TOMÁŠ RAČEK

ZÁPISKY K PŘEDNÁŠKÁM

Obsah

Úvodní slovo

Tento text vznikl jako textová opora pro prvních několik témat předmětu PřF:C1471. Původně sloužil pouze jako podklad pro přípravu přednášek, nicméně v souvislosti s distanční výukou, která byla v současnosti zavedena, jsem se pokusil jej upravit do podoby vhodné pro samostudium.

Jelikož napsat studijní text je záležitost na minimálně několik měsíců, nelze očekávat, že tento materiál vytvořený za několik dnů bude úplně kvalitativně na úrovni vysokoškolských skript. Při jeho hodnocení prosím o shovívavost, pokusím se jej průběžně aktualizovat a opravovat.

Obsahem tento text pokrývá část o problémech spojených s reprezentací reálných čísel v počítači, kapitoly k lineární algebře a úvod k funkcím více proměnných. Při jeho přípravě jsem čerpal zejména z následujících zdrojů:

- Reprezentace reálných čísel: předmět FI:PA081 Programování numerických výpočtů
- Lineární algebra: vynikající kurz 18.06 Linear Algebra na MIT

Protože je užitečné v některých příkladech využít počítač, text obsahuje i úryvky kódu v jazyce Python. Umět programovat není pro pochopení jednotlivých příkladů povětšinou nutné, nicméně ti, které by to zajímalo, mohou tyto útržky použít pro další studium.

Naleznete-li v textu chybu, prosím o její nahlášení, abych ji mohl v příští revizi opravit. Díky.

Tomáš Raček

Poslední aktualizace: 4. června 2020

Problémy s reprezentací reálných čísel

Pro velkou část výpočetních úloh lze s úspěchem využít počítač. Výkon dnešních osobních počítačů je tak velký, že nám umožňuje řešit problémy, jejichž řešení by bylo ještě před několika desítkami let nemyslitelné.

Výpočty na počítači mají však jistá specifika – vlastnosti, o kterých musíme vědět, abychom nebyli nepříjemně překvapeni. Co tím myslíme si povíme v této kapitole.

Už jen dnešní mobilní telefony jsou mnohonásobně výkonnější než řídící počítač Apolla 11, které přistálo v roce 1969 na Měsíci.

Motivační příklady

Uveďme dva jednoduché příklady, na kterých demonstrujeme neočekávané chování při výpočtech s reálnými čísly.

Určitý integrál

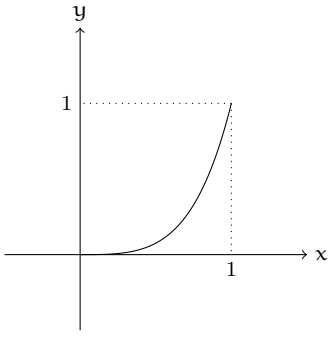
Začněme příkladem z matematické analýzy. Určete hodnotu integrálu:

$$\int_0^1 x^{30} e^{x-1}\,dx$$

Tento příklad je typickým zástupcem pro metodu *per partes*. Polynomu lze každou aplikací snižovat stupeň o jedna (člen e^{x-1} zůstává samozřejmě nezměněn). Nicméně třicetkrát aplikovat tuto metodu je možná až příliš...

Dříve než se pustíme do alternativního řešení, zamysleme se nad tím, jaký výsledek můžeme očekávat. Na intervalu od 0 do 1 jsou obě funkce (x^{30} i e^{x-1}) nezáporné, rostoucí a konvexní, totéž můžeme tedy očekávat od jejich součinu. Dosadíme-li meze pro osu x, získáme tak i meze pro osu y:

$$0^{30} e^{0-1} = 0$$
$$1^{30} e^{1-1} = 1$$



Obrázek 1: Náčrt hledané funkce

Odtud vidíme, že obsah pod křivkou danou naší funkcí je celý obsažen v jednotkovém čtverci, hledaný výsledek tedy bude určitě číslo mezi 0 a 1 (přesněji mezi 0 a 1/2, jak můžeme vidět z náčrtku).

Když víme, jaký výsledek můžeme očekávat, přistoupíme k řešení. Zobecníme příklad (pro $n = 30$) a označme integrál J_n:

$$J_n = \int_0^1 x^n e^{x-1}\,dx$$

Řešení skrze *per partes*, první aplikace:

Připomeňme metodu *per partes* obecně:

$$\int f'(x)g(x)\,dx = f(x)g(x) - \int f(x)g'(x)\,dx$$

Konkrétně zde:

$$f'(x) = e^{x-1}$$
$$g(x) = x^n$$

$$J_n = \int_0^1 x^n e^{x-1}\,dx = \left[x^n e^{x-1}\right]_0^1 - n\int_0^1 x^{n-1}e^{x-1}\,dx =$$
$$J_n = 1^n e^0 - n\int_0^1 x^{n-1}e^{x-1}\,dx$$
$$J_n = 1 - n\int_0^1 x^{n-1}e^{x-1}\,dx$$

