\boldsymbol{x}'$ má tedy složky v intervalu $[\pm\frac{1}{2}]$ a nulové jsou právě ty složky, kde je $\boldsymbol{x}^*$ celočíselné. Pak spočteme $z = \max_i |x'_i|$. Je-li $z = 0$, je vektor $\boldsymbol{x}^*$ celočíselný; a je-li $z > 0$, za i^0 vezmeme ten index, pro nějž se maxima z nabývá. To je jistě legitimní branching strategie; nemá ovšem žádnou konkrétní výhodu, snad jen tu, že se snadno naprogramuje.

Poznámka

V numerických algoritmech bývají často výpočty přesné jen na určitý počet desetinných míst. Je-li správným výsledkem např. číslo 1, někdy můžeme od numerického algoritmu obdržet mírně odlišné číslo, např. 1 ± 10^{-15}. Abychom se nemuseli zabývat numerickými problémy, učiníme následující zjednodušení: zvolíme dostatečně malou chybu, řekněme 10^{-8}, a numericky spočtené číslo prohlásíme za celé, jestliže se od skutečně celého čísla liší nanejvýš o $\pm 10^{-8}$.

Ve skriptu budeme v proměnné `q` sledovat počet řešených LP — to jen kvůli informaci, kolik práce výpočet obnáší.

BaB.m: B&B krok 1.

```
3  function xopt = MyBaB(c, A, b, U, v, LB, UB)
4    % argumenty maji stejny vyznam jako u linprog
5    [x0, ov, flag] = linprog(c, A, b, U, v, LB, UB);
6    if flag ~= 1 % LP je nepripustny
7      disp('infeasible!');
8      return; % skoncit
9    end
10   LLB = LB;
11   LUB = UB;
12    Lx = x0;
13   Lov = ov;
```

BaB.m: B&B krok 2, začátek.

```
15   n = 1; % n = pocet prvku seznamu L
16   q = 1; % pocitadlo LP - zatim jsme vyresili jeden LP
17   while n >= 1 % dokud je seznam L neprazdny
```

BaB.m: B&B krok 3.

```
19       [fmin, j] = min(Lov); % j je index LP v seznamu L...
20         % s nejmensi ucelovou hodnotou
21       LB0 = LLB(:, j);
22       UB0 = LUB(:, j);
23       xstar = Lx(:, j);
24       [z, i0]=max(abs(xstar - round(xstar)));
25         % i0 = branching index
```

BaB.m: B&B krok 4.

```
27       LLB = LLB(:, [1:(j - 1), (j + 1):n]);
28       LUB = LUB(:, [1:(j - 1), (j + 1):n]);
29        Lx =  Lx(:, [1:(j - 1), (j + 1):n]);
30       Lov = Lov([1:(j - 1), (j + 1):n]);
```

BaB.m: B&B krok 5.

```
32       if z <= 10^-8  % je xstar celociselne?
33         xopt = round(xstar); % vystupem je xopt
34         disp(['Optimal objective value: ', num2str(fmin)]);
35         disp(['LPs solved: ', num2str(q)]);
36         return; % skoncit
37       end
```

BaB.m: B&B krok 6.

```
39      UB1 = UB0;
40      UB1(i0) = floor(xstar(i0));
41      LB1 = LB0;
42      LB1(i0) = ceil(xstar(i0));
43      [x1, ov1, flag1] = linprog(c, A, b, U, v, LB1, UB0);
44      [x2, ov2, flag2] = linprog(c, A, b, U, v, LB0, UB1);
45      q = q + 2; % pocitadlo - vyresili jsme dva LP
```

BaB.m: B&B krok 7.

```
47      if flag1 == 1 % (LP') je pripustny, ...
48        % (LP') vkladame do seznamu L
49        LLB = [LLB, LB1];
50        LUB = [LUB, UB0];
51         Lx = [ Lx,  x1];
52        Lov = [Lov; ov1];
53      end
```

BaB.m: B&B krok 8.

```
55      if flag2 == 1 % (LP'') je pripustny, ...
56        % (LP'') vkladame do seznamu L
57        LLB = [LLB, LB0];
58        LUB = [LUB, UB1];
59         Lx = [ Lx,  x2];
60        Lov = [Lov; ov2];
61      end
```

BaB.m: B&B krok 2, konec.

```
63      n = length(Lov); % n = pocet prvku seznamu L
64    end % konec hlavniho while cyklu
65    disp('infeasible!'); % seznam L je prazdny
66  end
```

10.4 Problém batohu

Použijme jednoduchý testovací celočíselný program — tzv. *problém batohu* s daty $\boldsymbol{c} \in \mathbb{R}^n$ a $\boldsymbol{w} \in \mathbb{R}$

$$\max_{\boldsymbol{x}\in\{0,1\}^n} \boldsymbol{c}'\boldsymbol{x} \text{ s.t. } \boldsymbol{c}'\boldsymbol{x} \leq \boldsymbol{w}. \tag{10.5}$$

Jde o tento problém: je dáno n kontejnerů s hmotnostmi $c_1, \ldots, c_n$ a loď s nosností w (v tomto kontextu se lodi někdy říká „batoh“ — odtud název problému). Cílem je určit, které kontejnery máme na loď naložit, aby naložená hmotnost byla co nejvyšší, ale přitom nepřesáhla nosnost w.

Vezměme pro příklad $n = 10$. Nechť $c_1, \ldots, c_{10}$ jsou náhodné hmotnosti z rovnoměrného rozdělení na intervalu $[0, 300]$.

BaB.m: Problém batohu, generování dat.

```
68 n = 10;
69 c = random('unif', 0, 300, [n, 1]);
```

Z volme $w = 1000$. Vyřešíme (10.5) pomocí našeho B&B-solveru:

BaB.m: Problém batohu, řešení.

```
72 x = MyBaB(-c, c', 1000, [], [], zeros(n, 1), ones(n, 1));
```

Za dolní a horní meze vkládáme vektory ze samých nul a jedniček. Systém omezení neobsahuje žádné rovnosti; proto je čtvrtý a pátý argument prázdný. Účelovou funkci píšeme s minusem, neboť jde o maximalizační problém, zatímco solver `MyBaB` pracuje s minimalizací.

Konkrétní výsledek samozřejmě závisí na hodnotách náhodných hmotností $\boldsymbol{c}$; výsledek může vypadat například takto (jen výsledné optimální řešení `x` zde pro úsporu místa uvádíme jako řádkový vektor, byť skript vrací vektor sloupcový):

```
Optimal objective value: -999.9542
LPs solved: 185
x = 1 0 1 0 1 1 1 0 0 0
```

Abychom ilustrovali, že náš solver skutečně *není* konkurenceschopný vůči profesionálnímu softwaru typu CPLEX, změřme výpočetní čas: funkce `tic` zapíná stopky a funkce `toc` vypíná stopky. Zadáme-li

BaB.m: Problém batohu, doba řešení.

```
71 tic;
72 x = MyBaB(-c, c', 1000, [], [], zeros(n, 1), ones(n, 1));
73 toc;
```

obdržíme informaci typu

```
Elapsed time is 1.988427 seconds.
```

To je dosti špatný výsledek pro problém batohu s deseti proměnnými!

10.5 Vizualizace B&B stromu

Nyní funkci `MyBaB` rozšíříme o dodatečné instrukce, které nakreslí strom větvení výpočtu ze strany 62. Kořen stromu nakreslíme jako bod v rovině se souřadnicemi $(x, y) = (0, 0)$. K němu si také připojíme údaj $d = 1$; údaj d řekne, že synové mají být nakresleni na souřadnicích $(x - \frac{d}{2}, y - 1)$ (levý syn) a $(x + \frac{d}{2}, y - 1)$ (pravý syn). Sestoupíme-li ve stromě o patro níže, bude třeba vzdálenost synů d zkrátit na polovinu (jak je patrné z obrázku ze strany 62).

Trojici údajů (x, y, d) budeme chápat jako dodatečnou informaci o lineárním programu v seznamu $\mathfrak{L}$. Budeme tuto trojici ukládat jako sloupcový vektor v pomocné matici `Lpl` („`pl`“ znamená: „data pro `plot`“).

Funkci `MyBaB` rozšíříme o dodatečné instrukce, zbytek programu zůstává nezměněn. Říkejme této rozšířené verzi solveru `MyBaBPlotTree`. Zavoláme-li jej opět na problém batohu obdržíme navíc obrázek podobného typu jako na straně 62.

BaBPlot.m: Modifikace kroku 1.

```
3  function xopt = MyBaB(c, A, b, U, v, LB, UB)
4    [x0, ov, flag] = linprog(c, A, b, U, v, LB, UB);
5    if flag ~= 1
6      disp('infeasible!');
7      return;
8    end
9    LLB = LB;
10   LUB = UB;
11    Lx = x0;
12   Lov = ov;
13   Lpl = [0; 0; 1];                                  % [nove]
14     % koren vykreslime na souradnice ...
15     % (0,0) a vzdalenost jeho synu ma byt d = 1
```

BaBPlot.m: Modifikace kroku 4.

```
28     % ze seznamu Lpl cteme rovinne souradnice...
29       % aktualniho vrcholu a hodnotu
30     px = Lpl(1, j);                                 % [nove]
31     py = Lpl(2, j);                                 % [nove]
32     pd = Lpl(3, j);                                 % [nove]
33     Lpl = Lpl(:, [1:(j - 1), (j + 1):n]);           % [nove]
34       % aktualni vrchol vyradit ze seznamu Lpl
35     LLB = LLB(:, [1:(j - 1), (j + 1):n]);
36     LUB = LUB(:, [1:(j - 1), (j + 1):n]);
37      Lx =  Lx(:, [1:(j - 1), (j + 1):n]);
38     Lov = Lov([1:(j - 1), (j + 1):n]);
```

BaBPlot.m: Modifikace kroku 6.

```
47     UB1 = UB0;
48     UB1(i0) = floor(xstar(i0));
49     LB1 = LB0;
50     LB1(i0) = ceil(xstar(i0));
51     [x1, ov1, flag1] = linprog(c, A, b, U, v, LB1, UB0);
52     [x2, ov2, flag2] = linprog(c, A, b, U, v, LB0, UB1);
53     q = q + 2;
54     plot([px, px - pd / 2], [py, py - 1], ...      % [nove]
55       'b.-' , [px, px + pd / 2], ...               % [nove]
56       [py, py - 1], 'b.-');                        % [nove]
57       % k bodu se souradnicemi (x,y) vykreslime oba
58       % syny (x - d / 2, y - 1)
```

BaBPlot.m: Modifikace kroku 7.

```
60      if flag1 == 1
61        LLB = [LLB, LB1];
62        LUB = [LUB, UB0];
63         Lx = [ Lx,  x1];
64        Lov = [Lov; ov1];
65        Lpl = [Lpl, [px - pd / 2; py - 1; pd / 2]]; % [nove]
66          % vloz (x,y,d) pro (LP') do seznamu LPl,...
67          % vzdalenost synu se zmensi na polovinu
68      else                                          % [nove]
69        plot(px - pd / 2, py - 1, '.r');            % [nove]
70          % je-li (LP') nepripustny, vykresli vrchol cervene
71      end
```

BaBPlot.m: Modifikace kroku 8.

```
73      if flag2 == 1
74        LLB = [LLB, LB0];
75        LUB = [LUB, UB1];
76         Lx = [ Lx,  x2];
77        Lov = [Lov; ov2];
78        Lpl = [Lpl, [px + pd / 2; py - 1; pd / 2]]; % [nove]
79          % vloz (x,y,d) pro (LP'') do seznamu LPl, ...
80          % vzdalenost synu se zmensi na polovinu
81      else                                          % [nove]
82        plot(px + pd / 2, py - 1, '.r');            % [nove]
83          % je-li (LP'') nepripustny, vykresli ...
84          % vrchol cervene
85      end
```

BaBPlot.m: Vykreslení branching tree.

```
91  n = 10;
92  c = random('unif', 0, 300, [n, 1]);
93  x = MyBaBPlotTree(-c, c', 1000, [], [],           % [nove]
94    zeros(n, 1), ones(n, 1));                       % [nove]
```

Doporučená literatura

- Conforti, M., Cornuejols, G., Zambelli, G. 2014. *Integer Programming*. Springer. ISBN 978-3-319-11007-3. https://doi.org/10.1007/978-3-319-11008-0
- Hillier, F. S., Lieberman, G. J. 2018. *Introduction to Operations Research*. McGraw-Hill Education. ISBN 978-0-07-352345-3. http://highered.mheducation.com/sites/0073523453/information{_}center{_}view0/index.html
- Jablonský, J. 2007. *Operační výzkum: Kvantitativní modely pro ekonomické rozhodování*. Professional Publishing. ISBN 978-80-86946-44-3. https://www.researchgate.net/publication/40354299
- Pelikán, J. 2001. *Diskrétní modely v operačním výzkumu*. Professional Publishing. ISBN 978-80-86419-17-6

Kapitola 11

Logistická regrese

Jsme v pozici banky, za kterou přijde klient s přáním půjčky. Zda klientovi peníze půjčíme, se rozhodneme podle pravděpodobnosti jeho schopnosti půjčku vrátit. Od klientů si vždy zjistíme jejich měsíční příjem a jejich majetek. Podle zkušeností s minulými klienty banky pak budeme posuzovat nové klienty.

Klíčová slova: *regrese s binární vysvětlovanou proměnnou, logistické rozdělení, Logistická regrese, metoda maximální věrohodnosti, nelineární programování.*

11.1 Modelování binární vysvětlované proměnné

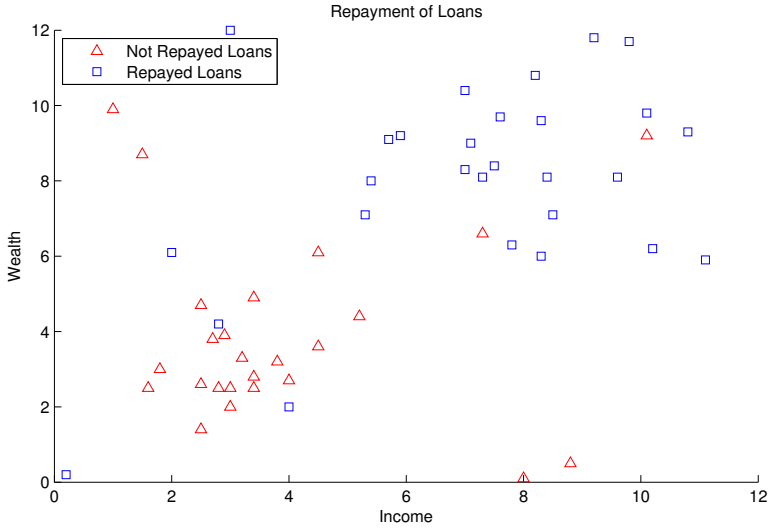
K dispozici máme historii n klientů, kteří si od nás v minulosti půjčili a u kterých známe jejich příjem $x_{i,1}$, jejich majetek $x_{i,2}$ a zda nám půjčku vrátili, či ne. Uvažujeme, že každý klient si od nás půjčí právě jednou, a velikost půjčky nezohledňujeme. Proměnnou představující výsledek půjčky si označíme y_i a tato tzv. binární proměnná nabývá hodnot

$$y_i = \begin{cases} 1, & \text{jestliže } i\text{-tý klient půjčku v pořádku splatil,} \\ 0, & \text{jestliže } i\text{-tý klient nesplatil (tyv. default).} \end{cases}$$

Nejdříve si naše data zobrazíme. Na horizontální osu si vyneseme příjmy klientů a na vertikální osu jejich majetky. Splacenou půjčku si označíme modrým čtverečkem a nesplacenou červeným trojuhelníkem.

Logit.m: Načtení dat

```
3 clients = xlsread('BankClients.csv');
4 clients(:, 2) = clients(:, 2) / 10e3;
5 clients(:, 3) = clients(:, 3) / 10e4;
6 clBad=clients(clients(:, 1) == 0, 2:3);
7 clGood=clients(clients(:, 1) == 1, 2:3);
```



Obrázek 11.1: Výsledky půjček klientů.

Logit.m: Zobrazení dat

```
9  figure;
10 hold on;
11 plot(clBad(:, 1), clBad(:, 2), '^r');
12 plot(clGood(:, 1), clGood(:, 2), 'sb');
13 title('Repayment of Loans');
14 xlabel('Income');
15 ylabel('Wealth');
16 l1 = 'Not Repayed Loans';
17 l2 = 'Repayed Loans';
18 legend(l1, l2, 'Location', 'northwest');
```

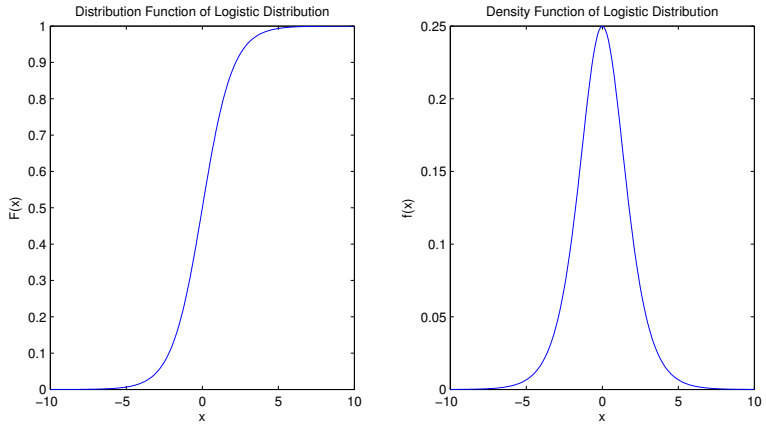
Budeme chtít vysvětlit závislé proměnné y_i pomocí nezávislých proměnných $x_{i,1}$ a $x_{i,2}$ pro $i = 1, \ldots, n$. Tento problém však nemůžeme řešit klasickou lineární regresí, protože proměnné y_i nabývají pouze hodnot 0 a 1. Představíme si tedy úlohu regrese s binární vysvětlovanou proměnnou. Označme Y_i náhodnou binární veličinu, u které pozorujeme hodnoty y_i. Pravděpodobnostní model předpokládáme ve tvaru

$$\mathsf{P}(Y_i = 1|x_{i,1}, x_{i,2}) = F\Big(\beta_1 + x_{i,1}\beta_2 + x_{i,2}\beta_3\Big) \qquad \text{pro } i = 1, \ldots, n \tag{11.1}$$

nebo ekvivalentně

$$\mathsf{P}(Y_i = 0|x_{i,1}, x_{i,2}) = 1 - F\Big(\beta_1 + x_{i,1}\beta_2 + x_{i,2}\beta_3\Big) \qquad \text{pro } i = 1, \ldots, n, \tag{11.2}$$

kde F je vhodná pravděpodobnostní distribuční funkce. Nejčastěji se za tuto funkci volí distribuční funkce logistického nebo normálního rozdělení. Na levé straně vzorce (11.1) máme pravděpodobnost, že náhodná veličina Y_i nabývá hodnoty 1. Na pravé straně pak máme nějakou distribuční funkci, která má jako parametr součet konstanty β_1 a vysvětlujících proměnných přenásobených koeficienty β_2 a β_3 podobně jako v lineární regresi. U minulých půjček je tato pravděpodobnost rovna 1 nebo 0, protože už víme, zda ke splacení došlo nebo ne. U budoucích půjček pak z tohoto modelu získáme odhad pravděpodobnosti



Obrázek 11.2: Distribuční funkce (vlevo) a hustota (vpravo) logistického rozdělení.

splacení v intervalu $(0, 1)$. Za funkci F použijeme distribuci logistického rozdělení, které si nejdříve představíme a pak si vykreslíme grafy jeho distribuční funkce a hustoty.

Logistické rozdělení

Logistické rozdělení je spojité pravděpodobnostní rozdělení s distribuční funkcí

$$F(x; \mu, \sigma) = \frac{\frac{x-\mu}{\sigma}}{1 + \frac{x-\mu}{\sigma}} = \frac{1}{1 + e^{-\frac{x-\mu}{\sigma}}}. \tag{11.3}$$

s parametry μ a σ. Logistické rozdělení připomíná normální rozdělení, ale má těžší chvosty.

Logit.m: Grafy distribuční funkce a hustoty logistického rozdělení

```
figure;
subplot(1, 2, 1);
xAxis = -10:0.1:10;
yAxis = cdf('Logistic', xAxis, 0, 1);
plot(xAxis, yAxis);
title('Distribution of Logistic Distribution');
xlabel('x');
ylabel('F(x)');
subplot(1, 2, 2);
xAxis = -10:0.1:10;
yAxis = pdf('Logistic', xAxis, 0, 1);
plot(xAxis, yAxis);
title('Density of Logistic Distribution');
xlabel('x');
ylabel('f(x)');
```

Vrátíme se zpět k našemu problému a do modelu (11.1) dosadíme logistickou distribuční funkci. Dostaneme tak model logistické regrese.

Logistická regrese

Logistická regrese se zabývá odhadem pravděpodobnosti výskytu nějakého jevu modelovaného jako binární náhodná veličina Y_i, u které pozorujeme hodnotu y_i, v závislosti na k vysvětlujících proměnných $\boldsymbol{x}_i = (x_{i,1}, \ldots, x_{i,k})$. Pravděpodobnost, že jev nastane, se modeluje jako

$$\mathsf{P}(Y_i = 1|\boldsymbol{x}_i) = \frac{1}{1 + e^{-\boldsymbol{x}_i\boldsymbol{\beta}}} \qquad \text{pro} \quad i = 1, \ldots, n, \tag{11.4}$$

kde $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_k)'$ jsou neznámé parametry. Podobný model lze použít, i pokud je vysvětlovaná proměnná kategoriální (může nabývat hodnot z konečné množiny) nebo ordinální (může nabývat hodnot z konečné uspořádané množiny).

Logit.m: Výpočet logistické funkce

```
36 function logistic = myLogistic(beta, X)
37   [n m] = size(X);
38   inside = [ones(n, 1) X] * beta;
39   logistic=1 ./ (1 + exp(-inside));
40 end
```

11.2 Odhady metodou maximální věrohodnosti

Nyní se pokusíme odhadnout koeficienty $\boldsymbol{\beta} = (\beta_1, \beta_2, \beta_3)'$. K jejich odhadu nepoužijeme metodu nejmenších čtverců jako u lineární regrese, ale metodu maximální věrohodnosti, kterou si teď představíme.

Metoda maximální věrohodnosti

Metoda maximální věrohodnosti (anglicky maximum likelihood estimation) je univerzální způsob odhadu parametrů. Pozorovaná data y_i uvažujeme jako nezávislé náhodné veličiny Y_i. Předpokládáme, že známe tvar jejich rozdělení, ale neznáme nějaké jeho parametry $\boldsymbol{\beta}$. Neznámé parametry $\boldsymbol{\beta}$ odhadneme tak, že budeme maximalizovat tzv. věrohodnostní funkci $l(\boldsymbol{\beta})$. V případě spojitých veličin Y_i s hustotami $f_i(y_i)$ a sdruženou hustotou $f(y_1, \ldots, y_n)$ má věrohodnostní funkce tvar

$$l(\boldsymbol{\beta}) = f(y_1, \ldots, y_n) = f_1(y_1) \cdots f_n(y_n). \tag{11.5}$$

V případě diskrétních veličin Y_n má věrohodnostní funkce tvar

$$l(\boldsymbol{\beta}) = \mathsf{P}(Y_1 = y_1, \ldots, Y_n = y_n) = \mathsf{P}(Y_1 = y_1) \cdots \mathsf{P}(Y_n = y_n). \tag{11.6}$$

Často se místo věrohodnostní funkce $l(\boldsymbol{\beta})$ používá tzv. logaritmická věrohodnostní funkce $L(\boldsymbol{\beta}) = \log l(\boldsymbol{\beta})$, která nabývá maxima ve stejném bodě jako $l(\boldsymbol{\beta})$ a v některých případech se s ní lépe počítá, protože logaritmus převádí součin na součet. Body $\hat{\boldsymbol{\beta}}^{MLE}$, ve kterých funkce $l(\boldsymbol{\beta})$ nebo $L(\boldsymbol{\beta})$ nabývá svého maxima, jsou odhady parametrů $\boldsymbol{\beta}$ metodou maximální věrohodnosti. Tyto odhady jsou po splnění některých dalších předpokladů konzistentní, eficientní a asymptoticky normální.

V našem případě pozorujeme data y_n, která uvažujeme jako nezávislé diskrétní náhodné veličiny Y_n s rozdělením (11.4). Neznámé parametry odhadneme $\boldsymbol{\beta}$ tak, že budeme maximalizovat logaritmickou

věrohodnostní funkci, kterou si můžeme vyjádřit jako

$$L(\boldsymbol{\beta}) = \sum_{i=1}^{n} \log \mathsf{P}(Y_i = y_i | x_{i,1}, x_{i,2}) = \sum_{\{i:y_i=0\}} \log \mathsf{P}(Y_i = 0 | x_{i,1}, x_{i,2}) + \sum_{\{i:y_i=1\}} \log \mathsf{P}(Y_i = 1 | x_{i,1}, x_{i,2}), \tag{11.7}$$

kde $\mathsf{P}(Y_i = 1|x_{i,1}, x_{i,2})$ máme definovanou ve vzorci (11.4) a $\mathsf{P}(Y_i = 0|x_{i,1}, x_{i,2}) = 1 - \mathsf{P}(Y_i = 1|x_{i,1}, x_{i,2})$. Maximalizaci funkce $L(\boldsymbol{\beta})$ si ještě převedeme na minimalizaci funkce $-L(\boldsymbol{\beta})$.

Logit.m: Záporná věrohodnostní funkce

```
42 function negLik = myNegLik(beta)
43   bad = sum(log(1 - myLogistic(beta, clBad)));
44   good = sum(log(myLogistic(beta, clGood)));
45   negLik = -bad - good;
46 end
```

11.3 Hledání optima nelineární účelové funkce

Máme již naprogramovanou funkci pro výpočet záporné logaritmické věrohodnostní funkce, zbývá tedy provést její minimalizaci. Funkce (11.7) zřejmě není lineární, musíme tedy přistoupit k tzv. nelineárnímu programování. Pro nelineární optimalizaci bez omezujících podmínek nabízí MATLAB funkci `fminunc`. Nelineární programování budeme řešit pomocí numerického algoritmu, musíme zvolit nějaké počáteční řešení, např. $(1, 1, 1)'$. Algoritmus se pak pokusí iterativně toto počáteční řešení zlepšovat. Nyní již můžeme najít minimum záporné logaritmické věrohodnostní funkce a tedy i odhady $\hat{\boldsymbol{\beta}} = (\hat{\beta}_1, \hat{\beta}_2, \hat{\beta}_3)'$.

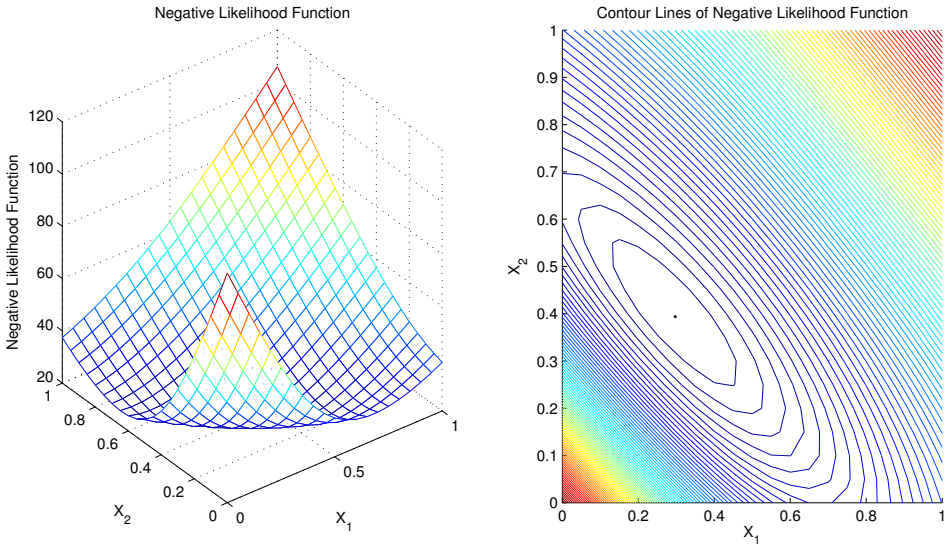
Funkce `fminunc`

Funkce `fminunc` hledá minimum funkce více proměnných bez omezujících podmínek. Zápis `x = fminunc(@fun, x0)` nám do proměnné `x` uloží optimální řešení. Argument `fun` představuje funkci, kterou chceme minimalizovat. Pokud chceme předat jako argument funkci (doposud jsme za argument měli pouze proměnné, vektory nebo matice), použijeme symbol `@` následovaný názvem funkce. Pokud jsme si dříve nadefinovali např. nějakou funkci `myFun`, její minimum najdeme pomocí `x = fminunc(@myFun, x0)`. Jedná se ovšem o numerický algoritmus, kterému ještě musíme dodat nějaký počáteční bod `x0`, ze kterého pak může hledat lokální minimum.

Logit.m: Minimalizace záporné věrohodnostní funkce

```
48 init = [1 1 1];
49 betaHat = fminunc(@myNegLik, init)
```

Pro představu, s jakou funkcí vlastně nelineární programování pracuje, si vykreslíme grafy záporné logaritmické věrohodnostní funkce. Grafy si vykreslíme dva a oba budou vyjadřovat závislost funkce $-L(\boldsymbol{\beta})$ na β_2 a β_3 při pevně zvoleném $\beta_1 = \hat{\beta}_1^{MLE}$. V prvním grafu vizualizujeme hodnotu účelové funkce $-L(\boldsymbol{\beta})$ v prostoru parametrů β_2 a β_3. tento graf bude trojrozměrný, kde na ose x bude koeficient β_2, na ose y koeficient β_3 a na ose z hodnota účelové funkce $-L(\boldsymbol{\beta})$. Druhý graf pak bude dvojrozměrný se stejnými osami x a y. Účelovou funkci $-L(\boldsymbol{\beta})$ zde pak zobrazíme pomocí jejich izokvant (vrstevnic), tedy křivek, jejichž všechny body mají stejnou hodnotu.



Obrázek 11.3: Záporná věrohodnostní funkce při pevném $\beta_1 = \hat{\beta}_1$.

Logit.m: Grafy věrohodnostní funkce

```
[b1Grid, b2Grid] = meshgrid(0:0.05:1, 0:0.05:1); % mrizka
[m n] = size(b1Grid);
for i = 1:m % iterativni vypocet hodnot na mrizce
  for j = 1:n
    curBeta = [betaHat(1) b1Grid(i, j) b2Grid(i, j)];
    val(i, j) = myNegLik(curBeta);
  end
end
figure;
subplot(1, 2, 1);
mesh(b1Grid, b2Grid, val); % vykresleni 3d plochy
title('Negative Likelihood Function');
xlabel('X_1');
ylabel('X_2');
zlabel('Negative Likelihood Function');
subplot(1, 2, 2);
hold on;
contour(b1Grid, b2Grid, val, 80); % graf vrstevnic
plot(betaHat(2), betaHat(3), '.k');
title('Contour Lines of Negative Likelihood Fun.');
xlabel('X_1');
ylabel('X_2');
```

11.4 Předpověď v modelu

Pokud k nám do banky dorazí nový klient s měsíčním příjmem a majetkem např. $\boldsymbol{x}^* = (5, 10)'$, můžeme jednoduše odhadnout pravděpodobnost, že půjčku splatí, jako

$$\mathsf{P}(Y^* = 1) = \frac{1}{1 + e^{-(\hat{\beta}_1 + \hat{\beta}_2 x_1^* + \hat{\beta}_3 x_2^*)}}. \tag{11.8}$$

Logit.m: Předpověď nového klienta

```
74 newCl = [5 10];
75 newClProb = myLogistic(betaHat, newCl)
```

Dále si vykreslíme graf závislosti splacení půjčky na tzv. skóre

$$S_i = \hat{\beta}_1 + \hat{\beta}_2 x_{i,1} + \hat{\beta}_3 x_{i,2} \tag{11.9}$$

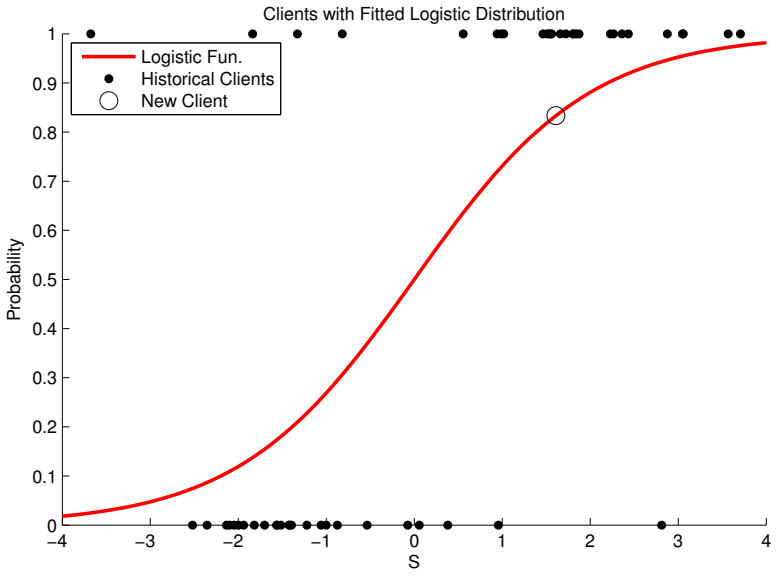
pro minulé klienty, proložíme ho odhadnutou logistickou křivkou a ještě zakreslíme pravděpodobnost splacení nového klienta. Nejdříve vykreslíme logistickou křivku, kterou interpretujeme jako pravděpodobnost splacení pro dané skóre. Dále vykreslíme binární proměnnou, zda minulí klienti půjčku splatili (hodnota 1) či nesplatili (hodnota 0), v závislosti na jejich skóre. Nakonec vykreslíme pravděpodobnost splacení našeho nového klienta (bod na logistické křivce) pro jeho skóre $S_i = \hat{\beta}_1 + 5\hat{\beta}_2 + 10\hat{\beta}_3$.

Logit.m: Graf splacení půjčky.

```
77 figure;
78 hold on;
79 s = -4:0.1:4;
80 plot(s, 1./ (1 + exp(-s)), 'r', 'LineWidth', 2);
81 s = betaHat(1) + clients(:, 2:3) * betaHat(2:3)';
82 plot(s, clients(:, 1), '.k', 'MarkerSize', 15);
83 s = betaHat(1) + newCl(1, 1:2) * betaHat(2:3)';
84 plot(s, newClProb, 'ok', 'MarkerSize', 10');
85 title('Clients with Fitted Logistic Distribution');
86 xlabel('S');
87 ylabel('Probability');
88 l1 = 'Logistic Fun.';
89 l2 = 'Historical Clients';
90 l3 = 'New Client';
91 legend(l1, l2, l3, 'Location', 'northwest');
```

11.5 Jednoduchá klasifikace podle pravděpodobností

Stejně jako v grafu 11.1 si opět zobrazíme výsledky půjček klientů v prostoru $x_{i,1}$ a $x_{i,2}$. Do grafu si ale ještě zakreslíme několik izokvant pravděpodobnosti splacení, tedy křivek, jejichž všechny body mají stejnou pravděpodobnost splacení. Izokvanty si necháme vykreslit dvě, což nám prostor rozdělí na 3 části. Izokvanty vybereme rovnoměrně, což znamená, že interval pravděpodobností $(0, 1)$ se rozdělí na intervaly $(0, \frac{1}{3}]$, $(\frac{1}{3}, \frac{2}{3}]$ a $(\frac{2}{3}, 1)$. Dostaneme tak tři třídy, které si postupně označíme jako ratingy C, B a A, do kterých můžeme klienty zařadit.

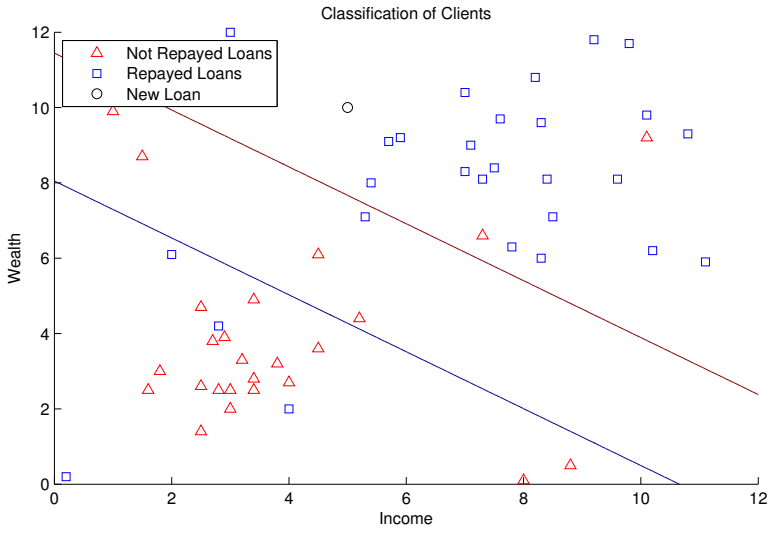


Obrázek 11.4: Závislost splacení půjčky na statistice S_i.

Logit.m: Graf klasifikace klientů.

```
figure;
hold on;
plot(clBad(:, 1), clBad(:, 2), '^r');
plot(clGood(:, 1), clGood(:, 2), 'sb');
plot(newCl(1, 1), newCl(:, 2), 'ok');
title('Classification of Clients');
xlabel('Income');
ylabel('Wealth');
l1 = 'Not Repayed Loans';
l2 = 'Repayed Loans';
l3 = 'New Loan';
legend(l1, l2, l3, 'Location', 'northwest');
[x1Grid, x2Grid] = meshgrid(0:0.1:12, 0:0.1:12);
[m n] = size(x1Grid);
for i = 1:m
  for j = 1:n
    curX = [x1Grid(i, j) x2Grid(i, j)];
    prob(i, j) = myLogistic(betaHat, curX);
  end
end
contour(x1Grid, x2Grid, prob, 2);
```

Podle roztřídění klientů do jednotlivých ratingů pak může banka zavést např. následující způsob rozhodování. Klientům s ratingem A banka půjčí za normálních podmínek, klientům s ratingem B půjčí s vyšším úrokem a klientům s ratingem C banka kvůli velké pravděpodobnosti nesplacení nepůjčí vůbec. Vidíme, že náš hypotetický klient x^* (označený kolečkem v grafu 11.5) by dostal rating A a banka by mu tak půjčila peníze za normálních podmínek.



Obrázek 11.5: Výsledky půjček klientů s izokvantami pravděpodobnosti splacení.

Doporučená literatura

- Cipra, T. 2014. *Finanční ekonometrie*. Ekopress. ISBN 978-80-86929-93-4. https://www.ekopress.cz/titdetail.php?tid=30238
- Greene, W. H. 2018. *Econometric Analysis*. Pearson. ISBN 978-0-13-446136-6. https://www.pearson.com/us/higher-education/program/Greene-Econometric-Analysis-8th-Edition/PGM334862.html
- James, G., Witten, D., Hastie, T., Tibshirani, R. 2013. *An Introduction to Statistical Learning: With Applications in R*. Springer. ISBN 978-1-4614-7137-0. https://doi.org/10.1007/978-1-4614-7138-7
- Ledolter, J. 2013. *Data Mining and Business Analytics with R*. Wiley. ISBN 978-1-118-44714-7. https://doi.org/10.1002/9781118596289
- Wooldridge, J. M. 2019. *Introductory Econometrics: A Modern Approach*. Cengage Learning. ISBN 978-1-337-55886-0. https://www.cengage.co.uk/books/9781337558860/

Kapitola 12

Support Vector Machines (SVM)

Představíme si metodu Support Vector Machines, která se používá pro oddělení množin. Ukážeme si ještě její modifikaci, která vynechá několik odlehlých pozorování.

Klíčová slova: *Support Vector Machines, separace množin, odlehlá pozorování, lineární programování.*

12.1 Separace množin

Geometricky se podstata metody SVM dá vystihnout tak, že jsou dány dvě množiny bodů v $\mathbb{R}^n$ a cílem je najít nadrovinu, která je „dobře“ odděluje (tzv. *separátor*), anebo konstatovat, že je oddělit nelze. V praxi se metoda často užívá v data miningu, a to v situacích, které jsou analogické kapitole 11 — připomeňme, že tam jsme konstruovali logistický scoringový model pro klienty bank. Tuto aplikaci použijeme jako základní příklad i zde. Sestrojíme jednoduchý SVM-klasifikátor klientů; lze říci, že SVM-klasifikace je alternativní metodou ke scoringu založeném na statistických metodách.

Mějme dány dvě množiny klientů bank, říkejme jim *dobří* a *špatní* (např. na základě minulé zkušenosti, zdali řádně spláceli úvěry). Klient je charakterizován dvojicí údajů (x, y); řekněme, že jde výši majektu (x) a pravidelnou měsíční mzdu (y). (V praktických aplikacích bývají takových údajů, tzv. *atributů*, desítky či stovky; zde volíme jen dva, aby bylo snadné kreslit obrázky.) Mějme n dobrých a m špatných klientů; jejich data uspořádejme do vektorů

$$\begin{aligned} \text{dobří klienti:} \quad & \boldsymbol{x} = (x_1, \ldots, x_n)', \quad \boldsymbol{y} = (y_1, \ldots, y_n)', \\ \text{špatní klienti:} \quad & \widetilde{\boldsymbol{x}} = (\widetilde{x}_1, \ldots, \widetilde{x}_m)', \quad \widetilde{\boldsymbol{y}} = (\widetilde{y}_1, \ldots, \widetilde{y}_m)'. \end{aligned}$$

Najít separátor dobrých a špatných klientů obnáší najít koeficienty θ_1, θ_2 přímky

$$y = \theta_1 + \theta_2 x$$

takové, že

$$y_i \geq \theta_1 + \theta_2 x_i \ \ (\forall i = 1, \ldots, n); \quad \widetilde{y}_j \leq \theta_1 + \theta_2 \widetilde{x}_j \ \ (\forall j = 1, \ldots, m). \tag{12.1}$$

Znalost separátoru pak umožňuje klasifikovat nové klienty: přijde-li nový klient s majetkem x^0 a mzdou y^0, SVM-klasifikátor jej prohlásí za *dobrého*, jestliže $y^0 > \theta_1 + \theta_2 x^0$; a prohlásí jej za *špatného*, jestliže

$y^0 < \theta_1 + \theta_2 x^0$. (Případ $y^0 = \theta_1 + \theta_2 x^0$, který teoreticky rovněž může nastat, můžeme například ztotožnit s odpovědí „nevím“).

12.2 Formulace pomocí lineárního programování

Podmínky (12.1) jsou lineární v θ_1, θ_2, a tak se θ_1, θ_2 snadno najdou pomocí lineárního programování. Bude výhodné zvolit obecnější tvar: předpokládejme, že jsou namísto vektorů $\boldsymbol{x}, \widetilde{\boldsymbol{x}}$ dány matice $\boldsymbol{X} \in \mathbb{R}^{n\times p}$ a $\widetilde{\boldsymbol{X}} \in \mathbb{R}^{m\times p}$ a namísto (12.1) pišme

$$\boldsymbol{y} \geq \boldsymbol{X}\boldsymbol{\theta}, \quad \widetilde{\boldsymbol{y}} \leq \widetilde{\boldsymbol{X}}\boldsymbol{\theta}, \tag{12.2}$$

kde $\boldsymbol{\theta} \in \mathbb{R}^p$ je vektor parametrů. Zvolíme-li nyní

$$p = 2, \quad \boldsymbol{X} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}, \quad \widetilde{\boldsymbol{X}} = \begin{pmatrix} 1 & \widetilde{x}_1 \\ 1 & \widetilde{x}_2 \\ \vdots & \vdots \\ 1 & \widetilde{x}_m \end{pmatrix}, \tag{12.3}$$

obdržíme přesně (12.1). Zvolíme-li ovšem například

$$p = 3, \quad \boldsymbol{X} = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{pmatrix}, \quad \widetilde{\boldsymbol{X}} = \begin{pmatrix} 1 & \widetilde{x}_1 & \widetilde{x}_1^2 \\ 1 & \widetilde{x}_2 & \widetilde{x}_2^2 \\ \vdots & \vdots & \vdots \\ 1 & \widetilde{x}_m & \widetilde{x}_m^2 \end{pmatrix}, \tag{12.4}$$

vyřešením systému (12.2) získáme vektor $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)'$ takový, že dobré a špatné klienty odděluje parabola

$$y = \theta_1 + \theta_2 x + \theta_3 x^2;$$

SVM-klasifikátor pak nového klienta (x^0, y^0) klasifikuje podle nerovnosti $y^0 \lessgtr \theta_1 + \theta_2 x^0 + \theta_3 (x^0)^2$.

Obecně může být separátorů (tj. vektorů $\boldsymbol{\theta}$) splňujících (12.2) mnoho, nebo také nemusí existovat žádný.

12.3 Případ, kdy je separátorů mnoho

V takovém případě bývá cílem najít separátor nikoliv coby přímku, ale coby *pás co největší šířky* oddělující body (x_i, y_i) od bodů $(\widetilde{x}_j, \widetilde{y}_j)$. Šířku pásu označme δ. Vyjdeme-li z (12.1), cílem je pak najít co největší δ takové, že

$$y_i \geq \delta + \theta_1 + \theta_2 x_i \ \ (\forall i = 1, \ldots, n); \quad \widetilde{y}_j \leq -\delta + \theta_1 + \theta_2 \widetilde{x}_j \ \ (\forall j = 1, \ldots, m).$$

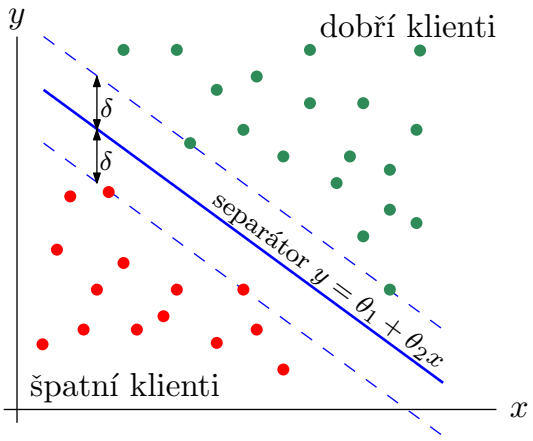
Užijeme-li maticovou notaci z (12.2), cílem je najít co největší δ takové, že

$$\boldsymbol{y} \geq \delta \boldsymbol{e} + \boldsymbol{X}\boldsymbol{\theta}, \quad \widetilde{\boldsymbol{y}} \leq -\delta \boldsymbol{e} + \widetilde{\boldsymbol{X}}\boldsymbol{\theta}, \tag{12.5}$$

kde $\boldsymbol{e}$ je vektor samých jedniček příslušného rozměru. Situaci ilustruje následující obrázek, kde dobří klienti $(x_i, y_i)_{i=1,\ldots,n}$ jsou zobrazeni jako zelené body v rovině a špatní klienti $(\widetilde{x}_j, \widetilde{y}_j)_{j=1,\ldots,m}$ jsou zobrazeni jako červené body v rovině.

Vzhledem k (12.5) můžeme okamžitě psát lineární program vhodný pro solver `linprog`:

$$\min_{\binom{\delta}{\boldsymbol{\theta}}} \begin{pmatrix} -1 & \boldsymbol{0}_{1\times p} \end{pmatrix} \begin{pmatrix} \delta \\ \boldsymbol{\theta} \end{pmatrix} \quad \text{s.t.} \quad \begin{pmatrix} \boldsymbol{e}_{n\times 1} & \boldsymbol{X} \\ \boldsymbol{e}_{m\times 1} & -\widetilde{\boldsymbol{X}} \end{pmatrix} \begin{pmatrix} \delta \\ \boldsymbol{\theta} \end{pmatrix} \leq \begin{pmatrix} \boldsymbol{y} \\ -\widetilde{\boldsymbol{y}} \end{pmatrix}. \tag{12.6}$$



Obrázek 12.1: Oddělení klientů.

Nyní můžeme volat

```
xi = linprog([-1; zeros(p,1)], [ones(n+m,1), [X; -Xtilde]], [y; -ytilde]),
```

kde data špatných klientů $(\widetilde{\boldsymbol{X}}, \widetilde{\boldsymbol{y}})$ jsme označili jako `(Xtilde, ytilde)`. Výsledkem je optimální vektor `xi`; jeho první složka `xi(1)` je optimální šíře oddělujícího pásu δ a `theta = xi(2:p+1)` je nalezený separátor $\boldsymbol{\theta}$, při němž je šíře oddělujícího pásu největší. Solver pro SVM je velice jednoduchý a nepotřebuje další komentář:

SVM.m: Solver SVM

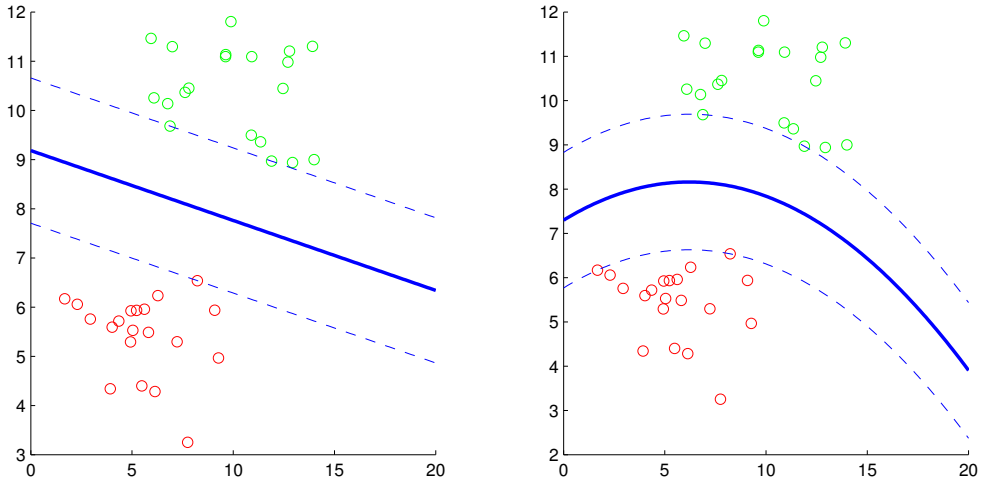
```
function [delta, theta] = SolveSVM(X, y, Xtilde, ytilde)
  [n, p] = size(X);
  [m, p] = size(Xtilde);
  xi = linprog([-1; zeros(p, 1)], [ones(n + m, 1), ...
    [X; -Xtilde]], [y; -ytilde]);
  delta = xi(1);
  theta = xi(2:(p + 1));
end
```

Můžeme zkusit testovat solver `SolveSVM` na simulovaných datech a nakreslit si obrázek. Řekněme, že náhodně vygenerujeme $n = 20$ dobrých klientů (x_i, y_i) a $m = 20$ špatných klientů $(\widetilde{x}_j, \widetilde{y}_j)$ například pomocí

$$x_i \sim N(10, 3^2), \quad y_i \sim N(10, 1), \quad \widetilde{x}_i \sim N(5, 2^2), \quad \widetilde{y}_i \sim N(5, 1). \tag{12.7}$$

Napišme skript, který s využitím již hotové funkce `SolveSVM` najde a vykreslí separátor ve tvaru přímky a paraboly (dobří klienti jako zelené body, špatní jako červené body):

Nejprve vygenerujeme náhodná data podle (12.7):



Obrázek 12.2: Separátor ve tvaru přímky (vlevo) a paraboly (vpravo).