Zde si můžeme všimnout, že nově vzniklý integrál je velmi podobný našemu původnímu. Dostáváme rekurentní vztah:

$$\mathbf{J_n = 1 - nJ_{n-1}}$$

Určeme ještě hodnotu pro triviální případ pro $n = 0$:

$$\mathbf{J_0} = \int_0^1 x^0 e^{x-1}\,dx = \int_0^1 e^{x-1}\,dx = \left[e^{x-1}\right]_0^1 = e^0 - e^{-1} = \mathbf{1 - 1/e}$$

J_{30} řešme postupně od J_1:

$$\mathbf{J_0 = 1 - 1/e}$$
$$\mathbf{J_1} = 1 - 1\cdot J_0 = 1 - (1 - 1/e) = \mathbf{1/e}$$
$$\mathbf{J_2} = 1 - 2\cdot J_1 = 1 - 2(1/e) = \mathbf{1 - 2/e}$$
$$\vdots$$
$$\mathbf{J_{30}} =$$

Abychom tohle vše nemuseli počítat ručně, pomůžeme si krátkým programem:

```
>>> from math import e
>>> J = {}
>>> J[0] = 1 - 1/e
>>> for n in range(1, 31):
...     J[n] = 1 - n * J[n - 1]
>>> J[30]
-3296762455608386.5
```

Připomeňme, že funkce *range*(*m*, *n*) vrací hodnoty $m, m+1, \ldots, n-1$.

Výsledek je nicméně zjevně nesmyslný – z počáteční analýzy víme, že musíme získat číslo mezi 0 a 1. Počítá tedy Python špatně?

Jednoduchá porovnání

Zkusme ještě několik jednoduchých porovnání v interpretu Pythonu:

```
>>> 1 + 1 == 2
True
>>> 0.1 + 0.1 == 0.3
False
>>> 0.1 + 0.1 == 0.2
True
>>> 0.1 + 0.2 == 0.3
False
```

Opět výsledek posledního porovnání může být překvapivý. Nejedná se o chybu Pythonu. V jiném programovacím jazyce bychom obdrželi podobné výsledky. Abychom porozuměli, co se děje, musíme si říct něco o tom, jak jsou čísla reprezentována v počítači.

Reprezentace čísel v počítači

Protože celých nebo reálných čísel je nekonečně mnoho, nelze uložit libovolné číslo v konečné paměti. Omezujeme se tedy na reprezentace, které jsou užitečné pro velké množství použití. Číslo je z hlediska programátora uloženo jako proměnná (nebo konstanta) vhodného datového typu. To mimo jiné definuje, kolik místa v paměti zabírá.

Celá a reálná čísla jsou reprezentována v počítači odlišně, zaměřme se nejprve na jednodušší případ.

Reprezentace celých čísel

Jak asi víme, počítač používá pro reprezentaci čísel dvojkovou soustavu. Zopakujme si nejdříve, jak lze rozepsat číslo v soustavě desítkové:

$$101 = 1 \cdot 10^2 + 0 \cdot 10^1 + 1 \cdot 10^0$$

Úplně analogicky to lze provést pro jiný základ, máme-li číslo v jiné soustavě. Uveďme jednoduchý příklad korespondence mezi dvojkovou a standardní desítkovou soustavou[1]:

$$(101)_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 4 + 0 + 1 = 5$$

Pro uložení celého čísla se v počítači běžně používají 4 byty, tedy 32 bitů (takový typ se označuje třeba v jazycích odvozených z jazyka C jako *int*). Jelikož každá z číslic může nabývat hodnot 0 nebo 1, dostáváme 2^{32} různých kombinací[2].

$$2^{32} = 4294967296$$

Omezíme-li se na kladná čísla (*unsigned int*), jsme se tedy schopni reprezentovat čísla v rozmezí 0 až 4294967295. Všechny standardní operace (sčítání, odečítání, násobení, celočíselné dělení) pak probíhají shodně jako v desítkové soustavě. Pokud potřebujeme i záporná čísla,

[1] Dolní index u čísla vyjadřuje, v jaké soustavě je číslo zapsáno. U desítkové soustavy ho nepíšeme, pokud tuto skutečnost nechceme v textu explicitně odlišit.

[2] Z kombinatoriky se jedná samozřejmě o variaci s opakováním, kde vybíráme ze dvou prvků 32krát.

je situace maličko složitější. Spokojíme se tedy pouze s faktem, že rozsah je zhruba poloviční (v obou směrech).

Pokud bychom ovšem chtěli například sečíst dvě čísla, jejichž součet by byl vyšší než je zmíněné maximum, došlo by k tzv. *přetečení*, což můžeme ilustrovat to můžeme na jednoduchém příkladu:

$$4294967294 + 1 = 4294967295$$
$$4294967294 + 2 = 0$$
$$4294967294 + 3 = 1$$

Pokud je pro nás daný rozsah dostatečný[3], výsledky jsou přesné.

[3] Některé jazyky mají ještě 8B typ *long*, který už zvládne uložit 18446744073709551616 různých hodnot. Jazyky, jako je Python, svůj celočíselný typ mají omezen pouze velikostí dostupné paměti, tam je tento problém bezpředmětný.

Reprezentace reálných čísel

U reálných čísel je situace výrazně složitější, vždyť už jen mezi 0 a 1 máme nekonečně mnoho reálných čísel. Zjevně nám tedy nepomůže se omezit na libovolně malý rozsah, v rámci kterého bychom reprezentovali čísla přesně.