SVM.m: Generování dat

```
12 n = 20;
13 m = 20;
14 x = random('norm', 10, 3, [n, 1]);
15 y = random('norm', 10, 1, [n, 1]);
16 xtilde = random('norm', 5, 2, [m, 1]);
17 ytilde = random('norm', 5, 1, [m, 1]);
```

Vytvoříme obrázek; do levé části budeme kreslit lineární separátor. Začneme tím, že vykreslíme vygenerované datové body.

SVM.m: Vykreslení dat.

```
19 figure;
20 subplot(1, 2, 1);
21 hold on;
22 plot(x, y, 'og', xtilde, ytilde, 'or');
23 subplot(1, 2, 2);
24 hold on;
25 plot(x, y, 'og', xtilde, ytilde, 'or');
```

Vytvoříme matice $\boldsymbol{X}$ a $\widetilde{\boldsymbol{X}} =$ `Xtilde` podle (12.3) a voláme `SolveSVM`. A konečně zbývá vykreslit trojici přímek — silněji vlastní separátor $y = \theta_1 + \theta_2 x$ a čárkovaně kraje oddělujícího pásu $y = \theta_1 + \theta_2 x \pm \delta$.

SVM.m: Vykreslení lineárního separátoru.

```
X = [ones(n, 1), x];
Xtilde = [ones(m, 1), xtilde];
[delta, theta] = SolveSVM(X, y, Xtilde, ytilde);
t = [0:0.1:20];
plot(t, theta(1) + theta(2) * t, 'b', 'LineWidth', 2);
plot(t, theta(1) + theta(2) * t - delta, 'b--');
plot(t, theta(1) + theta(2) * t + delta, 'b--');
```

Nyní celý postup zopakujeme (už bez komentářů); do pravého obrázku vykreslíme kvadratický separátor. Užijeme proto matice $\boldsymbol{X}$, $\widetilde{\boldsymbol{X}}$ ve tvaru (12.4).

SVM.m: Vykreslení kvadratického separátoru.

```
X = [ones(n, 1), x, x.^2];
Xtilde = [ones(m, 1), xtilde, xtilde.^2];
[delta, theta] = SolveSVMx(X, y, Xtilde, ytilde);
t = [0:0.1:20];
plot(t, theta(1) + theta(2) * t + theta(3) * ...
  t.^2, 'b', 'LineWidth', 2);
plot(t, theta(1) + theta(2) * t + theta(3) * ...
  t.^2 - delta, 'b--');
plot(t, theta(1) + theta(2) * t + theta(3) * ...
  t.^2 + delta, 'b--');
```

12.4 Případ, kdy separátor neexistuje

Samozřejmě se může stát, že body oddělit nejde; dobří a špatní klienti mohou být „promícháni“. To ovšem snadno poznáme; optimální hodnota δ lineárního programu (12.6) je pak záporná. Potom dolní a horní mez oddělujícího pásu jen vymění své role a výsledkem optimalizace bude *co nejužší pás obsahující oba typy klientů*; mimo pás už je separace jednoznačná. Klasifikátor pak typicky nově příchozího klienta spadajícího do oddělujícího pásu označí odpovědí „nevím“. Situace pak může vypadat jako na obrázku 12.3.

12.5 Očištění o odlehlé body

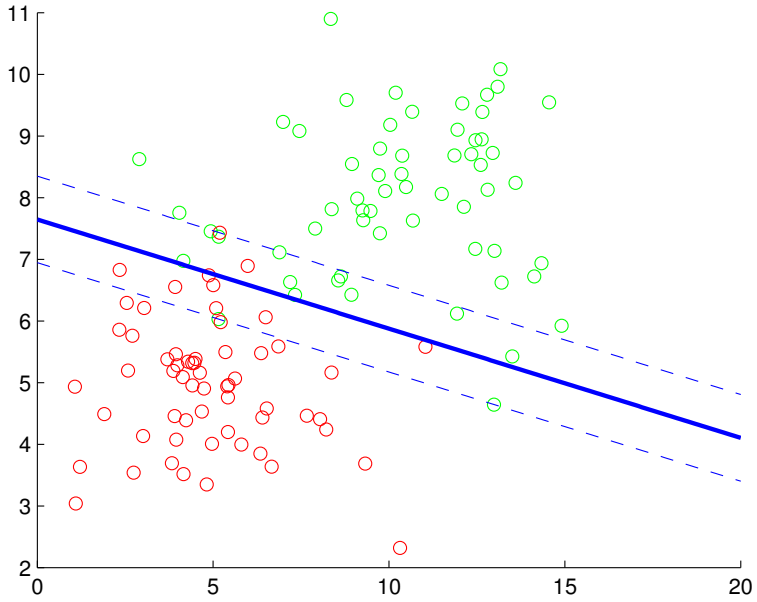
V praxi se často připouští, že některá data mohou být v jistém smyslu netypická, nereprezentativní, extrémní, či dokonce chybná; říkává se jim *odlehlé body* či *outliers*. Takové body mohou způsobit, že model má slabou klasifikační schopnost; optimální hodnota δ lineárního programu (12.6) je hodně záporná a pás odpovědí „nevím“ je hodně široký. Bývá proto povoleno z datového souboru několik bodů vynechat; cílem je vynechat právě ony odlehlé body (pokud v datech existují), které způsobují špatnou oddělitelnost.

Jak takové body najít? Přímo se nabízí intuitivně zřejmá úvaha. S ohledem na (12.6) položme

$$N = n + m$$

a

$$\boldsymbol{U} = \begin{pmatrix} \boldsymbol{X} \\ -\widetilde{\boldsymbol{X}} \end{pmatrix}, \quad \boldsymbol{v} = \begin{pmatrix} \boldsymbol{y} \\ -\widetilde{\boldsymbol{y}} \end{pmatrix}. \tag{12.8}$$



Obrázek 12.3: Oddělení klientů, kde separující pás obsahuje dobré i špatné klienty, tzb. pás „nevím“.

Nechť matice $\boldsymbol{U}^k$ vznikne z matice $\boldsymbol{U}$ vypuštěním k-tého řádku, kde $k \in \{1, \ldots, N\}$. Analogicky, vektor $\boldsymbol{v}^k$ nechť vznikne z vektoru $\boldsymbol{v}$ vypuštěním k-té složky. Vyřešíme nyní lineární programy $(LP_1), \ldots, (LP_N)$ tvaru

$$(LP_k) \quad \min_{\binom{\delta}{\boldsymbol{\theta}}} \begin{pmatrix} -1 & \mathbf{0}_{1\times p} \end{pmatrix} \begin{pmatrix} \delta \\ \boldsymbol{\theta} \end{pmatrix} \text{ s.t. } \begin{pmatrix} \boldsymbol{e}_{(N-1)\times 1} & \boldsymbol{U}^k \end{pmatrix} \begin{pmatrix} \delta \\ \boldsymbol{\theta} \end{pmatrix} \leq \boldsymbol{v}^k. \tag{12.9}$$

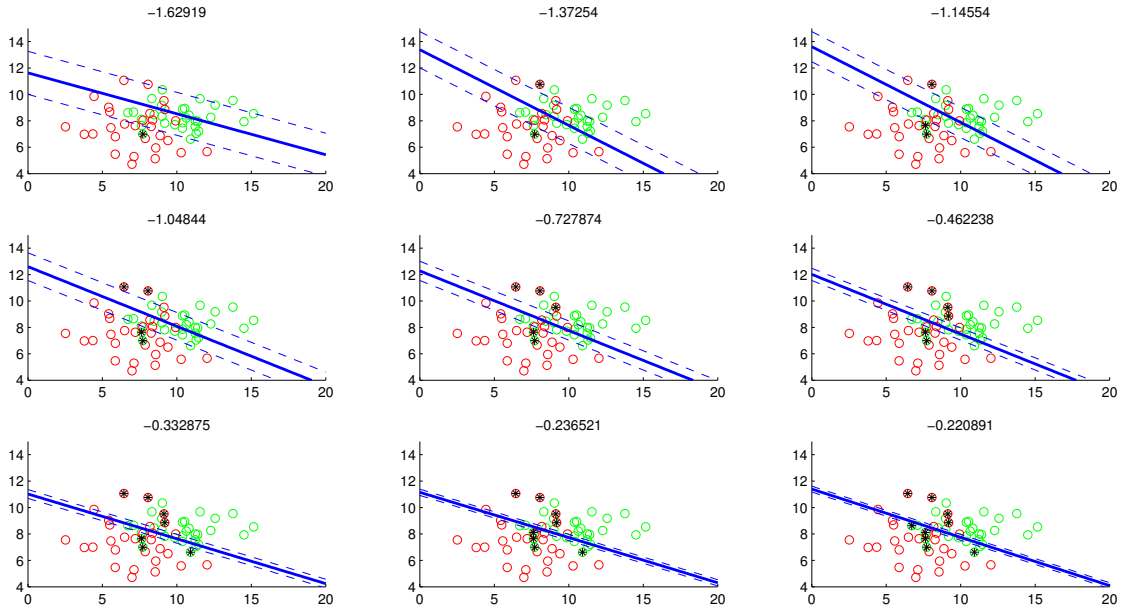
Jsou-li

$$\delta_1^*, \ldots, \delta_N^* \tag{12.10}$$

jejich optimální šíře dělící pásů, vybereme ten index i_0, pro nějž je $\delta_{i_0}^*$ největší. Tím jsme vlastně zjistili, že vypuštěním i_0-tého bodu dosáhneme nejlepší separace. Máme-li povoleno vypouštět více bodů, řekněme r, pak celou proceduru r-krát opakujeme.

Řekněme, že máme povoleno vypustit $r = 9$ bodů (číslo 9 volíme jen proto, abychom mohli vypouštění bod po bodu kreslit do obrázku 3×3; samozřejmě by bylo také možné skript animovat). Nakreslíme následující obrázek:

V prvním obrázku je výsledek separace po vyřazení prvního bodu (označen černou hvězdičkou); číslo nad obrázkem je šíře pásu δ. Ve druhém obrázku jsou vypuštěny dva body (označeny černě), a tak dále; v posledním obrázku je vypuštěno devět bodů a šíře oddělujícího pásu (byť stále záporná) je velmi blízko nuly. Body jsme generovali (jako příklad) pro $n = 30$ a $m = 30$ s $x_i \sim N(10, 3^2)$, $y_i \sim N(8, 1)$, $\widetilde{x}_j \sim N(7, 2^2)$ a $\widetilde{y}_j \sim N(7, 2^2)$.



Obrázek 12.4: Postupné vypouštění bodů.

OutlierSVM.m: Generování dat.

```
12       n = 30;
13       m = 30;
14       N = n + m;
15       x = random('norm', 10, 3, [n 1]);
16       y = random('norm',  8, 1, [n 1]);
17  xtilde = random('norm',  7, 2, [m 1]);
18  ytilde = random('norm',  7, 2, [m 1]);
```

OutlierSVM.m: Příprava matic.

```
20       X = [ones(n, 1), x];
21  Xtilde = [ones(n, 1), xtilde];
22       U = [X; -Xtilde];
23       v = [y; -ytilde];
24  % ulozime si souradnice bodu pro pozdejsi vizualizaci
25      xp = [x; xtilde];
26      yp = [y; ytilde];
27       r = 9; % pocet povolenych bodu k vypusteni
```

OutlierSVM.m: Příprava vizualizace.

```
29 figure;
30 for l = 1:r % devet obrazku s vykreslenim dat
31   subplot(3, 3, l);
32   hold on;
33   plot(x, y, 'og', xtilde, ytilde, 'or');
34   axis([0 20 4 15]) % na kazdem obrazku stejne meritko
35 end
```

OutlierSVM.m: Hlavní cyklus.

```
37 for i = 1:r
38    delta = []; % zde si ukladame hodnoty delta
39   Thetas = []; % zde si ukladame jim odpovidajici...
40     % vektory theta
```

OutlierSVM.m: Vypuštění jednoho bodu.

```
42   for k = 1:N
43     Uk = U([1:(k - 1), (k + 1):N], :); % vypoustime ...
44       % k-ty bod
45     vk = v([1:(k - 1), (k + 1):N]); % vypoustime k-ty bod
46     xi = linprog([-1; zeros(2, 1)], ...
47       [ones(N-1, 1), Uk], vk);
48     delta(k) = xi(1); % ulozime optimalni delta
49     Thetas(:, k) = xi(2:3); % ulozime i nalezeny separator
50   end
51   [deltaopt, j] = max(delta); % j = index, kde je delta...
52     % nejvyssi, tento bod budeme vypoustet z datasetu
53   theta = Thetas(:, j); % theta = separator ...
54     % odpovidajici delta(j)
```

OutlierSVM.m: Vizualizace separátoru oddělujícího pásu.

```
56   subplot(3, 3, i);
57   hold on;
58   t = [0:0.1:20];
59   plot(t, theta(1) + theta(2) * t, 'b', 'LineWidth', 2);
60   plot(t, theta(1) + theta(2) * t - deltaopt, 'b--');
61   plot(t, theta(1) + theta(2) * t + deltaopt, 'b--');
62   title(deltaopt); % hodnotu delta vypiseme nad obrazek
```

OutlierSVM.m: Vizualizace označení vypuštěného bodu.

```
64    for l = i:r % cernou hvezdickou vyznacime vypusteny bod
65      subplot(3, 3, l);
66      plot(xp(j), yp(j), '*k', 'MarkerSize', 6); % v xp,yp
67        % mame ulozeny souradnice bodu
68    end
```

OutlierSVM.m: Vypuštěný bod vymažeme z datasetu.

```
70     U = U([1:(j - 1), (j + 1):N], :);
71     v = v([1:(j - 1), (j + 1):N]);
72    xp = xp([1:(j - 1), (j + 1):N], :);
73    yp = yp([1:(j - 1), (j + 1):N]);
74    N = N - 1; % velikost datasetu se zmensila o jednicku
75  end % konec hlavniho for-cyklu
```

Poznámka

Úlohu vypustit r bodů tak, aby šířka pásu δ byla co největší, lze také psát jako *smíšený* lineární program

$$\begin{aligned} \max_{\substack{\delta\in\mathbb{R}\\ \boldsymbol{\theta}\in\mathbb{R}^2\\ \boldsymbol{w}\in\{0,1\}^n\\ \boldsymbol{z}\in\{0,1\}^m}} \quad & \delta \\ \text{s. t.} \quad & \underbrace{y_i + Mw_i \geq \delta + \theta_1 + \theta_2 x_i \quad \forall i = 1,\dots n}_{(\star)}, \\ & \underbrace{\widetilde{y}_j - Mz_j \leq -\delta + \theta_1 + \theta_2 \widetilde{x}_j \quad \forall j = 1,\dots n}_{(\dagger)}, \\ & \underbrace{\boldsymbol{e}'\boldsymbol{w} + \boldsymbol{e}'\boldsymbol{z} = r}_{(\ddagger)}, \end{aligned} \tag{12.11}$$

kde M je pevně zvolené velké číslo. Nula-jedničkové proměnné $\boldsymbol{w}$, $\boldsymbol{z}$ jsou indikátory bodů, které jsou vypuštěny. Omezení $(\ddagger)$ říká, že jich musí být právě r. Je-li $w_i = 1$, pak i-té omezení v $(\star)$ je jistě splněno (levá strana je jistě větší než pravá, je-li M dostatečně velké, bez ohledu na hodnoty x_i, y_i), a tedy model se chová, jako by i-tý bod mezi dobrými klienty nebyl v datech přítomen. Analogický je význam aditivního členu $-Mz_j$ v $(\dagger)$. Zřejmě je (12.11) smíšený lineární program, některé proměnné jsou totiž diskrétní. Nelze jej tudíž řešit pomocí solveru `linprog`. Jak lze takové problémy řešit, jsme ukázali v kapitole 10.

Doporučená literatura

- James, G., Witten, D., Hastie, T., Tibshirani, R. 2013. *An Introduction to Statistical Learning: With Applications in R*. Springer. ISBN 978-1-4614-7137-0. https://doi.org/10.1007/978-1-4614-7138-7
- Ledolter, J. 2013. *Data Mining and Business Analytics with R*. Wiley. ISBN 978-1-118-44714-7. https://doi.org/10.1002/9781118596289
- Manning, C. D., Raghavan, P., Schütze, H. 2008. *Introduction to Information Retrieval*. Cambridge University Press. ISBN 978-0-521-86571-5. https://doi.org/10.1017/cbo9780511809071
- Vanderbei, R. J. 2020. *Linear Programming: Foundations and Extensions*. Springer. ISBN 978-3-030-39414-1. https://doi.org/10.1007/978-3-030-39415-8

Kapitola 13

Model dravec-kořist

V této kapitole se budeme zabývat rovnicemi dravce-kořisti známými také jako Lotkovy-Volterrovy rovnice, které modelují velikost populace dravců a populace kořisti.

Klíčová slova: *model dravec-kořist, rovnice Lotka–Volterra, Goodwinův model.*

13.1 Motivace a popis modelu

Nejdříve si představíme ekonomickou úlohu, která na tento model povede. Budeme zkoumat souvislost mezi mzdami a zaměstnaností. Při vysoké zaměstnanosti roste vyjednávací pozice pracovníků, tím se zvyšují jejich mzdy a snižuje se zisk společnosti. Jak zisk klesá, společnost najímá méně pracovníků a vzroste nezaměstnanost, což vede ke zvýšení zisku. S vyšším ziskem pak společnost najímá více pracovníků a opět roste zaměstnanost. Pozorujeme tedy cyklické chování mezi výší mezd pracovníků a zaměstnaností. Toto dynamické chování lze zachytit modelem dravec-kořist.

Model dravec-kořist

Model dravec-kořist je dvojice diferenciálních rovnic představujících vývoj dvou populací. Základní ideu modelu dravec-kořist můžeme ilustrovat na populaci králíků (kořisti) a lišek (dravců). Budeme uvažovat následující předpoklady:

- Populace králíků se zvyšuje proporčně ke svojí velikosti.
- Králíci neumírají přirozenou cestou, umírají pouze, když je lišky zabijí. Jejich populace se snižuje proporčně ke své velikosti a velikosti populace lišek.
- Populace lišek se zvyšuje proporčně ke svojí velikosti a velikosti populace králíků.
- Lišky umírají přirozenou smrtí. Jejich populace se snižuje proporčně ke své velikosti.

Tyto předpoklady ovlivňující vývoj populací králíků a lišek lze vyjádřit rovnicemi

$$\begin{aligned} \frac{dx(t)}{dt} &= \alpha x(t) - \beta x(t)y(t), \\ \frac{dy(t)}{dt} &= \delta x(t)y(t) - \gamma y(t), \end{aligned} \tag{13.1}$$

kde

- $x(t)$ je velikost populace králíků v čase t,
- $y(t)$ je velikost populace lišek v čase t,
- $\frac{dx(t)}{dt}$ je tempo růstu populace králíků,
- $\frac{dy(t)}{dt}$ je tempo růstu populace lišek,
- α je parametr přirozeného přírůstku králíků,
- β je parametr úmrtí králíků způsobeného liškami,
- γ je parametr přirozeného úmrtí lišek,
- δ je parametr přírůstků lišek lovením králíků,
- $x_0 = x(0)$ je daná počáteční velikost populace králíků a
- $y_0 = y(0)$ je daná počáteční velikost populace lišek.

Velikosti populací $x(t)$ a $y(t)$ jsou funkce času. Koeficienty α, β, γ a δ a počáteční velikosti populace x_0 a y_0 jsou parametry modelu. Je-li v lese velké množství králíků, vzroste i počet lišek, které králíky začnou lovit. Tím se zmenší populace králíků, liškám dojde zdroj potravy a jejich populace se začne snižovat. Při malém počtu lišek pak začne růst populace králíků a dochází opět k cyklickému chování velikostí obou populací.

Jak jsme již zmínili v úvodu, dynamický model dravce-kořisti můžeme využít i v ekonomii. Úloha popisující dynamický vztah výši mezd a zaměstnaností vycházející z modelu dravce-kořisti se nazývá Goodwinův model. Zde máme výši mezd v roli lišek (dravců) a zaměstnanost v roli králíků (kořisti).

13.2 Diskretizace diferenciálního modelu

Nyní se podíváme, jak tento cyklický vývoj velikostí populace králíků a lišek bude vypadat. Nejdříve si zvolíme nějaké parametry α, β, γ a δ charakterizující přírůstky a úbytky a dále stanovíme počáteční velikosti populace králíků x_0 a velikost populace lišek y_0.

PredatorPrey.m: Parametry a počáteční stavy

```
3  alpha = 1;
4   beta = 1;
5  gamma = 1;
6  delta = 1;
7   xOld = 0.9;
8   yOld = 0.9;
```

Budeme používat numerické výpočty, protože rovnice (13.1) nemají řešení jako elementární funkce. Spojitý model nejdříve převedeme na model diskrétní. Vývoj velikostí populace tedy budeme sledovat v diskrétních časech od 0 do 30 s krokem 0.001. Budeme tedy mít 30 000 časových bodů.

PredatorPrey.m: Časové body

```
10     tMin = 0;
11     tMax = 30;
12  tChange = 0.001;
```

Protože uvažujeme diskrétní čas, musíme zdiskretizovat model (13.1). Tedy z diferenciálních rovnic uděláme rovnice diferenční

$$\begin{aligned} \frac{x_t - x_{t-\Delta}}{\Delta} &= \alpha x_{t-\Delta} - \beta x_{t-\Delta} y_{t-\Delta}, \\ \frac{y_t - y_{t-\Delta}}{\Delta} &= \delta x_{t-\Delta} y_{t-\Delta} - \gamma y_{t-\Delta}, \end{aligned} \tag{13.2}$$

kde x_t je velikost populace králíků v čase t, y_t je velikost populace lišek v čase t a Δ je délka jednoho časového skoku, v našem případě tedy 0.001. Pro každý časový bod $t = 0.001, 0.002, \ldots, 30$ můžeme rekurentně spočítat velikost nové populace x_t a y_t pomocí velikostí těchto populací $x_{t-\Delta}$ a $y_{t-\Delta}$ z předchozího časového bodu. Velikosti nových populací jsou dány

$$\begin{aligned} x_t &= x_{t-\Delta} + \Delta\Big(\alpha x_{t-\Delta} - \beta x_{t-\Delta} y_{t-\Delta}\Big), \\ y_t &= y_{t-\Delta} + \Delta\Big(\delta x_{t-\Delta} y_{t-\Delta} - \gamma y_{t-\Delta}\Big). \end{aligned} \tag{13.3}$$

Vykreslíme dva grafy. V prvním si ukážeme změnu velikosti populace králíků a lišek. Na horizontální osu budeme vynášet počty králíků a na vertikální osu počty lišek. V dalším grafu si pak zobrazíme vývoj populace králíků a lišek v čase. Na horizontální osu budeme vynášet čas a na vertikální osu počty králíků i lišek. Nejdříve si označíme osy grafu pomocí následujícího skriptu.

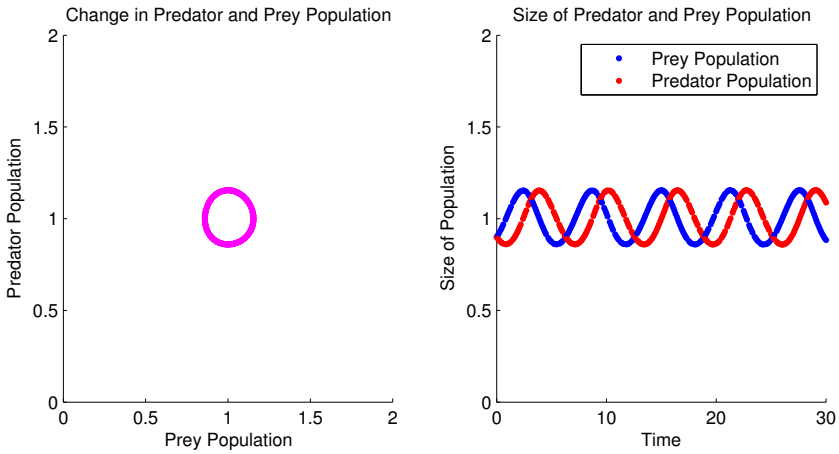
PredatorPrey.m: Příprava grafu

```
14 figure;
15 subplot(1, 2, 1);
16 hold on;
17 axis([0 2 0 2]);
18 title('Change in Predator and Prey Population');
19 xlabel('Prey Population');
20 ylabel('Predator Population');
21 subplot(1, 2, 2);
22 hold on;
23 axis([tMin tMax 0 2]);
24 title('Size of Predator and Prey Population');
25 xlabel('Time');
26 ylabel('Size of Population');
```

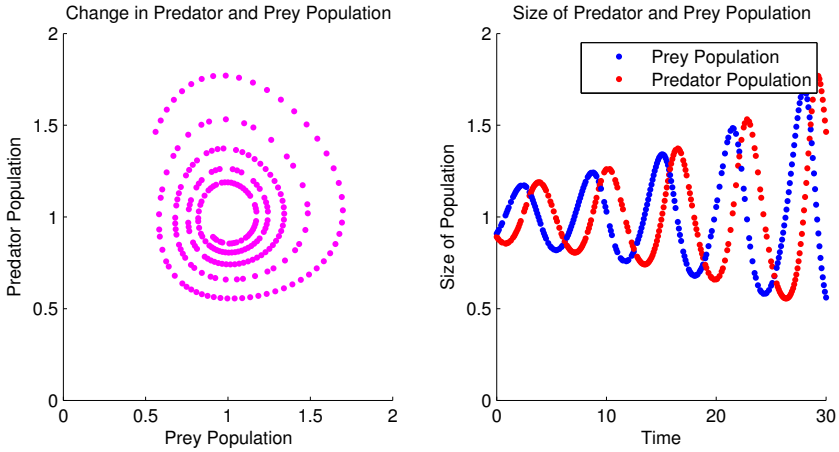
Teď již můžeme přistoupit k samotnému výpočtu vývoje populací. Budeme postupovat iteračně. Počáteční velikosti populací v čase 0 máme zadané a v každém dalším časovém okamžiku t vypočteme velikosti populace z předchozího času $t-\Delta$ pomocí rovnic (13.3). Oba grafy si budeme chtít vykreslit jako animaci, v každé iteraci tedy zavoláme příkaz `plot`. Abychom ale MATLAB nezahlcovali (počítáme 30 000 iterací), budeme vykreslovat pouze každý stý bod. Celkem tedy vykreslíme pouze 300 bodů. Toho docílíme pomocí modula dělení (zbytek po celočíselném dělení, např. $7 \bmod 3 = 1$), kdy budeme kontrolovat, zda aktuální čas vynásobený 10 je celé číslo. Podmínkou projdou tedy pouze časy $t = 0, 0.1, 0.2, \ldots, 30$. Pro efekt animace si na konci každé iterace program na zlomek vteřiny zastavíme. V obou grafech můžeme pozorovat cyklické chování vývoje populace králíků a lišek.