Pojďme se podívat, jakým způsobem jsou tedy reálná čísla v počítači uložena. Typickým zástupcem může být 4bytový typ *float* programovacího jazyka C. Opět máme k dispozici 2^{32} různých hodnot, nicméně zde bude jejich význam jiný. Reprezentace používá ekvivalent tzv. vědecké notace; pro ukázku zvolme dva příklady v desítkové soustavě:

$$12345 = 1{,}2345 \cdot 10^4$$
$$-0{,}001 = -1{,}0 \cdot 10^{-3}$$

Jak můžeme vidět, každé číslo je reprezentováno ve tvaru $\mathtt{x{,}yz}\ldots$ vynásobeno příslušnou mocninou desíti. První část nazýváme pojmem *mantisa*. Přidáme-li ještě informaci o znaménku, dostáváme obecný tvar:

$$\pm\text{mantisa} \cdot \text{základ}^{\text{exponent}}$$

Pro daný základ (ať už to je 2, 10 nebo libovolný jiný) tedy stačí uložit informaci o znaménku, mantise a exponentu.

Přesuneme-li se do dvojkové soustavy, máme k dispozici už pouze číslice 0 a 1. Převod z dvojkové do desítkové soustavy probíhá analogicky jako u celých čísel, pouze u desetinné části musíme použít i záporné exponenty. Ukažme si na jednoduchém příkladu:

$$(1100{,}101)_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} =$$
$$= 8 + 4 + 0 + 0 + 0{,}5 + 0 + 0{,}125 = 12{,}625$$

Ve vědecké notaci bychom tedy měli:

$$+1{,}100101 \cdot 2^3$$

Což bychom mohli převést do dvojkové soustavy takto[4]:

znaménko	0
mantisa	100101
exponent	11

[4] Ve skutečnosti je přesná reprezentace v počítači o něco složitější, ale takhle nám to pro představu stačí.

Datový typ *float* používá 1 bit pro znaménko, 8 bitů pro exponent a 23 bitů pro uložení mantisy. Největší číslo, které jsme schopni reprezentovat tímto typem je asi $3{,}4028235 \cdot 10^{38}$. 23 bitů mantisy nám pak odpovídá zhruba 7 desítkovým číslicím. To je pro mnohé aplikace více než dostatečné[5], nicméně i některé triviální případy mohou činit obtíže. Všimněme si, že pro reprezentaci desetinného části čísla máme k dispozici pouze záporné mocniny dvojky, tedy čísla:

[5] Dalším rozšířením může být 8bytový typ *double*, který zvládá 16 desítkových číslic.

$$
\begin{aligned}
2^{-1} &= 0{,}5 \\
2^{-2} &= 0{,}25 \\
2^{-3} &= 0{,}125 \\
2^{-4} &= 0{,}0625 \\
&\vdots
\end{aligned}
$$

Proto, abychom vyjádřili už jen třeba číslo 0,1 bychom ovšem potřebovali nekonečně mnoho takovýchto mocnin. Typ *float* nám jich však dává k dispozici pouze 23 (počet bitů mantisy).

Mohlo by se zdát, že dvojková soustava je v tomto směru velmi nepraktická, protože nám neumožňuje přesně zapsat i takto triviální čísla. Podobný problém máme ale i v desítkové soustavě. Jak bychom zapsali přesně číslo odpovídající zlomku $1/3$?

$$\frac{1}{3} = 0{,}333333\ldots$$

Jak vidíme, i pro takto jednoduché číslo bychom v desítkové soustavě potřebovali nekonečně mnoho číslic. Naopak v trojkové soustavě je jeho vyjádření triviální, zjevně totiž platí:

$$\frac{1}{3} = 3^{-1} = 1 \cdot 3^{-1} = (0{,}1)_3$$

Necháme-li si v Pythonu vypsat číslo 0,1 na 20 desetinných míst, dostaneme následující:

```
>>> print(f'{0.1:.20}')
0.10000000000000000555
```

Zobrazený výsledek představuje nejbližší číslo k našemu zadanému.[6]

[6] Python používá 53 bitů pro mantisu (ekvivalent typu *double* v jazyce C).

Jak můžeme vidět, reálná čísla se v počítači obecně neukládají stejně, což je důsledek jejich konečné reprezentace ve dvojkové soustavě.

Vraťme se nyní k příkladům ze začátku kapitoly.

Motivační příklady – objasnění

Jednodušší z nich se zabýval porovnáním dvou reálných čísel.

```
>>> 0.1 + 0.2 == 0.3
False
```

Problematický případ už bychom měli nyní být schopni pochopit. Zobrazme si obě strany rovnosti zvlášť, například na 20 desetinných míst:

```
>>> print(f'{0.1 + 0.2:.20}')
0.30000000000000004441
>>> print(f'{0.3:.20}')
0.29999999999999999889
```

Obě čísla představují nejbližší reprezentaci, které můžeme dosáhnout. Jak správně porovnávat reálná čísla si ukážeme v následující kapitole.