PredatorPrey.m: Průběh velikosti populací

```
28 for t = tMin:tChange:tMax
29   xNew = xOld + tChange * ...
30     (alpha * xOld - beta * xOld * yOld);
31   yNew = yOld + tChange * ...
32     (delta * xOld * yOld - gamma * yOld);
33   xOld = xNew;
34   yOld = yNew;
35   if mod(10 * t, 1) == 0
36     subplot(1, 2, 1);
37     plot(xNew, yNew, '.m', 'MarkerSize', 10);
38     subplot(1, 2, 2);
39     plot(t, xNew, '.b', 'MarkerSize', 10);
40     plot(t, yNew, '.r', 'MarkerSize', 10);
41     legend('Prey Population', 'Predator Population');
42     pause(0.01);
43   end
44 end
```



Obrázek 13.1: Vývoj populace v čase s parametry $\alpha = 1$, $\beta = 1$, $\gamma = 1$ a $\delta = 1$ a s počátečními velikostmi populace $x_0 = 0.9$ a $y_0 = 0.9$.



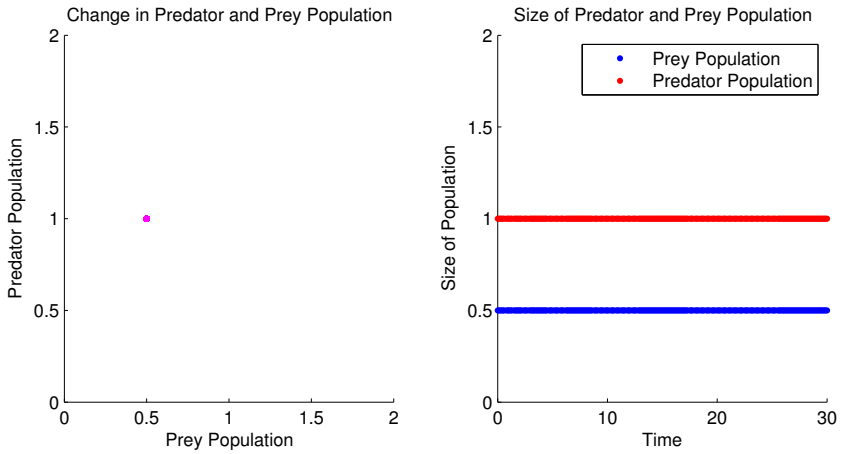
Obrázek 13.2: Chybné grafy vývoje populace v čase. Chyba je způsobena diskretizací s příliš velkým krokem.

Poznámka

Je třeba mít stále na paměti, že tento náš diskrétní postup je pouze aproximace spojitého vývoje populací modelu (13.1). Pokud bychom zvolili příliš velký krok mezi diskrétními časovými body, aproximace by mohla být značně nepřesná. V rekurentních rovnicích může mít totiž i relativně malá chyba velký dopad. To je často způsobeno opakováním iterací, kdy se chyba kumuluje a postupně zvětšuje. Jak podobná chyba může vypadat je ilustrováno na obrázku 13.2, kde jsme za krok zvolili hodnotu 0.1.

13.3 Rovnovážný stav

Dále si vykreslíme tyto grafy pro některé speciální případy parametrů a počátečních stavů. Nejdříve se pokusíme určit, zda v modelu existuje nějaký rovnovážný stav, tedy že obě populace budou mít nulový



Obrázek 13.3: Vývoj populace v čase s parametry $\alpha = 1$, $\beta = 1$, $\gamma = 1$ a $\delta = 2$ a s počátečními velikostmi populace $x_0 = 0.5$ a $y_0 = 1$.

přírůstek. Rovnovážnému stavu odpovídají rovnice

$$\begin{aligned} \alpha x - \beta xy &= 0, \\ \delta xy - \gamma y &= 0, \end{aligned} \tag{13.4}$$

které mají dvě možná řešení

$$x = 0 \quad \text{a} \quad y = 0 \qquad \text{nebo} \qquad x = \frac{\gamma}{\delta} \quad \text{a} \quad y = \frac{\alpha}{\beta}. \tag{13.5}$$

V prvním řešení nemáme žádné králíky ani žádné lišky, obě populace tedy nemohou růst a jejich velikost zůstane na 0. V druhém řešení za počáteční stavy populací zvolíme velikosti $x_0 = \frac{\gamma}{\delta}$ a $y_0 = \frac{\alpha}{\beta}$ a na následujícím obrázku pak uvidíme, že velikosti populací skutečně zůstanou v čase konstantní.

13.4 Chování při extrémních hodnotách parametrů

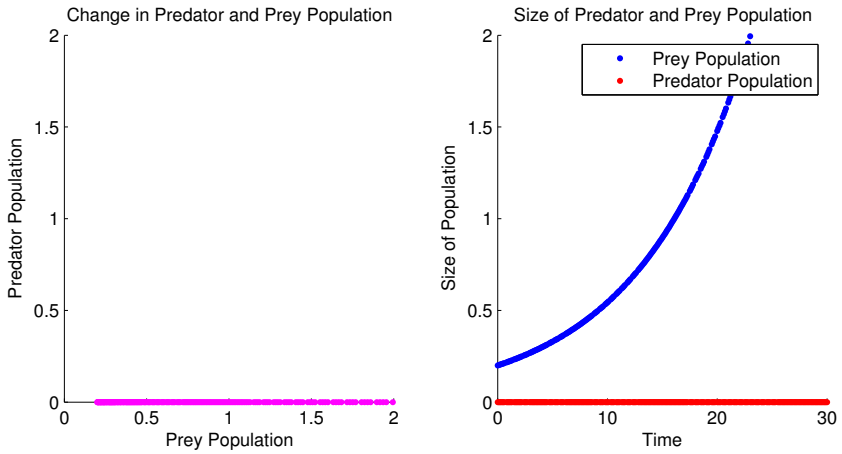
Dále se podíváme na situaci, kdy nebudeme mít žádné lišky, tedy $y_0 = 0$. V tomto případě nebudou králíci nijak vymírat a jejich populace bude exponenciálně růst rychlostí určenou parametrem α.

Podobně můžeme zkoumat situaci, kdy nebudeme mít žádné králíky, tedy $x_0 = 0$. Teď nebudou mít lišky, jak se rozmnožit a jejich populace bude exponenciálně klesat rychlostí určenou parametrem γ.

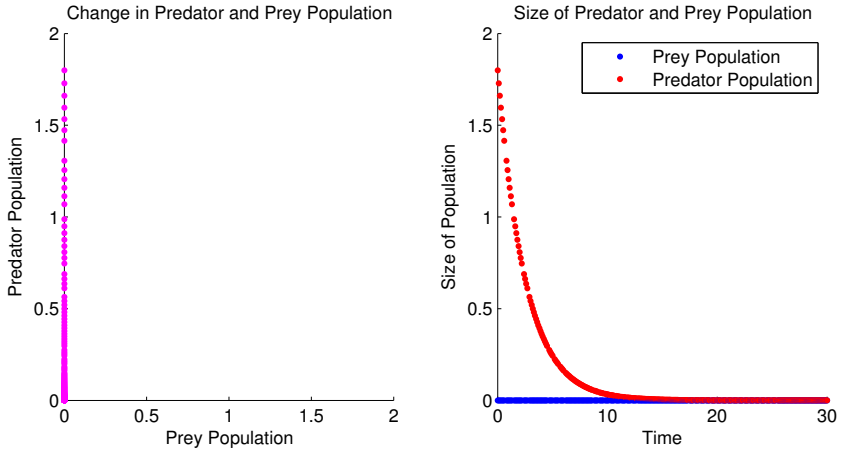
Poznámka

V obou předchozích případech je numerická simulace vlastně zbytečná. V případě $y_0 = 0$ se první rovnice (13.1) zjednoduší na tvar $\frac{dx(t)}{dt} = \alpha x(t)$, která má řešení $x(t) = x_0 e^{\alpha t}$. V případě $x_0 = 0$ se druhá rovnice (13.1) zjednoduší na tvar $\frac{dy(t)}{dt} = -\gamma y(t)$, která má řešení $y(t) = y_0 e^{-\gamma t}$. Simulace jen potvrdila to, co snadno obdržíme analyticky.

Nyní budeme mít v počáteční populaci králíky i lišky a nastavíme parametry tak, aby lišky ze začátku snědly velké množství králíků. Z počátku následujícího grafu se může zdát, že lišky populaci králíků vyhladily úplně, ale není tomu tak. Protože uvažujeme velikosti populace jako spojité veličiny, počet králíků se pouze přiblížil k 0. Jakkoliv malá populace králíků se ovšem může opět rozmnožit do velkých čísel (to platí i pro lišky v podobné situaci).



Obrázek 13.4: Vývoj populace v čase s parametry $\alpha = 0.1$, $\beta = 1$, $\gamma = 1$ a $\delta = 1$ a s počátečními velikostmi populace $x_0 = 0.2$ a $y_0 = 0$.



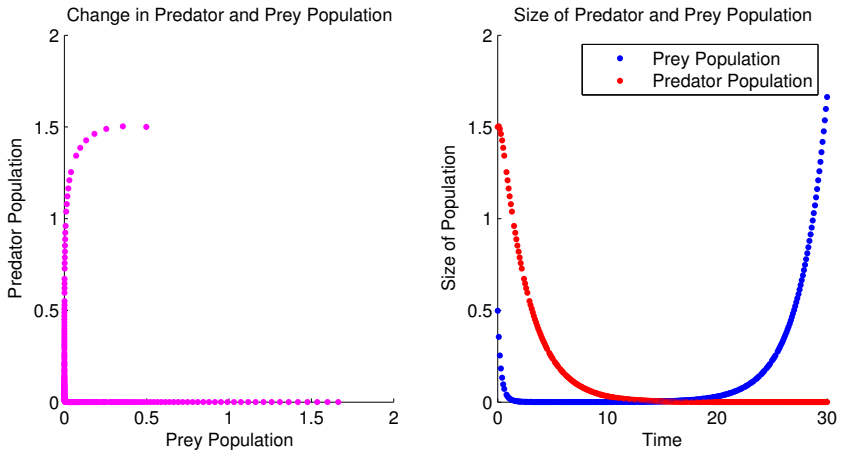
Obrázek 13.5: Vývoj populace v čase s parametry $\alpha = 1$, $\beta = 1$, $\gamma = 0.4$ a $\delta = 1$ a s počátečními velikostmi populace $x_0 = 0$ a $y_0 = 1.8$.

Poznámka

Pokud bychom uvažovali diskrétní hodnoty velikosti populace (tedy pouze celé králíky a lišky), může se pak jedna z populací dostat na 0, což bude mít za následek, že časem vymře i druhá populace.

Poznámka

Pro ilustraci diskrétního času jsme si grafy vykreslovali pouze jako několik bodů. Hezčího vizuálního efektu bychom dosáhli spojením sousedních bodů úsečkami. Tím bychom vlastně z diskrétního grafu s několika body udělali graf spojitý.



Obrázek 13.6: Vývoj populace v čase s parametry $\alpha = 0.4$, $\beta = 2.5$, $\gamma = 0.4$ a $\delta = 1$ a s počátečními velikostmi populace $x_0 = 0.5$ a $y_0 = 1.5$.

Poznámka

Model (13.1) lze dále rošiřovat. Uvažme například myslivce, který ze systému lišky a/nebo králíky odebírá. Pak je přirozené model (13.1) zobecnit do tvaru

$$\begin{aligned} \frac{dx(t)}{dt} &= \alpha x(t) - \beta x(t)y(t) - \mu, \\ \frac{dy(t)}{dt} &= \delta x(t)y(t) - \gamma y(t) - \nu, \end{aligned} \tag{13.6}$$

kde nové parametry μ a ν formalizují intenzitu lovu králíků a lišek. Metodami popsanými v této kapitole lze sledovat chování tohoto modelu. Lze například zkoumat, zda systém stále vykazuje cyklické chování, zda konverguje k rovnovážnému řešení, a případně za jakých podmínek se tak děje.

Jiné rozšíření modelu může spočívat v přidání populace dalšího druhu. Třetí druh může například lovit lišky i králíky, pouze lišky, nebo třeba pouze králíky. Model (13.1) by se pak rozšířil o funkci velikosti třetí populace $z(t)$ a rovnici vyjadřující přírustek třetí populace $\frac{dz(t)}{dt}$.

Doporučená literatura

- Nagle, R. K., Saff, E. B., Snider, A. D. 2018. *Fundamentals of Differential Equations*. Pearson. ISBN 978-0-321-97706-9. https://books.google.com/books?id=H7IVvgAACAAJ
- Shone, R. 2002. *Economic Dynamics: Phase Diagrams and Their Economic Application*. Cambridge University Press. ISBN 978-0-521-81684-7. https://doi.org/10.1017/CBO9781139165020

Kapitola 14

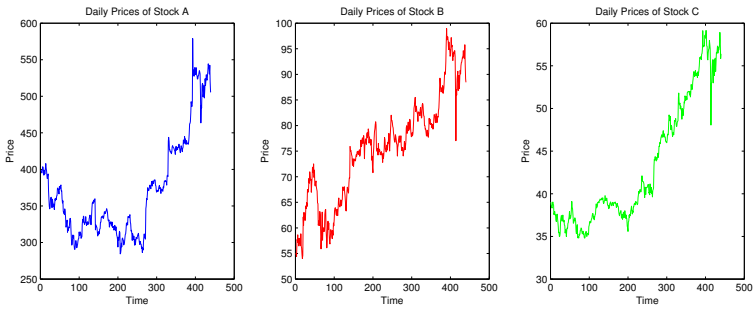
Markowitzův model

Jsme v pozici investora, který se rozhoduje, z jakých akcií složí své portfolio. Chceme mít co nejvyšší střední (očekávaný) výnos portfolia a zároveň co nejmenší riziko případných ztrát. To jsou často protichůdná kritéria (jde o problém vícekriteriální, přeněji dvoukriteriální optimalizace) a musíme tak mezi nimi najít určitý kompromis. Jako nástroj řešení použijeme konvexní kvadratické programování (CQP).

Klíčová slova: *optimalizace portfolia, moderní teorie portfolia, výnos investice, Markowitzův model, kvadratické programování, konvexní programování, CQP, vícekriteriální optimalizace, eficientní hranice.*

14.1 Výnos portfolia

Nechť je naše volba akcií omezena pouze na společnosti Amazon (akcie A), Facebook (akcie B) a Starbucks (akcie C). K dispozice máme denní ceny od začátku ledna roku 2014 do konce září roku 2015. Nejdříve se podíváme, jak časové řady jednotlivých akcií vypadají. Vedle sebe si vykreslíme grafy vývojů cen jednotlivých akcií.



Obrázek 14.1: Vývoje cen jednotlivých akcií.

Markowitz.m: Načtení a zobrazení dat

```
3  data = xlsread('Stocks.xls');
4  X = data(:, [1 3 4]);
5  [n m] = size(X);
6  figure;
7  subplot(1, 3, 1);
8  plot(X(:, 1), 'b');
9  title('Daily Prices of Stock A');
10 xlabel('Time');
11 ylabel('Price');
12 subplot(1, 3, 2);
13 plot(X(:, 2), 'r');
14 title('Daily Prices of Stock B');
15 xlabel('Time');
16 ylabel('Price');
17 subplot(1, 3, 3);
18 plot(X(:, 3), 'g');
19 title('Daily Prices of Stock C');
20 xlabel('Time');
21 ylabel('Price');
```

Nás budou ale spíš než absolutní ceny akcií zajímat jejich denní změny. K tomu použijeme výnosy akcií i v časech t.

Výnos investice

Výnosem se označuje zisk nějaké investice za dané časové období. Nechť je x_s počáteční hodnota investice a x_t konečná hodnota investice. Lze definovat více typů výnosů. Obyčejný výnos je dán předpisem

$$r_{s,t} = \frac{x_t - x_s}{x_s} \tag{14.1}$$

a logaritmický výnos je dán předpisem

$$r_{s,t}^{\log} = \ln \frac{x_t}{x_s} = \ln x_t - \ln x_s. \tag{14.2}$$

Pokud platí $V_s = V_t$, jsou oba druhy výnosů rovny 0. Pokud platí $x_s < x_t$, jsou oba výnosy kladné. A pokud platí $x_s > x_t$, jsou oba výnosy záporné. Kromě případu, kdy jsou oba výnosy rovny 0, jsou hodnoty obyčejných a logaritmických výnosů odlišné. Pro malé hodnoty jsou ale přibližně stejné. Výhoda logaritmických výnosů spočívá především ve snadnějším matematickém použití. Pomocí Taylorova rozvoje můžeme získat aproximaci

$$\ln(1+z) \approx z \qquad \text{pro malé } z,$$

a tedy

$$r_{s,t}^{\log} = \ln \frac{x_t}{x_s} \approx \frac{x_t - x_s}{x_s} = r_{s,t},$$

jsou-li hodnoty $r_{s,t}$ blízko 0. Pracujeme-li s výnosy v krátkém období, např. v denním či hodinovém horizontu, je tento předpoklad realistický.

My v našem příkladě použijeme výnosy logaritmické. Vytvoříme si časové řady logaritmických výnosů $\boldsymbol{y}_i = (y_{i,1}, \ldots, y_{i,n})'$ definované jako

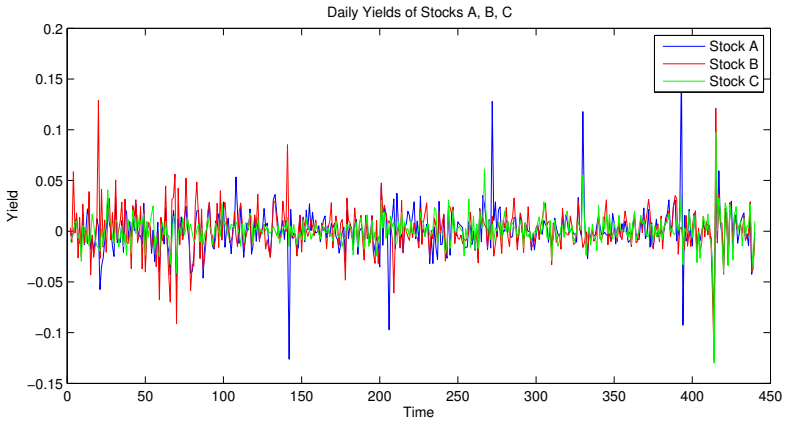
$$y_{i,t} = \ln \frac{x_{i,t}}{x_{i,t-1}} \qquad \text{pro } i = 1, \ldots, m \text{ a } t = 2, \ldots, n, \tag{14.3}$$

kde $x_{i,t}$ jsou minulé ceny i-té akcie v čase t, m je počet akcií (v našem případě $m = 3$) a n je počet pozorování minulých cen. Dále si do jednoho grafu vykreslíme časové řady logaritmických výnosů všech tří akcií.

Markowitz.m: Logaritmické výnosy

```
23 Y = log(X(2:n, :) ./ X(1:(n - 1), :));
24 figure;
25 hold on;
26 xAxis = (2:n)';
27 plot(xAxis, Y(:, 1), 'b');
28 plot(xAxis, Y(:, 2), 'r');
29 plot(xAxis, Y(:, 3), 'g');
30 legend('Stock A', 'Stock B', 'Stock C');
31 title('Daily Yields of Stocks A, B, C');
32 xlabel('Time');
33 ylabel('Yield');
```

Nyní budeme chtít spočítat budoucí výnos celého portfolia. Zatím jsme pracovali pouze s historickými cenami a výnosy, ale teď nás bude zajímat budoucnost. Do našich úvah tedy přidáme modelování



Obrázek 14.2: Vývoj logaritmických výnosů jednotlivých akcií.

náhody. Výnos akcií v čase $t = 2, \ldots, n+1$ budeme uvažovat jako náhodnou veličinu $Y_{i,t}$. Index času t zde zachycuje historické období $t = 2, \ldots, n$ i budoucí období $t = n+1$. V historických obdobích $t = 2, \ldots, n$ ovšem skutečný výnos $y_{i,t}$ známe. Hodnota $y_{i,t}$ je tedy *realizace* náhodné veličiny $Y_{i,t}$. Realizaci $y_{i,n+1}$ v budoucím čase zatím nepozorujeme a musíme si vystačit s jejími odhady.

Celkový výnos portfolia v čase $t = 2, \ldots, n+1$ zavedeme jako

$$Z_t = \sum_{i=1}^{m} w_{i,t} Y_{i,t}, \tag{14.4}$$

kde náhodná veličina $Y_{i,t}$ je výnos i-té akcie a $w_{i,t}$ označuje, jaký podíl celkového rozpočtu je investován do i-té akcie. Neuvažujeme krátké prodeje akcií, vyžadujeme tedy nezápornost $w_{i,t} \geq 0$ pro $i = 1, \ldots, m$. Krátký prodej znamená, že si investor za úplatu akcii „zapůjčí“ a okamžitě ji prodá třetí straně. Za nějaký čas pak akcii zase koupí (tržní cena se mezitím mohla změnit) a vrátí, co si původně půjčil. Jedná se o spekulaci na pokles ceny akcie a matematicky lze tuto operaci vyjádřit právě záporným množstvím. Protože $w_{i,t}$ představují část celku, chceme, aby se sečetly na jedničku, tedy $\sum_{i=1}^{m} w_{i,t} = 1$. Nezabýváme se absolutním množstvím akcií v portfoliu, pouze celkovým výnosem portfolia a relativními zastoupeními akcií.

14.2 Kritéria výběru portfolia

Přirozeně bychom chtěli do portfolia vybrat takové akcie, které nám zajistí co nejvyšší výnos. Protože ovšem budoucí výnos neznáme, musíme použít odhady založené na minulých cenách jednotlivých akcií. Jak jsme již zmínili na začátku, máme dva cíle. Jedním z nich je co nejvyšší očekávaný výnos portfolia všech akcií, který spočítáme jako

$$\mathsf{E} Z_{n+1} = \sum_{i=1}^{m} w_{i,n+1} \mathsf{E} Y_{i,n+1} = \sum_{i=1}^{m} w_{i,n+1} \mu_i = \boldsymbol{w}'_{n+1} \boldsymbol{\mu}, \tag{14.5}$$

kde $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_m)'$ je vektor středních hodnot výnosů akcií a $\boldsymbol{w}_{n+1} = (w_{1,n+1}, \ldots, w_{m,n+1})'$ je vektor vah. Neznámý vektor středních hodnot odhadneme pomocí vektoru výběrových průměru $\hat{\boldsymbol{\mu}} = (\hat{\mu}_1, \ldots, \hat{\mu}_m)'$ s prvky

$$\hat{\mu}_i = \frac{1}{n} \sum_{t=1}^{n} y_{i,t} \qquad \text{pro} \quad i = 1, \ldots, m.$$

Druhým cílem je pak co nejnižší riziko portfolia. To budeme měřit pomocí rozptylu výnosu portfolia. Čím menší rozptyl, tím je i menší pravděpodobnost výskytu extrémních hodnot výnosů (jde nám o snižování rizika extrémně záporných hodnot výnosů). Rozptyl portfolia spočítáme jako

$$\begin{aligned} \mathsf{var} Z_{n+1} &= \sum_{i=1}^{m} w_{i,n+1}^2 \mathsf{var}(Y_{i,n+1}) + \sum_{i \neq j} w_{i,n+1} \mathsf{cov}(Y_{i,n+1}, Y_{j,n+1}) w_{j,n+1} \\ &= \sum_{i=1}^{m} \sum_{j=1}^{m} w_{i,n+1} \Omega_{i,j} w_{j,n+1} \\ &= \boldsymbol{w}'_{n+1} \boldsymbol{\Omega} \boldsymbol{w}_{n+1}, \end{aligned} \tag{14.6}$$

kde $\boldsymbol{\Omega} = (\Omega_{i,j})_{i=1,j=1}^{m,m}$ je kovarianční matice vektoru $(Y_{1,n+1}, \ldots, Y_{m,n+1})'$. Její prvky $\Omega_{i,j}$ jsou kovariance mezi výnosem i-té a j-té akcie pro $i \neq j$ a $\Omega_{i,i}$ je rozptyl výnosu i-té akcie. Neznámou kovarianční matici akcií odhadneme pomocí výběrové kovarianční matice $\hat{\boldsymbol{\Omega}} = (\hat{\Omega}_{i,j})_{i=1,j=1}^{m,m}$ s prvky

$$\hat{\Omega}_{i,j} = \frac{1}{n-1} \sum_{t=1}^{n} (y_{i,t} - \hat{\mu}_i)(y_{j,t} - \hat{\mu}_j) \qquad \text{pro} \quad i = 1, \ldots, m, \quad j = 1, \ldots, m.$$

Markowitz.m: Statistické odhady

```
35 mu = mean(Y)    % vektor vyberovych prumeru
36 Omega = cov(Y) % vyberova kovariancni matice
```

14.3 Optimalizace portfolia

Když máme definováno, jakých cílů chceme dosáhnout, můžeme přistoupit k formulaci samotné optimalizační úlohy, tedy k tzv. Markowitzovu modelu.