Příklad s určitým integrálem je o něco komplikovanější. Připomeňme, že naším cílem bylo spočítat integrál J_{30} užitím rekurentního vztahu:

$$J_n = 1 - nJ_{n-1}$$

kde $J_0 = 1 - 1/e$. Zkusme nyní rozepsat výraz pro J_n pro několik prvních členů:

$$\begin{aligned} J_n &= 1 - nJ_{n-1} = 1 - n\,(1 - (n-1)J_{n-2}) = 1 - n + n(n-1)J_{n-2} = \\ &= 1 - n + n(n-1)(1 - (n-2)J_{n-3}) = \\ &= 1 - n + n(n-1) - n(n-1)(n-2)J_{n-3} = \dots \end{aligned}$$

Kdybychom pokračovali v rozepisování dále, poslední člen by pak obsahoval výraz $n!J_0$. Konkrétně pro náš příklad $30!J_0$. Jak víme, J_0 je zatížen chybou v reprezentaci (i když velmi malou), vynásobíme-li ji ale velkým číslem ($30! \approx 2{,}6 \cdot 10^{32}$), bude ve výsledku dominovat.

Řešením může být obrátit směr výpočtu tak, aby se chyba v každé iteraci zmenšovala. Upravme rekurentní vztah a vyjádřeme J_{n-1}:

$$J_{n-1} = \frac{1 - J_n}{n}$$

Máme-li vyčíslit hodnotu J_{30}, potřebujeme ale nyní odhadnout výsledek některého integrálu J_n, kde $n > 30$. Připomeňme, že integrál uvažujeme na intervalu 0 až 1, tedy vzhledem k charakteru funkce x^n nutně platí, že $\int_0^1 x^{n+1}\,dx < \int_0^1 x^{n+1}\,dx$, z čehož vyplývá $J_{n+1} < J_n$.

Protože víme, že J_n je číslo mezi 0 a 1, zvolme tedy například odhad $J_{35} = 0$. Opět vyjádřeme postup pomocí Pythonu:

```
>>> J = {}
>>> J[35] = 0
>>> for n in reversed(range(30, 36)):
...     J[n - 1] = (1 - J[n]) / n
...
>>> J[30]
0.03127967393216807
```

Toto číslo už představuje očekávaný výsledek a od správného se liší až na sedmnáctém desetinném místě! Změna výpočtu nám tak umožnila určit hodnotu integrálu s mnohem menší chybou. [7]

Tyto příklady demonstrovaly některé problémy, na které můžeme narazit při práci s reálnými čísly. Podívejme se nyní na některé obecné zásady práce s reálnými čísly.

Zásady práce s reálnými čísly

Jak už víme, reálná čísla se v počítači typicky neukládají přesně, což je důsledkem jejich konečné reprezentace. V několika následujících sekcích rozebereme některé z obecných problémů a ukážeme jejich řešení.

Python obsahuje i datový typ *Decimal*, který dokáže uložit desetinná čísla přesně. Jelikož se ovšem jedná o implementačně složitější typ, jeho použití s sebou přináší jistou režii.

Podrobnější pohled na jednotlivé mezivýsledky:

```
>>> J
{0: 0.6321205588285577,
 1: 0.36787944117144233,
 2: 0.26424111765711533,
 3: 0.207276647028654,
 4: 0.17089341188538398,
 5: 0.14553294057308008,
 6: 0.1268023565615195,
 7: 0.11238350406936348,
 8: 0.10093196744509214,
 9: 0.09161229299417073,
 10: 0.0838770700582927,
 11: 0.07735222935878028,
 12: 0.07177324769463667,
 13: 0.06694777996972334,
 14: 0.06273108042387321,
 15: 0.059033793641901866,
 16: 0.05545930172957014,
 17: 0.05219187059730757,
 18: -0.029453670751536265,
 19: 1.559619744279189,
 20: -30.19239488558378,
 21: 635.0402925972594,
 22: -13969.886437139707,
 23: 321308.38805421325,
 24: -7711400.313301118,
 25: 192785008.83252797,
 26: -5012410228.645727,
 27: 135335076174.43463,
 28: -3789382132883.17,
 29: 109892081853612.92,
 30: -3296762455608386.5}
```

[7] Zatímco počáteční chyba pro J_0 byla menší než 10^{-16}, chyba při odhadu J_{35} byla mnohonásobně vyšší. $J_{35} \approx 0{,}027$.

Porovnávání reálných čísel

V jednom z předchozích příkladů jsme viděli, že porovnávat reálná čísla pomocí operátorů (`==` a `!=` nebo ekvivalentních) může přinést neočekávané komplikace.[8]

[8] Překladače některých jazyků na takovéto problematické výskyty dokáží upozornit varováním.

Vzhledem k (nevyhnutelným) nepřesnostem bychom měli test na rovnost nahradit testem na blízkost, tj. místo otázky *Je* x *rovno* y*?* se ptát na *Je* x *dostatečně blízko* y*?* Co ale značí výraz *dostatečně blízko*? Tohle už musíme zvolit dle konkrétní aplikace, jestli bereme rozdíl za významný rozdíl 0,001 nebo třeba až 0,0000001.

Správné porovnání by tedy odpovídalo výrazu

$$|x - y| < \epsilon$$

kde ϵ je zvolená přesnost. Pro dříve zmíněný příklad tedy konkrétně:

```
>>> eps = 0.000001
>>> x = 0.1 + 0.2
>>> y = 0.3
>>> abs(x - y) < eps
True
```

Limity reprezentace

Dalším problémem, na který bychom mohli narazit u reálných čísel, je opět *přetečení*. To je situace, kdy při výpočtu překročíme maximální velikost čísla, které jsme schopni v počítači uložit[9]. Tento limit je poměrně velký, jak můžeme jednoduše ověřit:

[9] Existuje i stav nazvaný *podtečení*, kdy je výsledek výpočtu menší, než nejmenší hodnota, kterou jsme schopni reprezentovat. Vyzkoušejte např. vynásobit mezi sebou 1000 reálných čísel mezi 0 a 1, analogicky jako v následujícím příkladu.

```
>>> import sys
>>> sys.float_info.max
1.7976931348623157e+308
```

Nicméně i tak může nastat situace, že je při výpočtu překročíme. Uvažme výpočet geometrického průměru (pro jednoduchost uvažujme pouze kladná čísla):

$$G(a_1, \ldots, a_n) = \sqrt[n]{a_1 \cdot \ldots \cdot a_n} = \sqrt[n]{\prod_{i=1}^{n} a_i}$$

Zkusme nyní spočítat geometrický průměr 1000 reálných čísel mezi 0 a 10:

```
>>> import numpy as np
>>> array = np.random.random_sample(1000) * 10
>>> np.prod(array) ** (1 / 1000)
inf
```

I tímto jednoduchým příkladem rychle přesáhneme zmíněný limit. Řešením je v tomto případě jednoduchá transformace využívající vlastnosti logaritmů[10]. Níže s kompletním odvozením:

[10] Logaritmus součinu je součet logaritmů. Součet už však přetečením trpět nebude.

$$
\begin{aligned}
G(a_1, \ldots, a_n) &= \sqrt[n]{a_1 \cdot \ldots \cdot a_n} \\
\log G(a_1, \ldots, a_n) &= \log (a_1 \cdot \ldots \cdot a_n)^{1/n} \\
\log G(a_1, \ldots, a_n) &= 1/n \cdot \log (a_1 \cdot \ldots \cdot a_n) \\
\log G(a_1, \ldots, a_n) &= 1/n(\log a_1 + \ldots + \log a_n) \\
G(a_1, \ldots, a_n) &= \exp (1/n(\log a_1 + \ldots + \log a_n)) \\
G(a_1, \ldots, a_n) &= \exp \left(\frac{1}{n} \sum_{i=1}^{n} \log a_i \right)
\end{aligned}
$$

Upravený algoritmus už pak funguje korektně:

```
>>> import numpy as np
>>> array = np.random.random_sample(1000) * 10
>>> np.exp(np.sum(np.log(array)) / 1000)
3.76991962941375
```

Asociativita sčítání reálných čísel

Sčítání reálných čísel je asociativní operace, což znamená, že nezáleží na pořadí uzávorkování operací.

$$a + (b + c) = (a + b) + c$$

U sčítání reálných čísel v počítači to ovšem nemusí být vždy pravda. Uvažme zjednodušený příklad.

S přesností na 3 platné číslice[11] určete výsledek výrazu:

[11] Připomeňme, že typ *float* jich poskytuje asi 7, typ *double* pak 16.

$$1{,}23 + 0{,}001$$

Kdybychom nebyli nikterak omezeni na přesnost, výsledek by byl samozřejmě 1,231, nicméně v rámci zvolené přesnosti je korektní výsledek

$$1{,}23 + 0{,}001 = 1{,}23$$

Číslo 0,001 totiž při tomto sčítání v dané přesnosti odpovídá nule. Co kdybychom měli ale tento příklad?

$$1{,}23 + \underbrace{0{,}001 + \ldots + 0{,}001}_{10}$$

Pokud bychom počítali jednotlivé součty postupně, výsledek by zůstal nezměněn:

$$(1{,}23 + 0{,}001) + \underbrace{0{,}001 + \ldots + 0{,}001}_{9} = 1{,}23 + \underbrace{0{,}001 + \ldots + 0{,}001}_{9} = \ldots = 1{,}23$$

Změnou uzávorkování ale můžeme docílit jiného výsledku!

$$1{,}23 + \underbrace{(0{,}001 + \ldots + 0{,}001)}_{10} = 1{,}23 + 0{,}01 = 1{,}24$$

Jsou-li pro nás tyto rozdíly podstatné, je potřeba určit správné pořadí provedených operací.

Ukažme si nyní, že tento problém skutečně existuje, i když použijeme vyšší přesnost, jak je typické v běžných programovacích jazycích.

```
>>> import numpy as np
>>> array = np.random.random_sample(10)
>>> # Vytvoříme nové pole jako permutaci původního
>>> array2 = np.random.permutation(array)
>>> sum(array) == sum(array2)
False
>>> # Vyrobíme jinou permutaci
>>> array3 = np.random.permutation(array)
>>> sum(array) == sum(array3)
True
```

Opět můžeme vidět, že výsledek je v těchto dvou případech odlišný[12].

[12] Pokud si uvedený příklad vyzkoušíte, můžete dostat rozdílné výsledky v závislosti na vygenerovaných číslech a zvolené permutaci.