Markowitzův model

Teorie portfolia pomáhá určit, v jakém množství zahrnout vybraná aktiva do portfolia, tj. zvolit vektor vah $\boldsymbol{w}_t$ v čase t. Markowitzův model nevybírá aktiva individuálně podle jejich vlastností, ale podle vlastností portfolia jako celku a vzájemných vztahů jednotlivých aktiv. V modelu se optimalizují dvě kritéria: maximální střední hodnota výnosu portfolia a minimální riziko představované rozptylem výnosu portfolia. Tato dvě protichůdná kritéria se spojí v jedno a optimalizační úlohu lze pak formulovat jako

$$\begin{aligned} \max_{\boldsymbol{w}_t} \quad & \boldsymbol{w}_t'\boldsymbol{\mu} - \lambda \boldsymbol{w}_t'\boldsymbol{\Omega}\boldsymbol{w}_t \\ \text{s. t.} \quad & \sum_{i=1}^{m} w_{i,t} = 1, \\ & w_{i,t} \geq 0 \quad \text{pro } i = 1, \ldots, m, \end{aligned} \tag{14.7}$$

kde $\lambda \geq 0$ je parametr vyjadřující velikost naší preference nižšího rizika před vyššími výnosy, $\boldsymbol{\mu}$ je vektor středních hodnot výnosů aktiv, matice $\boldsymbol{\Omega}$ je kovarianční matice výnosů aktiv a $\boldsymbol{w}_t$ je vektor vah aktiv. Pokud se zvolí parametr λ blízko k 0, bude $\lambda \boldsymbol{w}_t'\boldsymbol{\Omega}\boldsymbol{w}_t$ v účelové funkci zanedbatelné a bude se tak preferovat vyšší očekávaný výnos (tzv. risk-lover). Pokud se naopak zvolí parametr λ vysoký, bude $\boldsymbol{w}_t'\boldsymbol{\mu}$ zanedbatelné a bude se tak preferovat nižší očekávaný rozptyl (tzv. konzervativní či rizikově-averzní investor).

Toto je příklad tzv. vícekriteriální, či v našem případě dvoukriteriální optimalizace, ve které obecně není zaveden pojem optimálního řešení. Místo toho se obvykle hledá tzv. eficientní řešení, které se získá podle nějakých dalších kritérií. V našem případě hledáme eficientní řešení $\boldsymbol{w}_t^*$ získané podle parametru λ.

Úloha (14.7) ovšem není jediný možný zápis Markowitzova modelu. Úlohu lze také formulovat jako

$$\begin{aligned} \max_{\boldsymbol{w}} \quad & \boldsymbol{w}'\boldsymbol{\mu} \\ \text{s. t.} \quad & \boldsymbol{w}'\boldsymbol{\Omega}\boldsymbol{w} \leq \omega, \\ & \sum_{i=1}^{m} w_i = 1, \\ & w_i \geq 0 \quad \text{pro } i = 1, \ldots, m, \end{aligned} \tag{14.8}$$

kde parametr $\omega > 0$ určuje omezení na maximální možný přípustný rozptyl portfolia. Další možná formulace je ve tvaru

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & \boldsymbol{w}'\boldsymbol{\Omega}\boldsymbol{w} \\ \text{s. t.} \quad & \boldsymbol{w}'\boldsymbol{\mu} \geq \rho, \\ & \sum_{i=1}^{m} w_i = 1, \\ & w_i \geq 0 \quad \text{pro } i = 1, \ldots, m, \end{aligned} \tag{14.9}$$

kde parametr ρ určuje omezení na minimální možný přípustný výnos portfolia. Lze dokázat, že pokud máme jeden z parametrů λ, ω a ρ pevně daný, můžeme jednoznačně určit oba zbývající parametry tak, že úlohy (14.7), (14.8) a (14.9) budou ekvivalentní. Dále se budeme zabývat už pouze úlohou (14.9), která je ve tvaru tzv. kvadratického programování.

Kvadratické programování

Úlohou konvexního kvadratického programování (CQP) rozumíme optimalizační úlohu, která je typicky zadaná ve tvaru

$$\begin{aligned} \min_{\boldsymbol{z}} \quad & \frac{1}{2}\boldsymbol{z}'\boldsymbol{H}\boldsymbol{z} + \boldsymbol{f}'\boldsymbol{z} \\ \text{s. t.} \quad & \boldsymbol{A}\boldsymbol{z} \leq \boldsymbol{b}, \\ & \boldsymbol{A}_{EQ}\boldsymbol{z} = \boldsymbol{b}_{EQ}, \end{aligned} \tag{14.10}$$

kde $\boldsymbol{z}$ je vektor m proměnných, $\boldsymbol{H}$ je pozitivně semidefinitní matice řádu m, $\boldsymbol{f}$ je vektor s m prvky, $\boldsymbol{A}$ je matice o p řádcích a m sloupcích, $\boldsymbol{b}$ je vektor s p prvky, $\boldsymbol{A}_{EQ}$ je matice o r řádcích a m sloupcích a $\boldsymbol{b}_{EQ}$ je vektor s r prvky. Protože je matice $\boldsymbol{H}$ pozitivně semidefinitní, účelová funkce je konvexní. Kvadratické programování má široké uplatnění. Může se využít např. při odhadu koeficientů lineární regrese pomocí metody nejmenších čtverců, když do modelu přidáme nějaká dodatečná omezení na hledané koeficienty. Snadno se nahlédne, že (14.10) je zobecněním lineárního programování, stačí totiž položit $\boldsymbol{H} = \boldsymbol{0}$.

Nyní převedeme úlohu (14.9) na tvar (14.10). Vytvoříme si matici $\boldsymbol{H}$ o rozměrech $m \times m$, vektor $\boldsymbol{f}$ s m prvky, matici $\boldsymbol{A}$ o rozměrech $(m+1) \times m$, vektor $\boldsymbol{b}$ s $(m+1)$ prvky, matici $\boldsymbol{A}_{EQ}$ o rozměrech $1 \times m$ a vektor $\boldsymbol{b}_{EQ}$ s jedním prvkem

$$\begin{aligned} \boldsymbol{H} &= 2 \cdot \boldsymbol{\Omega}, \\ \boldsymbol{f} &= (0, \ldots, 0)', \\ \boldsymbol{A} &= \begin{pmatrix} -\mu_1 & -\mu_2 & \cdots & -\mu_{m-1} & -\mu_m \\ -1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & -1 & 0 \\ 0 & 0 & \cdots & 0 & -1 \end{pmatrix}, \\ \boldsymbol{b} &= (-\rho, 0, \ldots, 0)', \\ \boldsymbol{A}_{EQ} &= (1, \ldots, 1), \\ \boldsymbol{b}_{EQ} &= 1. \end{aligned} \tag{14.11}$$

Teď již můžeme Markowitzův model s pevně zvoleným parametrem ρ řešit jako úlohu kvadratického programování.

Funkce quadprog

Podobně jako v případě lineárního programování obsahuje Matlab i funkci pro řešení kvadratického programování. Klasicky se tato funkce volá jako

```
x = quadprog(H, f, A, b, Aeq, beq),
```

kde jednotlivé argumenty odpovídají vektorům a maticím modelu (14.10). Do proměnné `x` se uloží optimální řešení úlohy. Pokud budeme chtít více informací o proběhlém výpočtu, můžeme po funkci požadovat více výstupů pomocí zápisu

```
[x, fval, exitflag] = quadprog(H, f, A, b, Aeq, beq).
```

V proměnné `fval` je uložena optimální hodnota účelové funkce. Proměnná `exitflag` indikuje, jak algoritmus skončil, a může nabývat následujících hodnot:

- **1** Funkce našla řešení `x`.
- **0** Překročen maximálně povolený počet iterací.
- **−2** Úloha nemá přípustné řešení.
- **−3** Účelová funkce je neomezená.

Markowitz.m: Markowitzův model

```
38 function [sol val flag] = optiPortfolio(mu, Omega, rho)
39     H = 2 * Omega;
40     f = zeros(m, 1);
41     A = [-mu; -eye(m)];
42     b = [-rho; zeros(m, 1)];
43   Aeq = ones(1, m);
44   beq = 1;
45   [sol val flag] = quadprog(H, f, A, b, Aeq, beq);
46 end
```

14.4 Eficientní hranice

Dále se pokusíme určit, jaká portfolia patří mezi ta nejlepší při různých hodnotách parametru ρ. Budeme postupovat tak, že pro danou hodnotu očekávaného výnosu vybereme vždy portfolio s nejmenším rizikem. Výsledná křivka se nazývá eficientní hranice.

Eficientní hranice

V Markovitzově modelu je eficientní hranice taková množina portfolií, ve které každé portfolio splňuje podmínku, že neexistuje portfolio se stejným středním výnosem a menším rizikem, ani portfolio se stejným rizikem a vyšším výnosem.

Nyní budeme řešit úlohu (14.9) pro různé hodnoty parametru ρ. Pro naše tři akcie víme, že střední výnos jakékoliv kombinace akcií bude aspoň tolik co výnos akcie s nejmenším středním výnosem. Podobně, střední výnos jakékoliv kombinace akcií nebude více než střední výnos akcie s nejvyšším výnosem. Budeme tedy uvažovat 100 rovnoměrně rozmístěných hodnot parametru ρ z tohoto intervalu. Mimo tento

interval bychom totiž nedostali žádná přípustná řešení. Pro každý bod ρ z intervalu nejdříve vyřešíme úlohu (14.9). Získáme tak optimální hodnoty proměnných w_i a optimální velikost rozptylu portfolia pro daný minimální výnos ρ. Pokud existuje řešení této úlohy, necháme si zobrazit dva grafy.

Markowitz.m: Začátek cyklu pro různé minimální střední hodnoty výnosů

```
48 rhoLow = min(mu);
49  rhoUp = max(mu);
50  rhoBy = (rhoUp - rhoLow) / 100;
51 figure;
52 for rho = rhoLow:rhoBy:rhoUp
53   [sol val flag] = optiPortfolio(mu, Omega, rho);
54   if flag == 1
```

V prvním grafu zobrazíme tzv. mean-var prostor. Na horizontální osu vykreslíme očekávaný výnos portfolia, tedy parametr ρ, a na vertikální osu riziko portfolia, tedy optimální hodnotu účelové funkce. Tento graf není přímo eficientní hranice, protože obsahuje portfolia, která při stejné úrovni rizika mohou mít vyšší střední výnos (v obrázku 14.3 jsou to portfolia v levé části prvního grafu). Portfolia, která leží na této hranici, jsou nedominovaná, tzn. že neexistuje portfolio se stejným výnosem a nižším rozptylem. Portfolia, která leží nad touto hranicí, jsou dominovaná, tzn. že existuje portfolio se stejným výnosem a nižším rozptylem, např. portfolio ležící na eficientní hranici.

Markowitz.m: Graf optimálních rozptylů portfolia

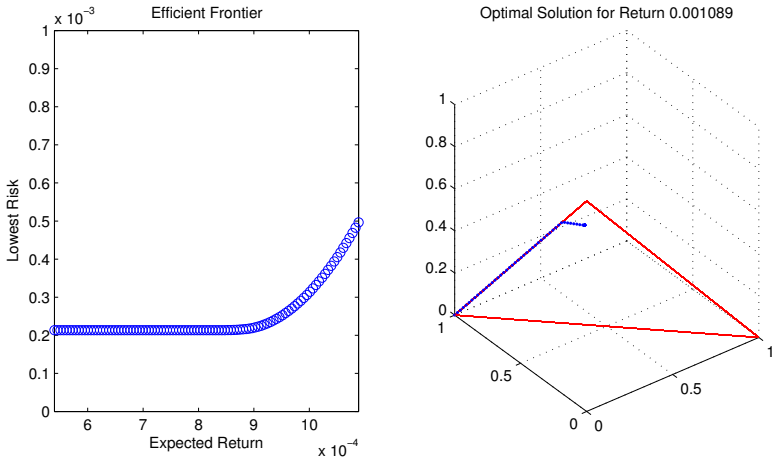
```
55     subplot(1, 2, 1);
56     plot(rho, val, 'o');
57     hold on;
58     axis([rhoLow, rhoUp, 0, 10^-3]);
59     title('Efficient Frontier');
60     xlabel('Expected Return');
61     ylabel('Lowest Risk');
```

V druhém grafu si vykreslíme optimální zastoupení jednotlivých akcií $w_{i,n+1}$ v trojrozměrném prostoru, do kterého si ještě přidáme množinu přípustných řešení. Množina všech možných portfolií $\{\boldsymbol{w}_{n+1} : \boldsymbol{e}'\boldsymbol{w}_{n+1} = 1,\ \boldsymbol{w} > \boldsymbol{0}\}$ tvoří simplex.

Markowitz.m: Graf optimálních řešení

```
62     subplot(1, 2, 2);
63     plot3(sol(1), sol(2), sol(3), '.');
64     plot3([0,0,1,0], [0,1,0,0], [1,0,0,1], 'r');
65     hold on;
66     grid on;
67     rStr = num2str(rho);
68     title(['Optimal Solution for Return ', rStr]);
```

Oba grafy si necháme vykreslit jako postupnou animaci. Toho docílíme tak, že na konci každé iterace cyklu necháme program čekat 0,1 sekundy.



Obrázek 14.3: Graf optimálních rozptylů portfolia pro dané střední výnosy (vlevo) a graf optimálních řešení (vpravo).

Markowitz.m: Konec cyklu

```
69      pause(0.1);
70    end
71  end
```

Zjistili jsme, že ideálně složené portfolio bude jedno z těch, které leží na eficientní hranici. Konkrétní volba našeho portfolia bude potom záviset na naší preferenci vyššího výnosu před nižším rizikem, resp. na minimálním požadovaném výnosu. Pokud si tedy zvolíme nějakou konkrétní dolní hranici očekávaného výnosu ρ, můžeme snadno dopočítat optimální zastoupení $w_{i,n+1}$ jednotlivých akcií v našem portfoliu.

Celý postup můžeme samozřejmě zopakovat i pro jinou trojici akcií, které máme v datech. Pokud vynecháme zobrazení trojrozměrného grafu optimálních zastoupení a řešení si necháme pouze vypsat, můžeme analyzovat i portfolia složená z více akcií.

Doporučená literatura

- Dupačová, J., Lachout, P. 2011. *Úvod do optimalizace*. MatfyzPress. ISBN 978-80-7378-176-7. http://matfyzpress.cz/matematika/20-uvod-do-optimalizace.html
- Francis, J. C., Kim, D. 2013. *Modern Portfolio Theory: Foundations, Analysis, and New Developments*. Wiley. ISBN 978-1-118-37052-0. https://www.wiley.com/en-us/Modern+Portfolio+Theory{%}3A+Foundations{%}2C+Analysis{%}2C+and+New+Developments{%}2C+{%}2B+Website-p-x000610741
- Kwon, R. H. 2013. *Introduction to Linear Optimization and Extensions with MATLAB*. CRC Press. ISBN 978-1-4398-6264-3. https://doi.org/10.1201/b13966

Kapitola 15

Cournotův oligopol

Chování oligopolistů, kteří vyrábějí jeden typ produktu, popisuje např. Cournotův model, jeden z mnoha možných modelů oligopolu. Spočteme si rovnovážný stav oligopolního trhu a ukážeme si vývoj dynamického modelu, ve kterém duopolisté reagují na objem výroby konkurence.

Klíčová slova: *oligopol, duopol, Cournotův model.*

15.1 Cournotův model

V této části si nejdříve představíme Cournotův model pro obecný počet oligopolistů a dále si ukážeme příklad pro dva oligopolisty (tzn. duopolisty).

Cournotův model oligopolu

Uvažujeme, že máme n oligopolistů. Objem výroby oligopolisty i označme q_i a rostoucí funkci celkových nákladů pro objem výroby q_i označme $c_i(q_i)$. Dále označme $p(q)$ klesající cenovou funkci závisející na celkovém objemu produkce všech oligopolistů $q = \sum_{i=1}^{n} q_i$. Zisk jednotlivých oligopolistů v závislosti na objemu výroby můžeme vypočítat pomocí tzv. ziskové funkce

$$z_i(q_1, \ldots, q_n) = p(q)q_i - c_i(q_i).$$

Optimální objem výroby q_i^*, který maximalizuje zisk i-tého oligopolu, musí splňovat podmínku prvního řádu

$$\frac{\partial z_i(q_1, \ldots, q_n)}{\partial q_i} = 0. \tag{15.1}$$

Rovnovážný stav je dán strategiemi $q_1^R, \ldots, q_n^R$, které splňují soustavu rovnic (15.1) pro všechna $i = 1, \ldots, n$.

Pro ilustraci budeme dále uvažovat pouze dva duopolisty a zvolme si cenovou funkci a nákladové

funkce jako

$$\begin{aligned} p(q) &= 9 - q, \\ c_1(q_1) &= 3q_1, \\ c_2(q_2) &= 3q_2, \end{aligned}$$

kde $q = q_1 + q_2$. Ziskové funkce pak můžeme dopočítat jako

$$\begin{aligned} z_1(q_1, q_2) &= (9 - q_1 - q_2)q_1 - 3q_1, \\ z_2(q_1, q_2) &= (9 - q_1 - q_2)q_2 - 3q_2. \end{aligned} \tag{15.2}$$

Z podmínky prvního řádu (15.1) dostaneme rovnice

$$\begin{aligned} q_1 &= 3 - \frac{1}{2}q_2, \\ q_2 &= 3 - \frac{1}{2}q_1, \end{aligned} \tag{15.3}$$

jejichž řešením je rovnovážný stav $q^R = (q_1^R, q_2^R)' = (2, 2)'$.

15.2 Statické chování

V této části uvažujme, že duopolisté mohou rozhodnout o objemu výroby pouze jednou. Nejdříve si sestavíme všechny možné kombinace množství q_1 a q_2, které bereme z mřížky 0 až 6 s krokem 0,1.

DuopolyStatic.m: Mřížka pro množství.

```
3 q1Seq = 0:0.1:6;
4 q2Seq = 0:0.1:6;
5 [q1Grid , q2Grid] = meshgrid(q1Seq, q2Seq);
```

Pro tyto kombinace si vypočteme zisky obou duopolistů pomocí funkcí (15.2).

DuopolyStatic.m: Zisk na mřížce.

```
7 z1Grid = (9 - q1Grid - q2Grid) .* q1Grid - 3 * q1Grid;
8 z2Grid = (9 - q1Grid - q2Grid) .* q2Grid - 3 * q2Grid;
```

Dále určíme tzv. best response rovnice. Optimální objem výroby prvního duopolisty $q_1^*(q_2)$ pro daný objem výroby druhého duopolisty je dán best response rovnicí

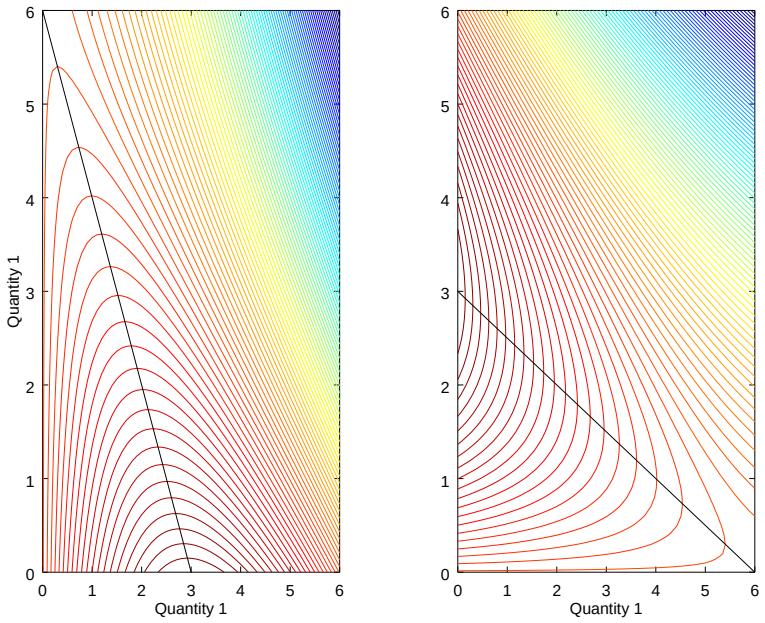
$$q_1^* = 3 - \frac{1}{2}q_2,$$

kterou pro potřeby vykreslení grafu závislosti q_2 na q_1 můžeme upravit do tvaru

$$q_2 = 6 - 2q_1^*.$$

Optimální objem výroby druhého duopolisty $q_2^*(q_1)$ pro daný objem výroby prvního duopolisty je dán best response rovnicí

$$q_2^* = 3 - \frac{1}{2}q_1.$$



Obrázek 15.1: Zisk a optimálního objem výroby duopolu 1 (vlevo) a duopolu 2 (vpravo).

DuopolyStatic.m: Optimální množství.

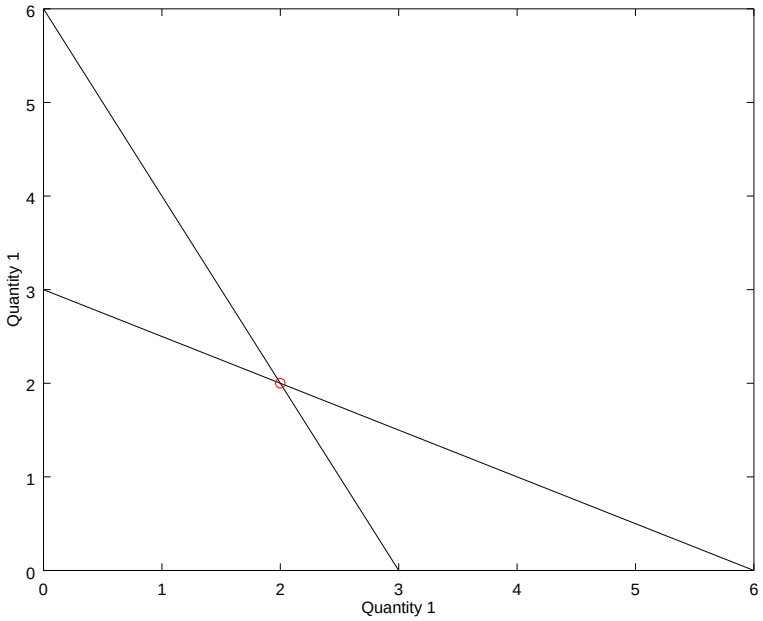
```
10 q2Opti1 = 6 - 2 * q1Seq;
11 q2Opti2 = 3 - 1 / 2 * q1Seq;
```

Teď již můžeme vykreslit izokvanty celkového zisku prvního duopolisty (levý graf obrázku 15.1) a druhého duopolisty (pravý graf obrázku 15.1) pro všechny kombinace množství q_1 a q_2. Do grafu jsme ještě přidali best response křivky.

DuopolyStatic.m: Grafy zisků a optimálního množství.

```
13 figure;
14 subplot(1, 2, 1);
15 contour(q1Grid, q2Grid, z1Grid, 100);
16 hold on;
17 plot(q1Seq, q2Opti1, '-k');
18 xlabel('Quantity 1');
19 ylabel('Quantity 1');
20 subplot(1, 2, 2);
21 contour(q1Grid, q2Grid, z2Grid, 100);
22 hold on;
23 plot(q1Seq, q2Opti2, '-k');
24 xlabel('Quantity 1');
25 ylabel('Quantity 1');
```

Dále vykreslíme best-response křivky do jednoho grafu. Tyto přímky se nám pak protnou v rovnovážném bodě $(2, 2)'$.



Obrázek 15.2: Best-response funkce duopolistů a Nashova rovnováha.

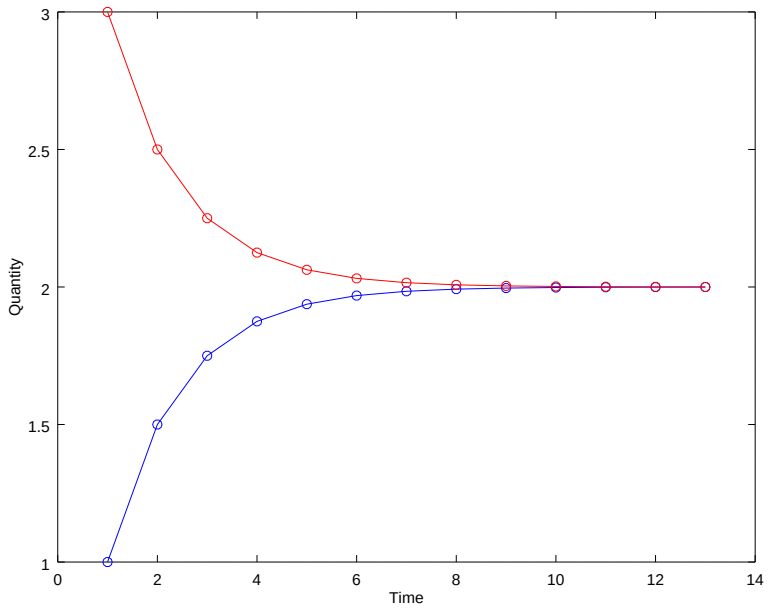
DuopolyStatic.m: Graf průniku best-response křivek.

```
figure;
plot(q1Seq, q2Opti1, '-k');
hold on;
plot(q1Seq, q2Opti2, '-k');
plot(2, 2, 'or');
xlabel('Quantity 1');
ylabel('Quantity 1');
```

15.3 Dynamické chování

Na rozdíl od předchozí sekce teď budeme uvažovat, že se duopolisté mohou o objemu výroby rozhodovat opakovaně. Oba duopolisté budou předpokládat, že jejich konkurent bude vyrábět stejné množství jako při předchozím rozhodnutí. Chování duopolistů budeme sledovat po 12 rozhodnutí, tedy 13 časových období (v čase 1 je objem výroby dán a o ničem se nerozhoduje). Začneme tedy v čase 1 a dostaneme se až do času 13. V každém kroce t duopolisté zvolí objem výroby daný best response rovnicemi

$$\begin{aligned} q_1^{[t]} &= 3 - \frac{1}{2} q_2^{[t-1]}, \\ q_2^{[t]} &= 3 - \frac{1}{2} q_1^{[t-1]}. \end{aligned} \tag{15.4}$$

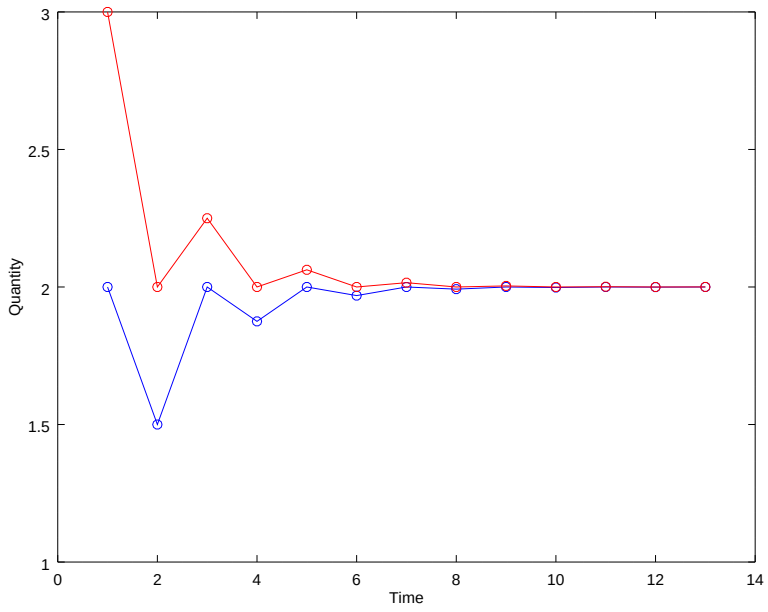


Obrázek 15.3: Vývoj optimálního objemu výroby v čase pro počáteční hodnoty $q_1^{[1]} = 1$ a $q_2^{[1]} = 3$.

DuopolyDynamic.m: Graf vývoje optimálního objemu výroby v čase

```
figure;
q1 = 1; % pocatecni q1
q2 = 3; % pocatecni q2
tMax = 13;
for t = 2:tMax
  q1Last = q1(t - 1);
  q2Last = q2(t - 1);
   q1Cur = 3 - 1 / 2 * q2Last;
   q2Cur = 3 - 1 / 2 * q1Last;
      q1 = [q1 q1Cur];
      q2 = [q2 q2Cur];
end
plot(1:tMax, q1, 'o-b');
hold on;
plot(1:tMax, q2, 'o-r');
xlabel('Time');
ylabel('Quantity');
```

Nejdříve se podívejme na objem výroby $q_1^{[t]}$ a $q_2^{[t]}$ při počátečních hodnotách $q_1^{[1]} = 1$ a $q_2^{[1]} = 3$. Pozorujeme, že se oba objemy výroby v čase symetricky přibližují k rovnovážnému stavu $q_1^R = 2$, $q_2^R = 2$. Pokud zvolíme počáteční hodnoty jako $q_1^{[1]} = 2$ a $q_2^{[1]} = 3$, vidíme, že objem výroby opět skončí v rovnovážném stavu, ale průběh v několika prvních krocích je komplikovanějšího charakteru. Jeden z duopolistů vždy zvolí hodnotu 2, kterou reaguje na předchozí hodnotu 2 druhého duopolisty. Druhý duopolista ovšem mezitím svoji hodnotu 2 změní na jiný objem výroby. Oba duopolisté se tak ze začátku střídají na hodnotách 2, až se ustálí v rovnovážném stavu.



Obrázek 15.4: Vývoj optimálního objemu výroby v čase pro počáteční hodnoty $q_1^{[1]} = 2$ a $q_2^{[1]} = 3$.

Poznámka

Jako cvičení si můžeme předchozí postup rozšířit pro více oligopolistů. Další modifikací může být zvolení jiné cenové funkce nebo jiných nákladových funkcí. Místo lineárních funkcí můžeme například použít kvadratické funkce. Musíme ovšem dodržet, že cenová funkce je klesající a nákladové funkce rostoucí v závislosti na objemu výroby.

Poznámka

Model přizpůsobení můžeme také uvažovat ve verzi se spojitým časem. Clearingové rovnice uvažme ve tvaru

$$q_1'(t) = \mu_1(3 - \frac{1}{2}q_2 - q_1),$$
$$q_2'(t) = \mu_2(3 - \frac{1}{2}q_1 - q_2),$$

kde v závorkách je uveden rozdíl mezi skutečnou produkcí a „správnou“ produkcí danou best-response funkcí. Parametry μ_1 a μ_2 určí rychlost přizpůsobení. Chování tohoto systému diferenciálních rovnic lze studovat a vizualizovat pomocí stejných metod jako v kapitole 13.

Doporučená literatura

- Gravelle, H., Rees, R. 2004. *Microeconomics*. Prentice Hall. ISBN 978-0-582-40487-8
- Shone, R. 2002. *Economic Dynamics: Phase Diagrams and Their Economic Application*. Cambridge University Press. ISBN 978-0-521-81684-7. https://doi.org/10.1017/CBO9781139165020

Kapitola 16

Diferenciální optimalizace

Diferenciální optimalizace je obor, ve kterém se hledá „co nejlepší“ funkce splňující zadaná omezení. Omezení typicky spočívají v podmínkách omezující funkční hodnoty a derivace. Na tento případ lze v jistém smyslu nahlížet jako na optimalizační analogii diferenciálních rovnic. Ukážeme si jeden speciální případ, kdy úlohu diferenciální optimalizace převedeme pomocí diskretizace definičního oboru na lineární program.

Klíčová slova: *diferenciální optimalizace, lineární programování.*

16.1 Formulace problému

Nechť máme zadanou nějakou funkci $f(x)$ s první derivací $f'(x)$ a druhou derivací $f''(x)$ na intervalu $[a, b]$. Tato funkce může mít například tvar

$$f(x) = 0.08\Big[0.2 \sin\Big(4x\Big)\Big(1 - 0.1(x-3)^2\Big) - 0.1(x-5)^2\Big], \qquad x \in [0, 10].$$

Ve zbytku kapitoly budeme pracovat právě s touto funkcí. Zapíšeme si jí tedy do MATLABu.

DiffOpti.m: Zadaná funkce

```
3   function f = funHill(x)
4     f = 0.08 .* (sin(4 .* x) .* 0.2 .*  ...
5       (1 - 0.1 * (x - 3).^2) - 0.1 .* (x - 5).^2);
6   end
```

Obecně mohou být derivace funkce $f(x)$ na definičním oboru libovolně velké. Naším úkolem bude najít funkci $g(x)$, jejíž první i druhé derivace jsou omezeny nějakými danými čísly. V našem příkladě budeme uvažovat, že první derivace musí ležet v intervalu $[-m_1, m_1] = [-0.05, 0.05]$ a druhá derivace v intervalu $[-m_2, m_2] = [-0.1, 0.1]$ na celém definičním oboru. Dále o funkci $g(x)$ předpokládáme, že v krajních bodech a a b bude nabývat stejných hodnot jako funkce $f(x)$.

DiffOpti.m: Zadaná omezení na derivace

```
8   max1d = 0.05;
9   max2d = 0.10;
```

Navíc budeme hledat takovou funkci $g(x)$, která se bude nejméně „lišit“ od $f(x)$. Odlišnost funkcí budeme měřit pomocí velikosti plochy ohraničené funkcemi $f(x)$ a $g(x)$. Půjde nám tedy o optimalizační úlohu

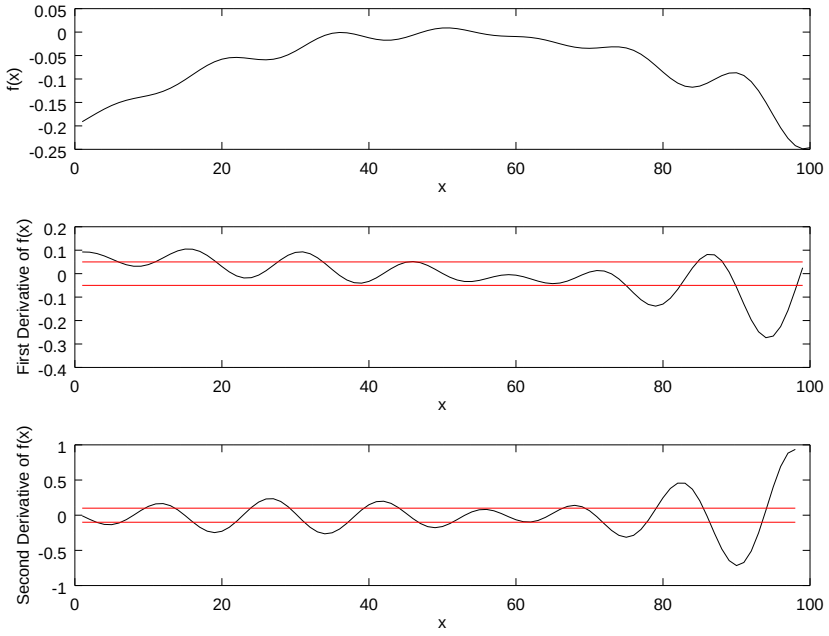
$$\begin{aligned} \min_{g(x)} \quad & \int_a^b \Big| f(x) - g(x) \Big| dx \\ \text{s.t.} \quad & g(a) = f(a), \\ & g(b) = f(b), \\ & \Big| \frac{\partial g(x)}{\partial x} \Big| \leq m_1 \qquad \forall x \in (a,b), \\ & \Big| \frac{\partial^2 g(x)}{\partial x^2} \Big| \leq m_2 \qquad \forall x \in (a,b). \end{aligned} \tag{16.1}$$

Než se pustíme do samotného řešení této úlohy, tak se podíváme, jak naše funkce $f(x)$ a její derivace vypadají. Vypočteme si hodnoty funkce $f(x)$ pro body z mřížky s krokem 0.1 a sousední hodnoty spojíme úsečkou. První i druhou derivaci aproximujeme pomocí diferencí ze stejné mřížky.

DiffOpti.m: Výpočet funkce f a jejích derivací

```
11  xFrom = 0.1;
12  xTo = 10;
13  xBy = 0.1;
14  x = xFrom:xBy:xTo; % mrizka
15  n = length(x);
16  f = funHill(x); % zadana funkce
17  f1d = diff(f) / xBy; % prvni derivace
18  f2d = diff(f1d) / xBy; % druha derivace
```

Aproximace funkcí $f(x)$, $f'(x)$ a $f''(x)$ si vykreslíme pod sebe a přídáme omezující intervaly pro první a druhou derivaci. Vidíme, že zadaná funkce v nějakých částech definičního intervalu podmínky na první i druhou derivaci nesplňuje. Hledání funkce $g(x)$ má tedy smysl.



Obrázek 16.1: Funkce $f(x)$ a její první a druhá derivace.

DiffOpti.m: Vykreslení funkce f a jejích derivací

```
20  figure(1);
21  subplot(3, 1, 1);
22  plot(f, '-k');
23  hold on;
24  xlabel('x');
25  ylabel('f(x)');
26  subplot(3, 1, 2);
27  plot(f1d, '-k');
28  hold on;
29  plot([1, n - 1], [ max1d,  max1d], '-r');
30  plot([1, n - 1], [-max1d, -max1d], '-r');
31  xlabel('x');
32  ylabel('First Derivative of f(x)');
33  subplot(3, 1, 3);
34  plot(f2d, '-k');
35  hold on;
36  plot([1, n - 2], [ max2d,  max2d], '-r');
37  plot([1, n - 2], [-max2d, -max2d], '-r');
38  xlabel('x');
39  ylabel('Second Derivative of f(x)');
```

Poznámka

Úloha může mít následující interpretaci. Funkci $f(x)$ si můžeme představit jako reliéf hor. Na těchto horách chceme postavit silnici. Aby byla silnice sjízdná, nesmí být kopec příliš strmý. Z kopců můžeme nějakou půdu odebrat, do dalších části můžeme naopak půdu přidat. Vytvoříme tak nový reliéf $g(x)$. Přirozeně nám jde o co nejméně zásahů do krajiny, což vyjadřuje minimalizace integrálu rozdílu funkcí v absolutní hodnotě. Podmínka na první derivaci zajišťuje, že nová silnice nebude příliš stoupat ani příliš klesat. Podmínka na druhou derivaci zajišťuje, že se stoupání a klesání příliš divoce nemění.

Poznámka

V této formulaci problému nám nezáleží na tom, jakou část úprav věnujeme odebírání půdy z kopce a jakou přidávání. V některých situacích by ale bylo ruzumné uvažovat, že půdu, kterou někde vykopáme, musíme přesunout do jiné části kopce. Celková velikost nového pohoří by tak byla stejná jako velikost původního pohoří. Toho lze docílit přidáním jedné podmínky do úlohy a čtenáři tuto modifikaci ponecháme jako cvičení.

16.2 Diskretizace úlohy a řešení pomocí lineárního programování

Úlohu (16.1) je nesnadné řešit, protože se v ní hledá nekonečně mnoho bodů funkce $g(x)$. Tuto úlohu si upravíme tak, že budeme hledat pouze konečný počet bodů funkce $g(x)$. Nebudeme mít tedy funkci $g(x)$ definovanou pro spojitou proměnnou x z intervalu $[a, b]$, ale pouze pro hodnoty $x_1, \ldots, x_n$. Body x_i pro $i = 1, \ldots, n$ vybereme jako ekvidistantní mřížku s krokem $\delta = 0.1$ a krajními body $x_1 = a$ a $x_n = b$. Hodnoty $g(x_i)$ pro $i = 1, \ldots, n$ pro jednoduchost označíme jako g_i. Podobně hodnoty $f(x_i)$ pro $i = 1, \ldots, n$ označíme jako f_i.

Od spojitého problému jsme přešli k diskrétní úloze. Integrál v účelové funkci nahradíme sumou. Derivace v podmínkách nahradíme diferencemi. Úlohu (16.1) tak převedeme na

$$\begin{aligned}
\min_{g_1,\ldots,g_n} \quad & \textstyle\sum_{i=1}^{n} \delta \left| f_i - g_i \right| \\
\text{s.t.} \quad & g_1 = f_1, \\
& g_n = f_n, \\
& \left| \frac{g_i - g_{i-1}}{\delta} \right| \leq m_1, \qquad i = 2, \ldots, n, \\
& \left| \frac{g_i - 2g_{i-1} + g_{i-2}}{\delta^2} \right| \leq m_2, \qquad i = 3, \ldots, n,
\end{aligned}$$

Zde již máme konečný počet proměnných. Úlohu ještě můžeme přeformulovat tak, abychom získali úlohu lineární programování. Toho docílíme tak, že do zápisu přidáme pomocné proměnné $y_1, \ldots, y_n$

a zbavíme se absolutních hodnot v účelové funkci i v podmínkách. Získáme

$$\begin{aligned}
\min_{\substack{g_1,\dots,g_n,\\ y_1,\dots,y_n}} \quad & \textstyle\sum_{i=1}^{n} \delta y_i \\
\text{s.t.} \quad & g_1 = f_1, \\
& g_n = f_n, \\
& g_i - g_{i-1} \le m_1\delta, && i = 2,\dots,n, \\
& g_i - g_{i-1} \ge -m_1\delta, && i = 2,\dots,n, \\
& g_i - 2g_{i-1} + g_{i-2} \le m_2\delta^2, && i = 3,\dots,n, \\
& g_i - 2g_{i-1} + g_{i-2} \ge -m_2\delta^2, && i = 3,\dots,n, \\
& y_i \ge g_i - f_i, && i = 1,\dots,n, \\
& y_i \ge f_i - g_i, && i = 1,\dots,n,
\end{aligned}$$

Tento zápis již odpovídá úloze lineárního programování. Dále převedeme všechny proměnné na levou stranu a konstanty na pravou stranu a upravíme všechny rovnosti na „$\le$“. Získáme úlohu

$$\begin{aligned}
\min_{\substack{g_1,\dots,g_n,\\ y_1,\dots,y_n}} \quad & \textstyle\sum_{i=1}^{n} \delta y_i \\
\text{s.t.} \quad & g_1 \le f_1, && && \text{(C1)} \\
& -g_1 \le -f_1, && && \text{(C2)} \\
& g_n \le f_n, && && \text{(C3)} \\
& -g_n \le -f_n, && && \text{(C4)} \\
& g_i - g_{i-1} \le m_1\delta, && i = 2,\dots,n, && \text{(C5)} \\
& -g_i + g_{i-1} \le m_1\delta, && i = 2,\dots,n, && \text{(C6)} \\
& g_i - 2g_{i-1} + g_{i-2} \le m_2\delta^2, && i = 3,\dots,n, && \text{(C7)} \\
& -g_i + 2g_{i-1} - g_{i-2} \le m_2\delta^2, && i = 3,\dots,n, && \text{(C8)} \\
& g_i - y_i \le f_i, && i = 1,\dots,n, && \text{(C9)} \\
& -g_i - y_i \le -f_i, && i = 1,\dots,n. && \text{(C10)}
\end{aligned}$$

Nakonec převedeme tento zápis do maticového tvaru

$$\begin{aligned}
\min_{\boldsymbol{z}} \quad & \boldsymbol{c}'\boldsymbol{z} \\
\text{s.t.} \quad & \boldsymbol{A}\boldsymbol{z} \le \boldsymbol{b},
\end{aligned}$$

kde jsme označili vektor proměnných jako $\boldsymbol{z} = (g_1,\dots,g_n,y_1,\dots,y_n)'$. Matici levých stran podmínek

a vektor pravých stran podmínek jsme označili jako

$$\boldsymbol{A} = \begin{pmatrix}
1 & 0 & & & \dots & & & 0 & 0 & 0 & & \dots & & 0 \\
-1 & 0 & & & \dots & & & 0 & 0 & 0 & & \dots & & 0 \\
0 & 0 & & & \dots & & & 0 & 1 & 0 & & \dots & & 0 \\
0 & 0 & & & \dots & & & 0 & -1 & 0 & & \dots & & 0 \\
1 & -1 & 0 & & \dots & & 0 & 0 & 0 & 0 & & \dots & & 0 \\
 & & & & \ddots & & & & & & & & & \\
0 & 0 & 0 & & \dots & & 0 & 1 & -1 & 0 & & \dots & & 0 \\
-1 & 1 & 0 & & \dots & & 0 & 0 & 0 & 0 & & \dots & & 0 \\
 & & & & \ddots & & & & & & & & & \\
0 & 0 & 0 & & \dots & & 0 & -1 & 1 & 0 & & \dots & & 0 \\
1 & -2 & 1 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & & \dots & & 0 \\
 & & & & \ddots & & & & & & & & & \\
0 & 0 & 0 & 0 & \dots & 0 & 1 & -2 & 1 & 0 & & \dots & & 0 \\
-1 & 2 & -1 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & & \dots & & 0 \\
 & & & & \ddots & & & & & & & & & \\
0 & 0 & 0 & 0 & \dots & 0 & -1 & 2 & -1 & 0 & & \dots & & 0 \\
1 & 0 & & & \dots & & & 0 & 0 & -1 & 0 & \dots & 0 & 0 \\
 & & & & \ddots & & & & & & & \ddots & & \\
0 & 0 & & & \dots & & & 0 & 1 & 0 & 0 & \dots & 0 & -1 \\
-1 & 0 & & & \dots & & & 0 & 0 & -1 & 0 & \dots & 0 & 0 \\
 & & & & \ddots & & & & & & & \ddots & & \\
0 & 0 & & & \dots & & & 0 & -1 & 0 & 0 & \dots & 0 & -1
\end{pmatrix}$$

(C1), (C2), (C3), (C4), (C5), (C6), (C7), (C8), (C9), (C10)

$$, \quad \boldsymbol{b} = \begin{pmatrix}
f_1 \\ -f_1 \\ f_n \\ -f_n \\ m_1\delta \\ \vdots \\ m_1\delta \\ m_1\delta \\ \vdots \\ m_1\delta \\ m_2\delta^2 \\ \vdots \\ m_2\delta^2 \\ m_2\delta^2 \\ \vdots \\ m_2\delta^2 \\ f_1 \\ \vdots \\ f_n \\ -f_1 \\ \vdots \\ -f_n
\end{pmatrix}.$$

Vektor účelové funkce jsme označili jako

$$\boldsymbol{c} = (0, \dots, 0, \delta, \dots, \delta)'.$$

Maticový zápis úlohy použijeme pro solver `linprog`. Nalezení hodnot $g_1, \dots, g_n$ funkce $g(x)$ si naprogramujeme jako samostatnou funkci.

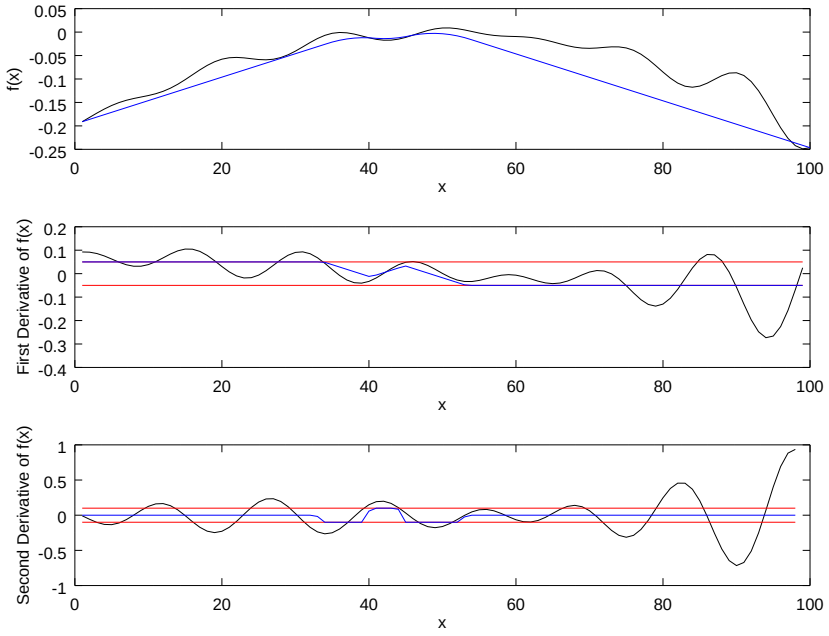
DiffOpti.m: Nalezení nové funkce splňující podmínky

```
41  function g = newHill(x, f, max1d, max2d, xBy)
42    n = length(x);
43    c = [ zeros(n, 1);                              % c
44          xBy * ones(n, 1)];
45    A = [ 1, zeros(1, 2 * n - 1);                   % A (C1)
46         -1, zeros(1, 2 * n - 1);                   % A (C2)
47          zeros(1, n - 1),  1, zeros(1, n);         % A (C3)
48          zeros(1, n - 1), -1, zeros(1, n);         % A (C4)
49          [ eye(n - 1), zeros(n - 1, 1)] + ...      % A (C5)
50            [zeros(n - 1, 1), -eye(n - 1)], ...
51            zeros(n - 1, n);
52          [-eye(n - 1), zeros(n - 1, 1)] + ...      % A (C6)
53            [zeros(n - 1, 1),  eye(n - 1)], ...
54            zeros(n - 1, n);
55          [ eye(n - 2), zeros(n - 2, 2)] + ...      % A (C7)
56            [zeros(n - 2, 1), -2 * eye(n - 2), ...
57            zeros(n - 2, 1)] + [zeros(n - 2, 2),  ...
58            eye(n - 2)], zeros(n - 2, n);
59          [-eye(n - 2), zeros(n - 2, 2)] + ...      % A (C8)
60            [zeros(n - 2, 1),  2 * eye(n - 2), ...
61            zeros(n - 2, 1)] + [zeros(n - 2, 2), ...
62            -eye(n - 2)], zeros(n - 2, n);
63          eye(n), -eye(n);                          % A (C9)
64         -eye(n), -eye(n)];                         % A (C10)
65    b = [ f(1);                                     % b (C1)
66         -f(1);                                     % b (C2)
67          f(n);                                     % b (C3)
68         -f(n);                                     % b (C4)
69          max1d * xBy * ones(n - 1, 1);             % b (C5)
70          max1d * xBy * ones(n - 1, 1);             % b (C6)
71          max2d * xBy^2 * ones(n - 2, 1);           % b (C7)
72          max2d * xBy^2 * ones(n - 2, 1);           % b (C8)
73          f';                                       % b (C9)
74         -f'];                                      % b (C10)
75    sol = linprog(c, A, b);
76    g = sol(1:n);
77  end
```

Teď již máme všechny prostředky pro nalezení funkce $g(x)$, která bude splňovat podmínky na derivace $m_1 = 0.05$ a $m_2 = 0.1$. Kromě samotných hodnot funkce $g(x)$ si spočteme i hodnoty její první a druhé derivace.

DiffOpti.m: Výpočet funkce g a jejích derivací

```
79  g = newHill(x, f, max1d, max2d, xBy);
80  g1d = diff(g) / xBy;
81  g2d = diff(g1d) / xBy;
```



Obrázek 16.2: Funkce $f(x)$ (černá) a $g(x)$ (modrá) a jejich první a druhé derivace.

Hodnoty nalezené funkci $g(x)$ přidáme do grafu z předchozí části. Vidíme, že derivace funkce $g(x)$ splňují zadaná omezení.