Sčítáme-li reálná čísla (zejména s velkými řádovými rozdíly), je užitečné je nejprve seřadit podle velikosti (v absolutní hodnotě) a sčítat od nejmenšího.[13]

[13] V Pythonu bychom mohli použít například:
`sum(sorted(array, key=abs))`

Stabilita algoritmu

Uveďme ještě poslední příklad – známou Gaussovu eliminační metodu (GEM) pro řešení systému lineárních rovnic.[14]

Ačkoliv je tato metoda velmi jednoduchá, v praxi se nepoužívá. Uvažme příklad:

[14] Připomeňme, že cílem je eliminace prvků pod hlavní diagonálou užitím ekvivalentních operací (přičtením k-násobku jednoho řádku k druhému, vynásobení řádku nenulovým reálným číslem a prohození dvou řádků).

$$\begin{aligned} 10^{-10}x_1 + \ x_2 &= 1 \\ x_1 + 2x_2 &= 3 \end{aligned}$$

První krok GEM pak vypadá následovně:

$$\left[\begin{array}{cc|c} 10^{-10} & 1 & 1 \\ 1 & 2 & 3 \end{array}\right] \sim \left[\begin{array}{cc|c} 10^{-10} & 1 & 1 \\ 0 & 2-10^{10} & 3-10^{10} \end{array}\right]$$

Vzhledem k omezené přesnosti bude ovšem tento systém ekvivalentní tomuto:[15]

[15] Protože sčítáme dvě čísla s velmi odlišnými řády, bude při výpočtu na počítači platit, že menší z nich se ve výsledku vůbec neprojeví.
Např. tedy $2 - 10^{10} = -10^{10}$.

$$\left[\begin{array}{cc|c} 10^{-10} & 1 & 1 \\ 0 & -10^{10} & -10^{10} \end{array}\right] \sim \left[\begin{array}{cc|c} 10^{-10} & 1 & 1 \\ 0 & 1 & 1 \end{array}\right] \sim \left[\begin{array}{cc|c} 10^{-10} & 0 & 0 \\ 0 & 1 & 1 \end{array}\right] \sim \left[\begin{array}{cc|c} 1 & 0 & 0 \\ 0 & 1 & 1 \end{array}\right]$$

Dostáváme tedy řešení $x_1 = 0, x_2 = 1$. Přesné řešení je nicméně $x_1 = x_2 = 1$. Přesvědčte se o tom dosazením do původní soustavy.

Správnější variantou je modifikace Gaussovy eliminační metody přidáním tzv. částečného výběru hlavního prvku (pivota). Rozdíl bude ten, že v každém kroku prohodíme zbývající nezpracované řádky tak, že pivotem při eliminace bude prvek s největší absolutní hodnotou. V našem příkladě tedy v prvním kroce dojde k prohození řádků:

$$\left[\begin{array}{cc|c} 10^{-10} & 1 & 1 \\ 1 & 2 & 3 \end{array}\right] \sim \left[\begin{array}{cc|c} 1 & 2 & 3 \\ 10^{-10} & 1 & 1 \end{array}\right]$$

Dále postupujeme standardně, až ke správnému výsledku[16]:

$$\left[\begin{array}{cc|c} 1 & 2 & 3 \\ 10^{-10} & 1 & 1 \end{array}\right] \sim \left[\begin{array}{cc|c} 1 & 2 & 3 \\ 0 & 1-2\cdot 10^{-10} & 1-3\cdot 10^{-10} \end{array}\right] \sim$$
$$\sim \left[\begin{array}{cc|c} 1 & 2 & 3 \\ 0 & 1 & 1 \end{array}\right] \sim \left[\begin{array}{cc|c} 1 & 0 & 1 \\ 0 & 1 & 1 \end{array}\right]$$

Gaussova eliminační metoda v původní podobě je tzv. *numericky nestabilní* algoritmus[17]. Podobně jako u jiných příkladů z této kapitoly jsme změnou algoritmu byli schopni dosáhnout přesného řešení.

Závěrem dodejme, že pro většinu obvyklých operací (např. z lineární algebry, které se budeme věnovat dále), existují předpřipravené knihovny funkcí, které jsou psány s ohledem na výše zmíněné problémy, a je doporučeno je použít místo vlastních řešení.[18]

[16] Opět poznamenejme, že vlivem velkého rozdílů v řádech bude platit, že

$$1 - 2 \cdot 10^{-10} = 1$$

[17] Algoritmus označíme za *stabilní*, pokud řešení, které užitím něho získáme, je přesným řešením pro nějaké blízké zadání.

[18] V některých případech budeme na tyto funkce odkazovat v poznámkách.

Pojmy lineární algebry

V poslední části předchozí kapitoly jsme si ve stručnosti připomněli Gaussovu eliminační metodu, jedním ze základních algoritmů lineární algebry. Než se pustíme do konkrétních témat, zopakujme si několik základních pojmů.