DiffOpti.m: Vykreslení funkce g a jejích derivací

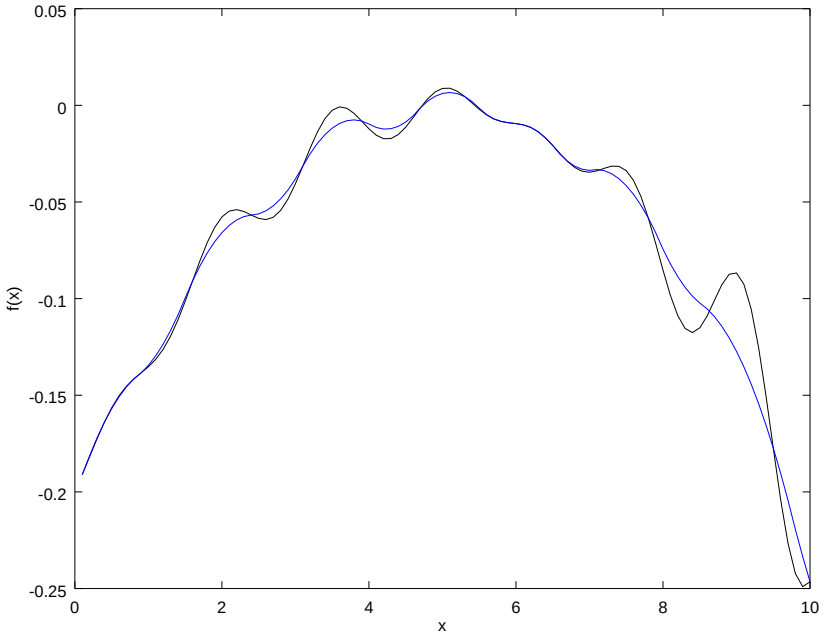
```
83  figure(1);
84  subplot(3, 1, 1);
85  plot(g, '-b');
86  subplot(3, 1, 2);
87  plot(g1d, '-b');
88  subplot(3, 1, 3);
89  plot(g2d, '-b');
```

Poznámka

Pokud bychom předpokládali nějaký tvar funkce $g(x)$, situace by byla jiná. Funkce $g(x)$ by patřila do nějaké třídy funkcí $\{G_{p_1,\ldots,p_m}|p_1,\ldots,p_m\}$ s parametry $p_1,\ldots,p_m$. Mohlo by se jednat například o funkce ve tvaru

$$G(x) = p_1 \sin(p_2 x) + p_3(x + p_4)^{p_5}.$$

Integrály a derivace v úloze (16.1) by pak měli analytické řešení (pokud budou funkce $f(x)$ a $G(x)$ „rozumné“) v závislosti na parametrech $p_1,\ldots,p_m$. Po analytické úpravě bychom pak dostali (obecně nelineární) účelovou funkci a soustavu podmínek, která už bude záviset pouze na konečném počtu m proměnných. Tento postup by ovšem mohl být značně komplikovaný, a proto je dobře, že v naší úloze nic takového nepředpokládáme. Získali jsme obecnější nástroj na řešení úloh tohoto typu (byť za cenu numerické chyby při diskretizaci).



Obrázek 16.3: Funkce $f(x)$ (černá) a $g(x)$ (modrá) při omezení $m_1 = 0.15$.

16.3 Chování pro různá omezení na derivaci

Nakonec se podíváme, jak budou vypadat funkce $g(x)$ pro různě přísná omezení na první derivaci. Budeme uvažovat omezení m_1 od 0.02 do 0.15 s krokem 0.01. Výpočet hodnot funkce $g(x)$ zabalíme do `for` cyklu a v každé iteraci si vykreslíme funkci $g(x)$ pro jinou hodnotu m_1. Pokud na konec každé iterace přidáme příkaz `pause`, graf se nám bude překreslovat pomalu, čímž získáme efekt animace.

DiffOpti.m: Různá omezení na derivaci.

```
91   figure(2);
92   for max1d = 0.02:0.01:0.15
93     g = newHill(x, f, max1d, max2d, xBy);
94     plot(x, f, '-k');
95     hold on;
96     plot(x, g, '-b');
97     hold off;
98     xlabel('x');
99     ylabel('f(x)');
100    pause(1);
101  end
```

Kapitola 17

Odhad rizika

Budeme zabývat analýzou rizika u akcií. Na rozdíl od kapitoly 14 se omezíme na jednorozměrnou časovou řadu, tedy pouze na jednu akcii. Riziko je možné modelovat více způsoby. Už jsme si představovali modelování rizika pomocí rozptylu a nyní si představíme jiný způsob, tzv. Value-at-Risk.

Klíčová slova: *Value-at-Risk, Wienerův proces, Interval spolehlivosti, Bootstrapová metoda.*

17.1 Měření rizika pomocí Value-at-Risk

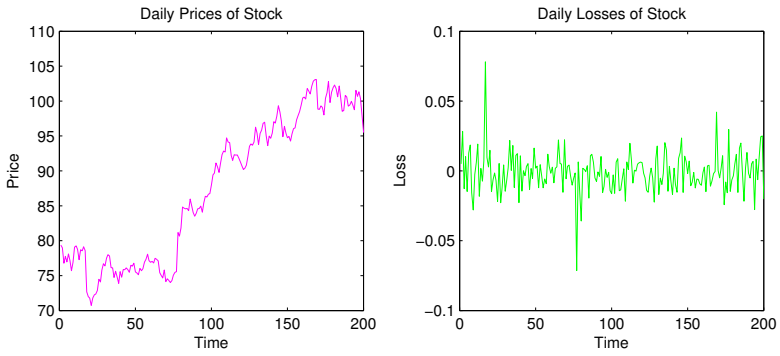
Nejdříve si načteme časovou řadu x_t, $t = 1, \ldots, n$ historických cen akcie společnosti Apple. Podobně jako v kapitole 14 řadu transformujeme na logaritmické výnosy. Protože řízení rizika se zabývá především negativními důsledky, budeme v této kapitole pracovat s logaritmickými ztrátami

$$z_t = -\ln \frac{x_t}{x_{t-1}} \qquad \text{pro } t = 2, \ldots, n, \tag{17.1}$$

ValueAtRisk.m: Načtení a úprava dat

```
3   n = 200;
4   data = xlsread('Stocks.xls');
5   X = data(1:(n + 1), 2);                % cena
6   Y = log(X(2:(n + 1), 1) ./ X(1:n, 1)); % log vynos
7   Z = -Y;                                % log ztrata
```

Vývoj absolutních cen a logaritmických ztrát si zakreslíme vedle sebe do grafu.



Obrázek 17.1: Absolutní ceny (vlevo) a logaritmické ztráty (vpravo) akcie společnosti Apple.

ValueAtRisk.m: Zobrazení dat

```
figure(1);
subplot(1, 2, 1);
plot(X(:, 1), 'm');
axis([0 n 70 110]);
title('Daily Prices of Stock');
xlabel('Time');
ylabel('Price');
subplot(1, 2, 2);
plot(Z(:, 1), 'g');
axis([0 n -0.1 0.1]);
title('Daily Losses of Stock');
xlabel('Time');
ylabel('Loss');
```

K dispozici máme historické ztráty denní $z_1, \ldots, z_n$. Nás bude zajímat budoucí denní (zítřejší) ztráta. Její hodnotu samozřejmě dnes ještě neznáme a budeme jí tedy modelovat jako náhodnou veličinu Z_{n+1}. Za předpokladu, že denní ztráty jsou nezávislé náhodné veličiny se stejným rozdělením, můžeme z historických dat odhadnout kromě střední hodnoty a rozptylu i tzv. Value-at-Risk.

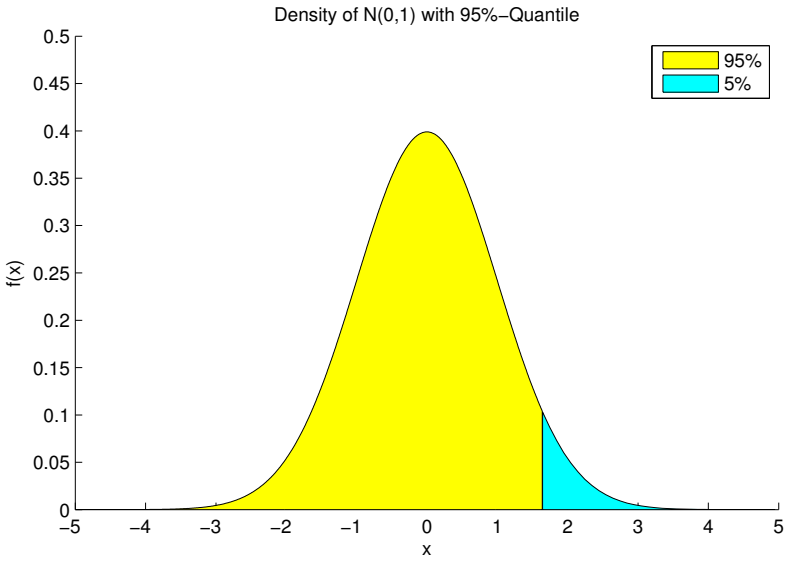
Value-at-Risk

Value at Risk (VaR; do češtiny by se dalo přeložit jako *hodnota v riziku*) je míra rizika často používaná ve financích. Matematicky se jedná o jednostranný kvantil (zpravidla se používá $\alpha = 0.95$) z rozdělení ztrát aktiva v průběhu určité doby. Pokud označíme náhodnou veličinu ztrát aktiva Z, α-procentní Value-at-Risk lze definovat jako

$$VaR_\alpha = \inf\Big\{z : P(Z \leq z) \geq \alpha\Big\} = \inf\Big\{z : F_Z(z) \geq \alpha\Big\},$$

kde $F_Z(z)$ je distribuční funkce náhodné veličiny Z.

Kvantil normálního normovaného rozdělení si můžeme ilustrovat na obrázku hustoty. 95%-kvantil je zjednodušeně taková hodnota, při které plocha pod funkcí hustoty nalevo od kvantilu tvoří 95 % celkové plochy pod funkcí hustoty. Tuto hodnotu můžeme nalézt pomocí kvantilové funkce `icdf`.



Obrázek 17.2: Kvantil normálního rozdělení.

NormQuantile.m: Kvantil normálního rozdělení

```
3   figure(1);
4   title('Density of N(0,1) with 95%-Quantile');
5   xlabel('x');
6   ylabel('f(x)');
7   hold on;
8   axis([-5 5 0 0.5]);
9   varNorm = icdf('norm', 0.95, 0, 1);
10  leftX = -5:0.01:varNorm; % hodnoty mensi nez kvantil
11  leftY = pdf('norm', leftX, 0, 1);
12  area(leftX, leftY, 'FaceColor', 'y');
13  rightX = varNorm:0.1:5; % hodnoty vetsi nez kvantil
14  rightY = pdf('norm', rightX, 0, 1);
15  area(rightX, rightY, 'FaceColor', 'c');
16  legend('95%', '5%');
```

17.2 Výpočet Value-at-Risk historickou metodou

Value-at-Risk můžeme odhadnout více způsoby. Začneme tzv. historickou metodou. Ta se vyznačuje tím, že jediným jejím předpokladem je nezávislost a stejné rozdělení náhodných veličin ztrát $Z_1, \ldots, Z_n, Z_{n+1}$. Historická data tak můžeme brát jako n realizací náhodné veličiny Z_{n+1}. Value-at-Risk vypočítáme jako empirický kvantil. Našich 200 historických ztrát seřadíme od nejmenší po největší a za odhad 0.95-kvantilu vezmemu tu $0.95 \cdot 200 = 190$. největší ztrátu.

V MATLABu spočítáme empirický kvantil pomocí funkce `quantile`. Výpočet Value-at-Risk historickou metodou si naprogramujeme jako funkci.

ValueAtRisk.m: Výpočet empirického VaR

```
23  function VaREmp = myVaREmpiric(Z)
24    VaREmp = quantile(Z, 0.95);
25  end
```

Poznámka

Předpoklad o stejném rozdělení cen či ztrát je v realitě problematický. Jednoduše řečeno je velmi naivní předpokládat, že se ceny budou chovat stejně dnes jako před rokem. V delším časovém horizontu totiž finanční řady vykazují změny svých charakteristik, a tak se často dává přednost dynamickým modelům, které se pokouší tyto změny zachytit.

17.3 Výpočet Value-at-Risk modelováním Wienerova procesu

Dalším možným způsobem odhadu je tzv. parametrická metoda. Ta předpokládá, že se ztráty chovají podle nějakého známeho modelu s neznámými parametry. Složitost těchto modelů může být různá, my se budeme zabývat jedním z těch nejjednodušších. Ceny akcie budeme modelovat pomocí tzv. Wienerova procesu.

Wienerův proces

Wienerův proces W_t je typ spojitého stochastického procesu. Stochastický proces je množina náhodných veličin W_t, kde t značí čas, který může být obecně být diskrétní (tzv. časová řada) nebo spojitý (tzv. spojitý stochastický proces). Wienerův proces je spojitý stochastický proces charakterizovaný následujícími vlastnostmi:

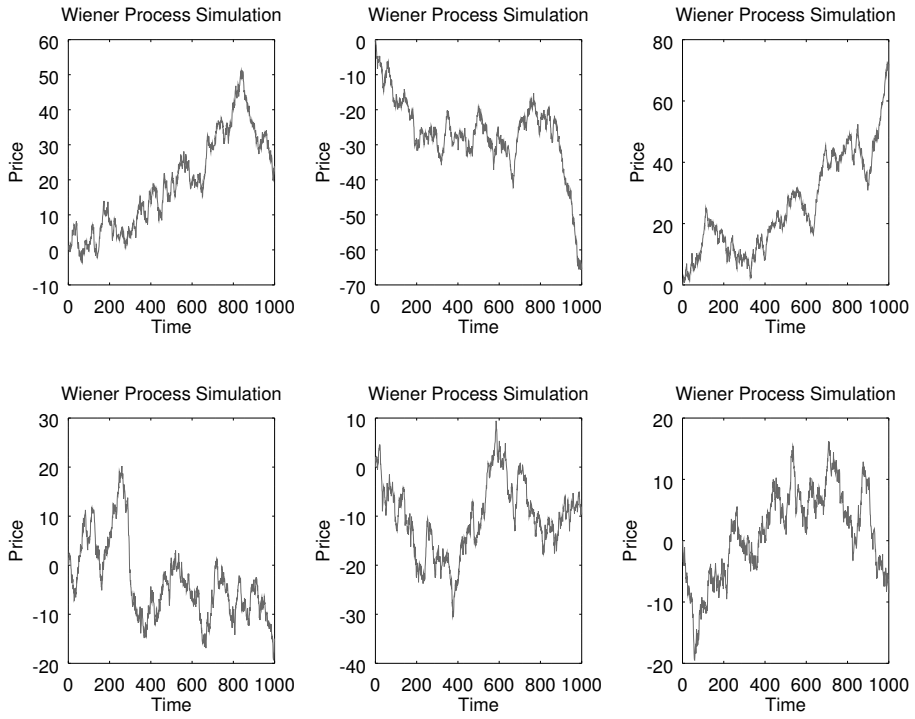
- $W_0 = 0$,
- Rozdělení přírustku $W_{t_1} - W_{s_1}$ je nezávislé na $W_{t_2} - W_{s_2}$ pro každou volbu $0 \leq s_1 \leq t_1 \leq s_2 \leq t_2$,
- Přírustek $W_t - W_s$ má normální rozdělení $N(0, t - s)$,
- Proces W_t je téměř jistě spojitý.

Rozšířením Wienerova procesu je tzv. Wienerův proces s driftem μ a směrodatnou odchylkou σ definovaný jako

$$Y_t = \mu t + \sigma W_t.$$

Přírustek tohoto procesu $Y_t - Y_s$ má normální rozdělení $\mathrm{N}\Big(\mu(t-s), \sigma(t-s)\Big)$.

Na chvilku odbočme od Value-at-Risk a podívejme se, jak Wienerův proces může vypadat. Protože se jedná o náhodný proces, každá jeho realizace probíhá jinak. Nasimulujeme si 6 realizací tohoto procesu v čase od 0 do 1000. Využijeme faktu, že přírustek Wienerova procesu za jednotkový čas má normální rozdělení $N(0, 1)$. Tímto způsobem spojitý proces přirozeně diskretizujeme a vytvoříme tak časovou řadu.



Obrázek 17.3: Simulace Wienerova procesu.

Wiener.m: Simulace Wienerova procesu

```
figure(1);
simN = 1000;
for i = 1:6
  simZ = zeros(1, simN);
  for t = 2:simN
    simZ(t) = simZ(t - 1) + random('norm', 0, 1);
  end
  subplot(2, 3, i);
  plot(1:simN, simZ, 'm');
  title('Wiener Process Simulation');
  xlabel('Time');
  ylabel('Price');
end
```

Ceny sledované akcie budeme modelovat jako náhodný proces X_t. Dále, budeme předpokládat, že se negativní logaritmické ceny chovají jako Wienerův proces s driftem μ a směrodatnou odchylkou σ.

$$-\ln X_t = \mu t + \sigma W_t.$$

Denní logaritmické ztráty jsou pak dány procesem

$$Z_t = -(\ln X_t - \ln X_{t-1}) = \mu + \sigma(W_t - W_{t-1}),$$

který má rozdělení $Z_t \sim N(\mu, \sigma^2)$. Celý model tedy spočívá v tom, že denní logaritmické ztráty mají normální rozdělení s neznámou střední hodnotou μ a neznámým rozptylem σ^2.

Prvním krokem bude tyto dva parametry odhadnout. To uděláme jednoduše pomocí výběrového průměru $\hat{\mu}$ a výběrového rozptylu $\hat{\sigma}^2$ historických dat. Dále už nám zbývá vypočítat kvantil normálního rozdělení $N(\hat{\mu}, \hat{\sigma}^2)$. Celý výpočet Value-at-Risk parametrickou metodou si opět naprogramujeme jako funkci.

ValueAtRisk.m: Výpočet parametrického VaR

```
27  function VaRPar = myVaRParametric(Z)
28    mu = mean(Z);
29    sigma = std(Z);
30    VaRPar = icdf('norm', 0.95, mu, sigma);
31  end
```

17.4 Porovnání konfidenčních intervalů

Nyní přirozeně vyvstává otázka, jestli je lepší pro odhad Value-at-Risk používat historickou metodu nebo parametrickou metodu. Triviální odpověď neexistuje, ale aspoň se v této části pokusíme porovnat intervaly spolehlivosti obou metod podle počtu pozorování (délky historie). Intervaly spolehlivosti vypočteme u obou metod pomocí tzv. bootstrapové metody.

Bootstrapová metoda

Boostrapová metoda slouží ke zjištění přesnosti odhadu (např. pomocí vychýlení, rozptylu nebo intervalu spolehlivosti). Spočívá ve vytvoření velkého počtu náhodných vzorků. Bootstrapová metoda vytvoří z původního vzorku několik nových vzorků pomocí náhodného výběru s opakováním. Na každém vzorku se provede odhad, výsledkem je tedy několik realizací odhadů, ze kterých se pak může spočítat výběrový průměr, výběrový rozptyl odhadu nebo jiné charakteristiky. Výhodou bootstrapové metody je její jednoduchost i v případě složitých estimátorů, u kterých by analytické odvození např. intervalů spolehlivosti bylo příliš náročné.

Bootstrapovou metodu pro historický a empirický Value-at-Risk si naprogramujeme jako funkci. Argumenty funkce budou původní vzorek, velikost jednoho nového vzorku a počet nových vzorků. Každý z nových vzorků je pak vygenerován stejným způsobem: Z původního vzorku se náhodně vyberou některá pozorování s tím, že se některá pozorování mohou i opakovat. Toho docílíme tak, že si nejdřív vygenerujeme vektor indexů pozorování, ve kterém každá složka je simulace čísla z diskrétního rovnoměrného rozdělení od 1 do n. Díky možnosti opakování může být nový vzorek dokonce obsahovat více pozorování než ten původní. Na každém bootstrapovém vzorku pak spočítame Value-at-Risk oběma metodami.

ValueAtRisk.m: Bootsrapová metoda

```
33  function [bootEmp bootPar] = myBootVaR(Z, sample, rep)
34    bootEmp = zeros(rep, 1);
35    bootPar = zeros(rep, 1);
36    for i = 1:rep
37      bootIndex = random('unid', length(Z), [sample 1]);
38      bootZ = Z(bootIndex, 1);
39      bootEmp(i, 1) = myVaREmpiric(bootZ);
40      bootPar(i, 1) = myVaRParametric(bootZ);
41    end
42  end
```

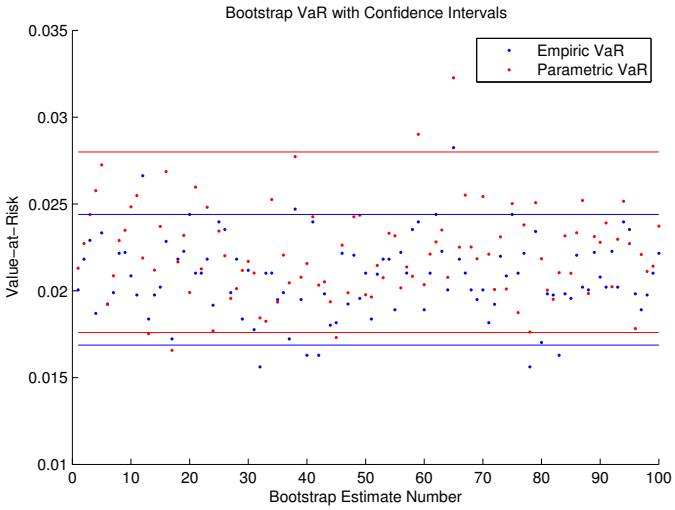
Pomocí bootstrapové metody si 1000 krát vypočteme historický i parametrický Value-at-Risk z náhodných vzorků o velikosti 200 pozorování. Z těchto 1000 hodnot pak pomocí funkce `quantile` vypočteme 95%-intervaly spolehlivosti. (Pozor, zde funkce `quantile` nemá vůbec co dočinění s kvantilem ve smyslu Value-at-Risk, používáme ji pro interval spolehlivosti, který je rovněž založen na kvantilech.) Historický i parametrický interval spolehlivosti spolu s prvními 100 hodnotami obou typů Value-at-Risku si zakreslíme do grafu.

ValueAtRisk.m: Graf odhadů VaR bootstrapovou metodou

```
44  figure(2);
45  [bootEmp bootPar] = myBootVaR(Z, n, 10e3);
46  confEmp = quantile(bootEmp, [0.025 0.975]);
47  confPar = quantile(bootPar, [0.025 0.975]);
48  title('Bootstrap VaR with Confidence Intervals');
49  xlabel('Bootstrap Estimate Number');
50  ylabel('Value-at-Risk');
51  hold on;
52  showNum = 1e2; % ukaz pouze prvnich 100 hodnot
53  plot(bootEmp(1:showNum, 1), '.b');
54  plot(bootPar(1:showNum, 1), '.r');
55  plot([1 showNum], [confEmp(1) confEmp(1)], '-b');
56  plot([1 showNum], [confEmp(2) confEmp(2)], '-b');
57  plot([1 showNum], [confPar(1) confPar(1)], '-r');
58  plot([1 showNum], [confPar(2) confPar(2)], '-r');
59  legend('Empiric VaR', 'Parametric VaR');
```

Dále se podíváme, jak budou intervaly spolehlivosti záviset na velikosti bootstrapových vzorků. Dá se očekávat, že s většími vzorky budou intervaly užší. Otázkou ale je, jaké budou rozdíly mezi intervaly historického a parametrického Value-at-Risku.

Intervaly spolehlivosti si spočteme pomocí `for` cyklu pro různé velikosti bootstrapových vzorků z mřížky od 20 do 200 s krokem 20. Šířky intervalů historického Value-at-Risku v závislosti na velikosti vzorku si zakreslíme do levého grafu, šířky intervalů parametrického Value-at-Risk do pravého grafu.

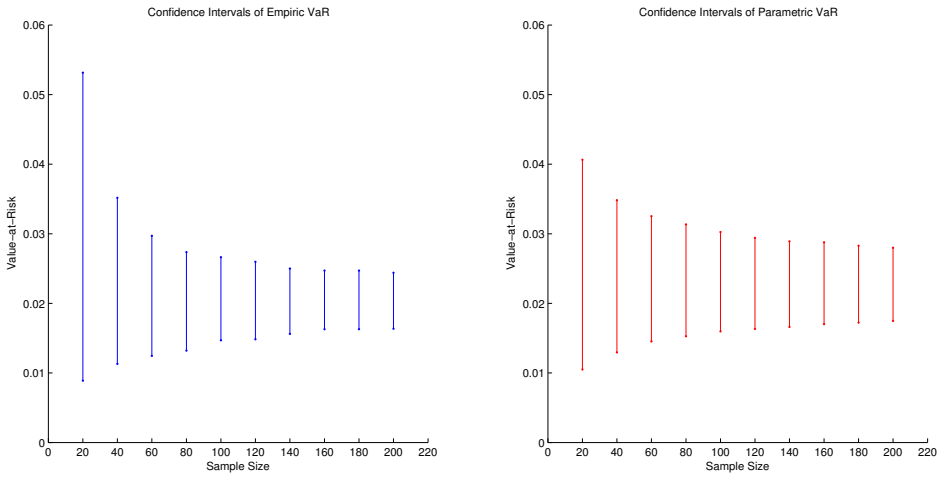


Obrázek 17.4: Odhady VaR bootstrapovou metodou.

ValueAtRisk.m: Graf konfidenčních intervalů VaR pro různé velikosti vzorků

```
figure(3);
sFrom = 20;
  sBy = 20;
  sTo = 200;
for s = sFrom:sBy:sTo
  [bootEmp bootPar] = myBootVaR(Z, s, 10e3);
  confEmp = quantile(bootEmp, [0.025 0.975]);
  confPar = quantile(bootPar, [0.025 0.975]);
  subplot(1, 2, 1);
  hold on;
  plot([s s], confEmp, '.-b');
  axis([sFrom - sBy, sTo + sBy, 0, 0.06]);
  title('Confidence Intervals of Empiric VaR');
  xlabel('Sample Size');
  ylabel('Value-at-Risk');
  subplot(1, 2, 2);
  hold on;
  plot([s s], confPar, '.-r');
  axis([sFrom - sBy, sTo + sBy, 0, 0.06]);
  title('Confidence Intervals of Parametric VaR');
  xlabel('Sample Size');
  ylabel('Value-at-Risk');
end
```

Vidíme, že pro malý počet pozorování je interval parametrického Value-atRisku užší. To je způsobeno tím, že parametrická metoda si nedostatek informace z malého počtu pozorování „nahrazuje“ předpokladem o normálním rozdělení. Pro dostatečně velký počet pozorování se historická metoda jeví jako stabilnější. Předpoklad o normálním rozdělení parametrické metody může být navíc zcestný a odhad



Obrázek 17.5: Konfidenční intervaly VaR pro různé velikosti vzorků.

Value-at-Risk tak může být vychýlený.

Doporučená literatura

- Cipra, T. 2014. *Finanční ekonometrie*. Ekopress. ISBN 978-80-86929-93-4. https://www.ekopress.cz/titdetail.php?tid=30238
- Francis, J. C., Kim, D. 2013. *Modern Portfolio Theory: Foundations, Analysis, and New Developments*. Wiley. ISBN 978-1-118-37052-0. https://www.wiley.com/en-us/Modern+Portfolio+Theory{%}3A+Foundations{%}2C+Analysis{%}2C+and+New+Developments{%}2C+{%}2B+Website-p-x000610741

Kapitola 18

Chaotické chování logistické rovnice

V této kapitole si představíme logistickou diferenční rovnici, která i přes svojí jednoduchost vykazuje komplexní chování. Analýza této rovnice spadá do tzv. teorie chaosu.

Klíčová slova: *teorie chaosu, logistická diferenční rovnice, bifurkační diagram.*

18.1 Logistická diferenční rovnice

Logistická diferenční rovnice je definovaná předpisem

$$x_{t+1} = f(x_t, \lambda) = \lambda x_t(1 - x_t), \qquad \text{pro} \quad x_t \in [0, 1], \quad t = 1, 2, \ldots. \tag{18.1}$$

Protože x_t může nabývat pouze hodnot z intervalu $[0, 1]$, diferenční rovnice je definována jenom pro některá λ. Zřejmě musí být λ nezáporné číslo, jinak by na pravé straně bylo záporné číslo a x_t by pak neleželo v $[0, 1]$. Hodnota λ je omezená i shora. Spočítejme si maximum funkce $f(x_t, \lambda)$. Podmínka prvního řádu má tvar

$$\frac{\partial f}{\partial x_t} = \lambda - 2\lambda x = 0.\text{a}$$

Funkce nabývá svého maxima v bodě $x = \frac{1}{2}$, ve kterém nabývá hodnoty $f(\frac{1}{2}, \lambda) = \frac{\lambda}{4}$. A aby $x_t \leq 1$, musí být $0 \leq \lambda \leq 4$.

Poznámka

Logistická rovnice se použije např. v následujícím příkladě. Uvažujme spotřebitele, který spotřebovává dva typy zboží a maximalizuje svůj užitek U. Spotřebu prvního zboží označme x, jeho cenu p a spotřebu druhého zboží y s cenou q. Spotřebitel je omezen rozpočtem m. Dále označme a preferenční parametr mezi oběma typy zboží. Předpokládáme, že spotřebitel se řídí užitkovou funkcí $U = x^a y^{1-a}$. Úloha maximalizace užitku má tvar

$$\begin{aligned} \max_{x,y} \quad & x^a y^{1-a} \\ s.t. \quad & px + qy \leq m, \\ & x \geq 0, \\ & y \geq 0. \end{aligned}$$

Řešením této úlohy je

$$x = \frac{m}{p}a, \qquad y = \frac{m}{q}(1-a).$$

Odvození ponecháme jako cvičení. Dále do modelu přidáme dynamiku. Budeme uvažovat, že parametr a závisí na minulých rozhodnutích podle vztahu

$$a_{t+1} = bx_t y_t.$$

Můžeme pak vyjádřit

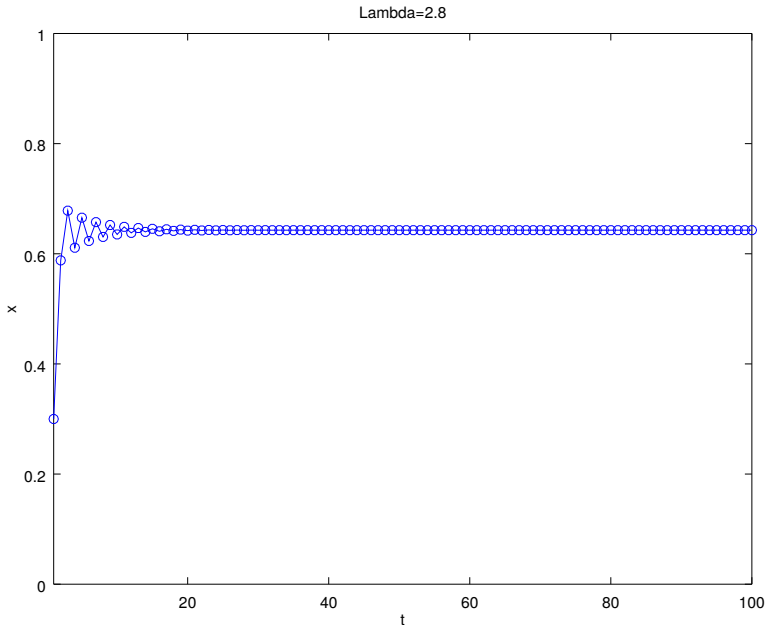
$$x_{t+1} = \frac{mb}{q} x_t (1 - px_t),$$

což po normalizaci cen $p = q = 1$ vede na logistickou diferenční rovnici (18.1).

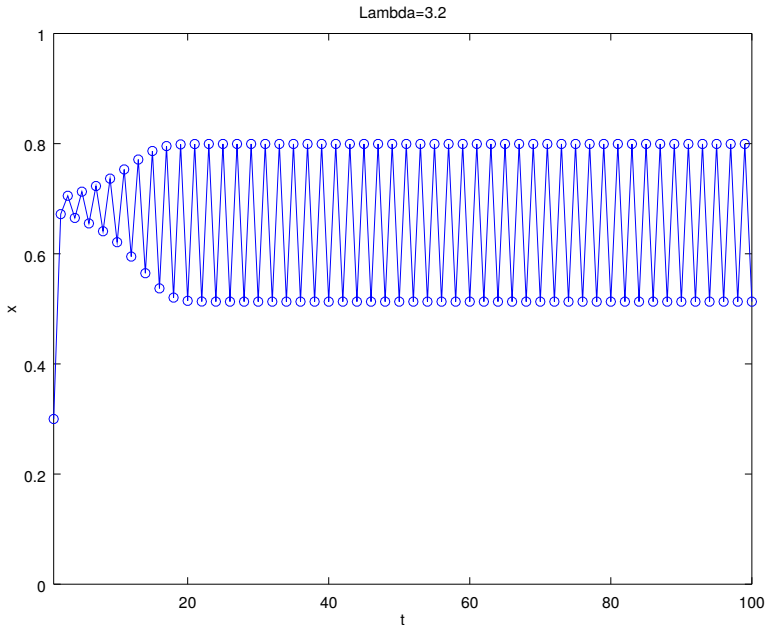
Podíváme se, jak bude vypadat trajektorie logistické diferenční rovnice pro různá λ. Jak brzy uvidíme, pro různá λ se bude logistická diferenční rovnice chovat odlišně. V MATLABu si vykreslíme prvních 100 hodnot s počáteční hodnotou $x_1 = 0.3$. Pro nějaké λ to můžeme udělat pomocí `for` cyklu, který vypočte hodnotu x_t pro časy $t = 2, \ldots, 100$. Celý výpočet ještě „zabalíme“ do dalšího `for` cyklu, který projde všechny možné hodnoty λ z mřížky od 0 do 4 s krokem 0.01.

Chaos.m:

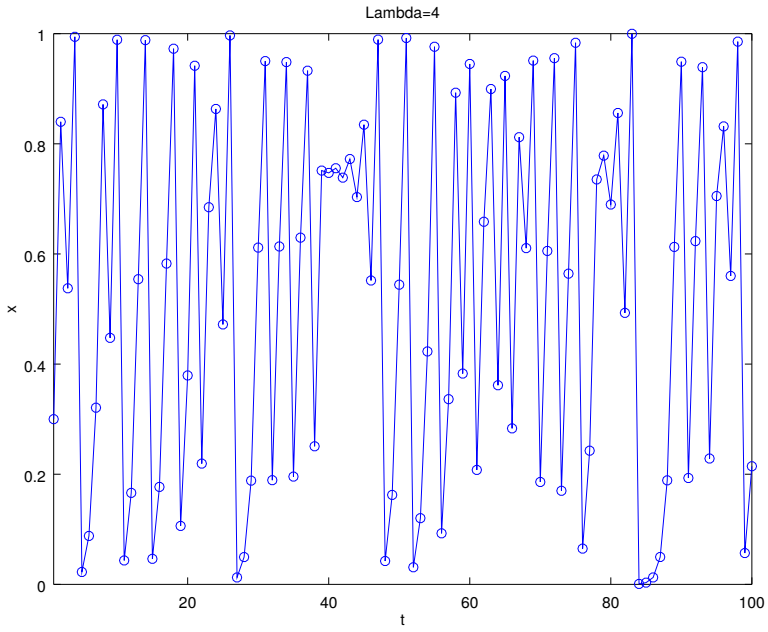
```
figure;
xStart = 0.30
for lambda = 0:0.01:4
  X(1) = xStart;
  for t = 2:100
    X(t) = lambda * X(t - 1) * (1 - X(t - 1));
  end
  plot(X, '-ob');
  axis([1 100 0 1]);
  title(strcat('Lambda=', num2str(lambda)));
  xlabel('t');
  ylabel('x');
  hold off;
  pause(0.1);
end
```



Obrázek 18.1: Trajektorie x_t pro $\lambda = 2.8$. Proces konverguje.



Obrázek 18.2: Trajektorie x_t pro $\lambda = 3.2$. V procesu se objevují 2-cykly.



Obrázek 18.3: Trajektorie x_t pro $\lambda = 4.0$. Proces se chová chaoticky.

Vidíme, že pro nějaké hodnoty λ se proces x_t ustálí na nějaké konstantě jako na obrázku 18.1. Pro další hodnoty se x_t chová cyklicky (objevují se zde dvojcykly jako na obrázku 18.2 i čtyřcykly). A pro některé hodnoty se x_t chová zdánlivě náhodně (v rovnici ovšem žádná náhoda nevystupuje, vše je deterministické), nepředvídatelně, chaoticky jako na obrázku 18.3. Odvětví matematiky, které se zabývá dynamickými systémy, kterou jsou těžko předvídatelné a velmi citlivé na počáteční podmínky, se nazývá teorie chaosu.

18.2 Rovnovážný stav

Budeme hledat rovnovážný stav, tedy takový stav, do kterého když se proces dostane, už tam zůstane, tedy $x_{t+1} = x_t = x^*$. Přesněji, hledáme λ, pro které existuje $\lim_{t\to\inf} x_t$. Řešme rovnici

$$\begin{aligned} x^* &= \lambda x^*(1 - x^*) \\ \lambda x^{*2} + (1 - \lambda)x^* &= 0 \\ x^*\Big(\lambda x^* + (1 - \lambda)\Big) &= 0, \end{aligned}$$

která má řešení

$$x^* = 0 \qquad \text{nebo} \qquad x^* = \frac{\lambda - 1}{\lambda}.$$

Konvergenci k bodu 0 pozorujeme pro $0 \leq \lambda \leq 1$, konvergenci k bodu $\frac{\lambda-1}{\lambda}$ pro $1 \leq \lambda \leq 3$.

18.3 Bifurkační diagram

Dále si vykreslíme tzv. bifurkační diagram. Na horizontální ose budeme mít hodnoty parametru λ a na vertikální ose hodnoty x_t. Bifurkační diagram vykresluje ty hodnoty, které systém asymptoticky navštíví. Např. pokud se systém pro dané λ dostane do bodu 0, ve kterém podle předchozí sekce již zůstane, bifurkační diagram vykreslí pro toto λ pouze hodnotu 0. Chaotické chování se naopak vyznačuje velkým

množstvím těchto tzv. hromadných bodů, které pak na obrázku působí jako souvislá plocha. Hromadný bod posloupnosti je takový bod, ke kterému konverguje nějaká vybraná podposloupnost. Naším cílem je vykreslit všechny hromadné body posloupnosti x_t v závislosti na λ.

Bifurkační diagram si v MATLABu vykreslíme následovně. Pro každé λ si spočteme trajektorii systému pro velký počet kroků, v našem případě pro $t = 2, \ldots, 500$. Prvních 200 kroků „zahodíme“ (předpokládáme, že sytém se ještě neustálil). Zbylých 300 hodnot x_t, $t = 201, \ldots, 500$ si pak vykreslíme pro dané λ. Dostaneme tak přibližně ty hodnoty, které proces asymptoticky navštěvuje.

`ChaosBifur.m`: Vykreslení bifurkačního diagramu

```
figure;
xStart = 0.30
for lambda = 0:0.01:4
  X(1) = xStart;
  for t = 2:500
    X(t) = lambda * X(t - 1) * (1 - X(t - 1));
  end
  plot(lambda * ones(400, 1), X(101:500), '.b');
  hold on;
  axis([0 4 0 1]);
  xlabel('lambda');
  ylabel('x');
end
```

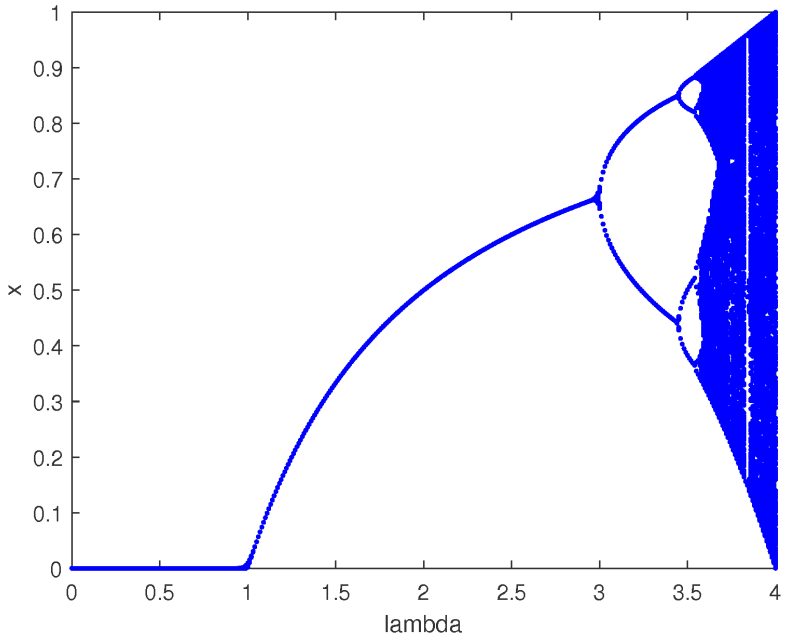
Poznámka

Tento kód rozhodně není ideálním způsobem, jak bifurkační diagram vykreslit. Vyžaduje totiž velké množství bodů, které zatěžují vykreslovací prostředí. Pro naše účely ale tento způsob postačí.

Pomocí tohoto grafu můžeme určit, kdy x_t směřuje k fixnímu bodu, kdy se objevují cykly (např. vidíme, že pro hodnoty λ od 3.0 do přibližně 3.4 se proces chová jako dvojcyklus) a kdy se rovnice chová chaoticky (např. hodnoty λ mezi 3.7 a 4.0 převážně vedou k chaotickému chování, ale pro několik hodnot λ z tohoto intervalu chaotické chování zmizí a objeví se cykly).

18.4 Citlivost na počáteční hodnotu

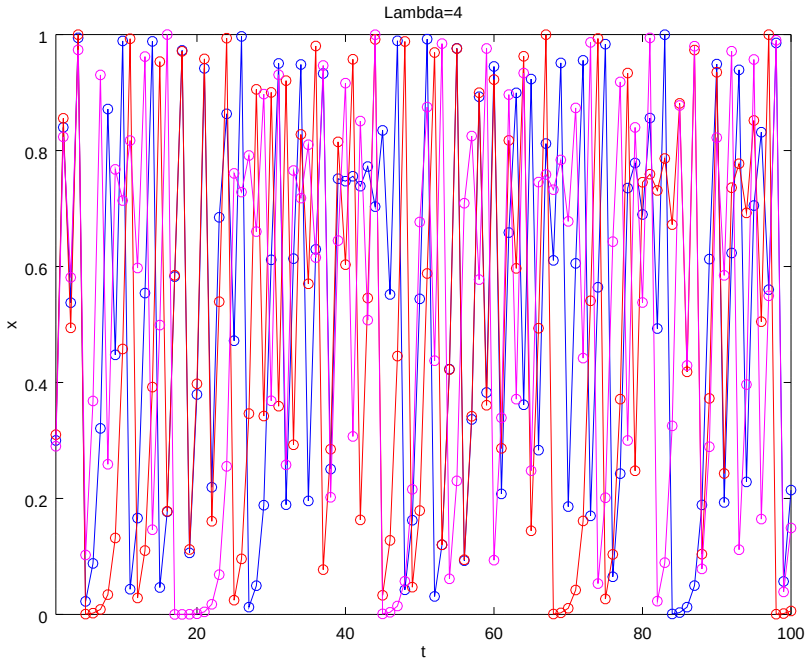
Chaotické dynamické systémy se vyznačují velkou citlivostí na počáteční podmínky. Podívejme se, jak bude vypadat trajektorie procesu pro počáteční hodnoty 0.29, 0.30 a 0.31. Použijeme podobný skript jako v úvodu této kapitoly, jen do něj přidáme procesy `y` a `z`.



Obrázek 18.4: Bifurkační diagram logistické diferenční rovnice.

ChaosMulti.m: Bifurkační diagram.

```
figure;
xStart = 0.30
yStart = 0.31
zStart = 0.29
for lambda = 0:0.01:4
  X(1) = xStart;
  Y(1) = yStart;
  Z(1) = zStart;
  for t = 2:100
    X(t) = lambda * X(t - 1) * (1 - X(t - 1));
    Y(t) = lambda * Y(t - 1) * (1 - Y(t - 1));
    Z(t) = lambda * Z(t - 1) * (1 - Z(t - 1));
  end
  plot(X, '-ob');
  hold on;
  plot(Y, '-or');
  plot(Z, '-om');
  axis([1 100 0 1]);
  title(strcat('Lambda=', num2str(lambda)));
  xlabel('t');
  ylabel('x');
  hold off;
  pause(0.1);
end
```



Obrázek 18.5: Trajektorie x_t při počátečních hodnotách 0.29, 0.30 a 0.31.

Vidíme, že pro λ vedoucí k chaotickému chování jsou trajektorie procesu naprosto odlišné. Při reálných aplikacích jsou známé hodnoty zatíženy nějakou chybou (ať už chybou měření nebo zaokrouhlovací chybou způsobenou počítačem) a chaotické chování tak značně komplikuje predikci v dynamických systémech.

Doporučená literatura

- Shone, R. 2002. *Economic Dynamics: Phase Diagrams and Their Economic Application*. Cambridge University Press. ISBN 978-0-521-81684-7. https://doi.org/10.1017/CBO9781139165020
- Strogatz, S. H. 2015. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. CRC Press. ISBN 978-0-8133-4910-7. https://doi.org/10.1201/9780429492563

Příloha

Použité značení

V tomto dodatku je popsáno značení běžně používané v matematických textech.

Písmena řecké abecedy

A	α	alfa	I	ι	ióta	P	ρ	ró
B	β	beta	K	κ	kappa	Σ	σ	sigma
Γ	γ	gama	Λ	λ	lambda	T	τ	tau
Δ	δ	delta	M	μ	mí	Υ	υ	ypsilon
E	ε	epsilon	N	ν	ný	Φ	ϕ	fi
Z	ζ	zéta	Ξ	ξ	ksí	X	χ	chí
H	η	éta	O	o	omikron	Ψ	ψ	psí
Θ	θ	théta	Π	π	pí	Ω	ω	omega

Množiny čísel

$\mathbb{N}$	přirozená čísla	$\mathbb{N}_0$	přirozená čísla včetně nuly
$\mathbb{Z}$	celá čísla	$\mathbb{Q}$	racionální čísla
$\mathbb{R}$	reálná čísla	$\mathbb{C}$	komplexní čísla

Příloha

B

Základní operace a funkce v Matlabu

Mnoho zde popsaných funkcí má daleko širší využití a více možností zadání argumentů, než je v tomto dodatku zmíněno. Pro kompletní popis funkcí je možno využít nápovědu `help` v příkazové řádce Matlabu nebo oficiální dokumentaci na adrese `http://www.mathworks.com/help/matlab/`.

Vektorové a maticové operace

`A+B`	sečtení matic
`c*A`	násobení matice skalárem
`A*B`	maticové násobení
`A.*B`	násobení matic po složkách
`A^c`	maticové mocnění
`A.^`	mocnění složek matice
`A'`	transpozice matice
`a:b:c`	aritmetická posloupnost od `a` do `c` s velikostí kroku `b` v řádkovém vektoru

Zaokrouhlovací funkce

`round(x)`	zaokrouhlení `x` na celé číslo
`floor(x)`	zaokrouhlení `x` dolů
`ceil(x)`	zaokrouhlení `x` nahoru

Vektorové funkce

`sum(x)`	součet prvků vektoru `x`
`prod(x)`	součin prvků vektoru `x`
`min(x)`	nejmenší prvek vektoru `x`
`max(x)`	největší prvek vektoru `x`
`diff(x,n)`	diference `n`-tého řádu vektoru `x`

Maticové funkce

`size(A)`	rozměry matice `A`
`zeros(m,n)`	matice samých nul o `m` řádcích a `n` sloupcích
`ones(m,n)`	matice samých jedniček o `m` řádcích a `n` sloupcích
`eye(n)`	jednotková čtvercová matice řádu `n`
`diag(x)`	čtvercová matice s diagonálou vektoru `x`
`eig(A)`	vlastní čísla matice `A`

Matematické funkce

`abs(x)`	absolutní hodnota `x`
`exp(x)`	exponenciála `x`
`log(x)`	přirozený logaritmus `x`
`sin(x)`	sinus `x`
`cos(x)`	cosinus `x`
`tan(x)`	tangens `x`

Pravděpodobnostní rozdělení

`cdf(name,x,a,b)`	distribuční funkce rozdělení `name` v bodech `x` s parametry `a` a `b`
`pdf(name,x,a,b)`	hustota rozdělení `name` v bodech `x` s parametry `a` a `b`
`icdf(name,p,a,b)`	inverzní funkce k distribuční funkci rozdělení `name` v pravděpodobnostech `p` s parametry `a` a `b`
`random(name,a,b,[m,n])`	matice s `m` řádky a `n` sloupci náhodných čísel z rozdělení `name` s parametry `a` a `b`

Statistické funkce

`mean(x)`	průměr vektoru `x`
`median(x)`	medián vektoru `x`
`var(x)`	rozptyl vektoru `x`
`std(x)`	směrodatná odchylka vektoru `x`
`cov(x)`	matice kovariancí sloupců matice `X`
`corrcoeff(X)`	matice korelací sloupců matice `X`

Optimalizační funkce

`linprog(c,A,b)`	řešitel lineárního programování
`quadprog(H,f,A,b,Aeq,beq)`	řešitel kvadratického programování

Načítání dat

`csvread('file.csv')`	načtení souboru `file.csv`
`xlsread('file.xls')`	načtení souboru `file.xls`

Funkce `plot` a práce s grafy

`figure`	otevření nového okna pro graf
`subplot(m,n,p)`	rozdělení okna grafu na `m` řádků a `n` sloupců a výběr `p`-tého okénka
`plot(x,y)`	vykreslení grafu dat s horizontálními souřadnicemi `x` a vertikálními souřadnicemi `y`
`plot3(x,y,z)`	vykreslení trojrozměrného grafu se souřadnicemi `x`, `y` a `z`
`axis([a,b,c,d])`	nastavení horizontální osy na rozsah od `a` do `b` a vertikální osy od `c` do `d`
`title('t')`	nadpis grafu
`xlabel('t')` / `ylabel('t')`	popis horizontální / vertikální osy
`legend('t1','t2',...)`	vloží legendu s popisem jednotlivých křivek
`hold on` / `hold off`	zapnutí / vypnutí dalšího zakreslování do grafu
`grid on` / `grid off`	zapnutí / vypnutí zobrazování mřížky v grafu
`histogram(x,n)`	histogram dat `x` v `n` sloupcích
`histfit(x,n,name)`	histogram dat `x` v `n` sloupcích s přidanou hustotou rozdělení `name`

Vizualizace 3D dat

`plot3(x, y, z)`	vykreslení trojrozměrného grafu se souřadnicemi `x`, `y` a `z`
`surf(X, Y, Z)`	vykreslení trojrozměrného povrchu se souřadnicemi `X`, a `Y` a výškou `Z`
`contour(X, Y, Z)`	vykreslení dvojrozměrných vrstevnic se souřadnicemi `X`, a `Y` a výškou `Z`
`meshgrid(x, y)`	tvorba dvojrozměrné mřížky dané kombinacemi vektorů `x` a `y`

Konverze typu proměnných

`int2str(x)`	zaokrouhlení čísla `x` a převedení na řetězec
`num2str(x)`	převedení čísla `x` na řetězec
`mat2str(A)`	převedení matice `A` na řetězec jejího Matlab zápisu

Nápověda

`help`	obsah nápovědy
`help name`	popis funkce `name`

Ostatní funkce

`pause(x)`	zastavení programu na `x` vteřin
`break`	přerušení průběhu cyklu

Příloha C

Seznam teoretický pojmů

Příloha D

Seznam MATLAB funkcí

Příloha E

Seznam skriptů