Vektory

Ústředním pojmem je v lineární algebře *vektor*, tedy uspořádaná n-tice čísel. Označujeme je nejčastěji písmeny jako u, v nebo w. Naproti tomu obyčejná reálná čísla (skaláry) budeme značit nejčastěji $a, b, c, \ldots$. Počet prvků vektoru označujeme jako jeho dimenzi. Např.[19]

[19] Často se místo hranatých používají kulaté závorky. Na významu se nic nemění, jen je potřeba je používat konzistentně.

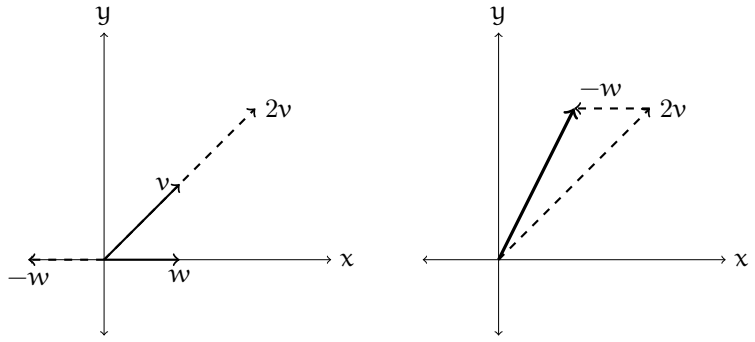
$$u = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}, \quad v = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad w = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Dimenze vektoru u je 3, zatímco u vektoru v a w pak 2. Základní operace s vektory jsou jejich sčítání[20] a násobení skalárem. Např.

[20] Sčítat lze samozřejmě pouze vektory stejné dimenze.

$$2v - w = 2\begin{bmatrix} 1 \\ 1 \end{bmatrix} - 1\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} + \begin{bmatrix} -1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Stejnou operaci můžeme vyjádřit i graficky:



Vektory běžně uvažujeme jako tzv. *sloupcové*. Někdy však může být užitečné pracovat s *řádkovými* vektory. Operace, která provádí „konverzi“ mezi těmito dvěma vyjádřeními, nazveme transpozicí[21].

[21] Zjevně platí $(v^T)^T = v$.

$$v^T = \begin{bmatrix} 1 \\ 1 \end{bmatrix}^T = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

Matice

Dalším důležitým pojmem je matice. Matice není nic jiného než obdélníkový způsob zápisu čísel. [22] Chceme-li pracovat s několika vektory najednou, můžeme je zapsat například jako jednotlivé sloupce matice. Potřebujeme-li navíc matici nějak pojmenovat, volíme velká písmena ze začátku abecedy ($A, B, \ldots$):

$$A = \begin{bmatrix} | & | \\ v & w \\ | & | \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

[22] Na tomto místě se hodí zmínit, že vektory i matice slouží často jako příhodný nástroj pro vyjádření určitých typů problémů. Data (= jednotlivá čísla) v nich uložená nabývají takového významu, jaký jim dáme. Může jít o koncentrace sloučenin v roztoku, informace o věku skupiny osob, souřadnic objektů v prostoru atd.

Pro matici A, která má m řádků a n sloupců, zapisujeme její rozměry explicitně jako $A_{m\times n}$.

Vertikální nebo horizontální linky v příkladech pouze explicitně zdůrazňují, že se jedná o sloupcový, resp. řádkový vektor.

Stejně jako u vektorů, i u matic je definována transpozice.

$$A^T = \begin{bmatrix} | & | \\ v & w \\ | & | \end{bmatrix}^T = \begin{bmatrix} — & v^T & — \\ — & w^T & — \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

V našem případě k žádné změně nedošlo, platí, že $A^T = A$. O takové matici A řekneme, že je *symetrická*.

Přehled pojmů pro tuto chvíli zakončíme klíčovým termínem celé lineární algebry.

Lineární kombinace

Možná jste někdy přemýšleli nad tím, proč se lineární algebře říká *lineární*. Tohle označení souvisí s pojmem lineární kombinace. Umíme-li sčítat vektory a násobit je skalárem, nebude to pro nás nic nového.

Lineární kombinací vektorů v, w[23] nazveme výraz

$$av + bw$$

kde a, b jsou reálná čísla. Tyto nazveme koeficienty lineární kombinace.

[23] Zjevně těchto vektorů může být libovolný počet, všechny musí mít ale stejnou dimenzi.

Zopakujme nyní příklad ze začátku této kapitoly:

$$2v - w = 2\begin{bmatrix} 1 \\ 1 \end{bmatrix} - 1\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} + \begin{bmatrix} -1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Zde máme lineární kombinaci vektorů v a w s koeficienty 2 a -1.

Lineární kombinaci můžeme příhodně vyjádřit jako násobení matice vektorem. Matice bude obsahovat původní vektory jako své sloupce, koeficienty lineární kombinace zapíšeme jako nový vektor. [24]

$$2v - w = \begin{bmatrix} | & | \\ v & w \\ | & | \end{bmatrix} \cdot \begin{bmatrix} 2 \\ -1 \end{bmatrix}$$

[24] Obecně tedy:

$$a_1v_1 + \ldots + a_nv_n = \begin{bmatrix} | & & | \\ v_1 & \ldots & v_n \\ | & & | \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}$$

Systémy lineárních rovnic

Řešení systému lineárních rovnic je jeden z nejčastějších problémů, který se objevuje v mnoha různých aplikacích. V této kapitole si ukážeme, jak o takových systémech uvažovat z pohledu lineární algebry a zaměříme se na vlastnosti, které nám pomohou nalézt efektivně jejich řešení.[25]

[25] Lineární algebra se často učí jako přehled algoritmů, postupů pro řešení konkrétního typu problému („Máte-li systém lineárních rovnic, použijte Gaussovu eliminační metodu.“, ...). Tímto způsobem se sice můžeme naučit efektivně řešit příklady, nicméně s pochopením *jak to celé funguje a proč* nám to nepomůže. Následující kapitoly jsou proto koncipovány trochu odlišně a více než na prezentaci postupů se budou soustředit na analýzu vlastností a souvislosti mezi probíranými koncepty.

Opakování

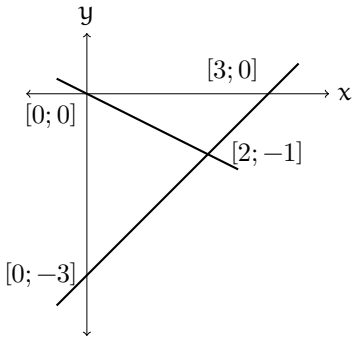
Uvažme jednoduchý systém dvou lineárních rovnic o dvou neznámých:

$$\begin{aligned} x + 2y &= 0 \\ x - \ y &= 3 \end{aligned}$$

Ať zvolíme libovolnou metodu řešení, nemělo by nás překvapit výsledek $x = 2, y = -1$.

Pojďme si přiblížit tuto soustavu rovnic graficky. Každou rovnici můžeme brát jako jedno z omezení na množinu všech bodů $[x; y]$. Grafem znázorňující každou z rovnic je přímka, kterou dokážeme jednoznačně určit dvěma body. Snadno nahlédneme, že první z nich může procházet například body $[0; 0]$ a $[2; -1]$, druhá pak body např. $[3; 0]$ a $[0; -3]$.

Řešením je pak jejich průnik – obecně množina bodů, která splňuje obě daná omezení.



Připomeňme, že by obecně mohly nastat ještě dvě situace. Pokud by se obě přímky překrývaly, vyjadřovaly by nám stejné (i když obecně jinak explicitně vyjádřené) omezení. Taková soustava by pak měla nekonečně mnoho řešení, konkrétně každý bod ležící na těchto přímkách. Naopak pokud by obě přímky byly rovnoběžné, nebudou mít žádný průnik, neboť zadávají neslučující se omezení. Soustava rovnic by pak neměla žádné řešení.

Systém lineárních rovnic pohledem lineární kombinace

Zkusme se nyní podívat na složitější příklad:

$$\begin{aligned} 2x - y &= 0 \\ -x + 2y - z &= -1 \\ -3y + 4z &= 4 \end{aligned}$$

Grafické vyjádření by nám v tomto případě nebylo moc užitečné, protože najednou potřebujeme pracovat v trojrozměrném prostoru. Každá rovnice nám pak zadává jednu rovinu. Nakreslit tyto roviny a určit z jejich průniku řešení by však bylo obecně nepraktické.

Zkusme se nyní podívat na stejný systém s využitím maticového počtu. Takový systém se obecně zapisuje ve tvaru

$$Ax = b$$

kde A je matice koeficientů, b je vektor pravé strany a x vektor neznámých. Konkrétně pro náš příklad:

$$\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -3 & 4 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 4 \end{bmatrix}$$

Takovouto soustavu můžeme zapsat do zjednodušeného tvaru vynecháním vektoru neznámých:

$$\left[\begin{array}{ccc|c} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & -1 \\ 0 & -3 & 4 & 4 \end{array}\right]$$

a dále řešit standardně např. Gaussovou eliminační metodou.

Zkusme se ale nyní zaměřit na jiný způsob řešení, který bude využívat vlastností lineárních systémů.

Předchozí systém lineárních rovnic můžeme rozepsat jako lineární kombinaci sloupců matice A:

$$x \cdot \begin{bmatrix} 2 \\ -1 \\ 0 \end{bmatrix} + y \cdot \begin{bmatrix} -1 \\ 3 \\ 3 \end{bmatrix} + z \cdot \begin{bmatrix} 0 \\ -1 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 4 \end{bmatrix}$$

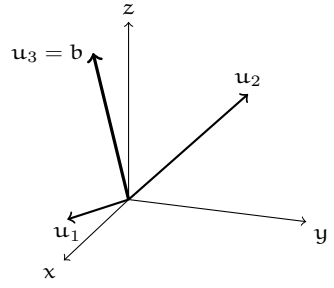
Hledáme tedy takové koeficienty lineární kombinace, které nám poté v součtu dají vektor pravé strany. V tomto vyjádření je ale řešení tohoto příkladu poměrně zjevné, povšimneme-li si, že poslední vektor a vektor pravé strany jsou shodné. Řešení je tedy $x = y = 0, z = 1$.

Tento příklad nám poměrně hezky demonstruje, jak souvisí lineární kombinace s řešitelností soustavy lineárních rovnic.

SYSTÉM $Ax = b$ MÁ ŘEŠENÍ právě tehdy, když je vektor b lineární kombinací sloupců matice A.

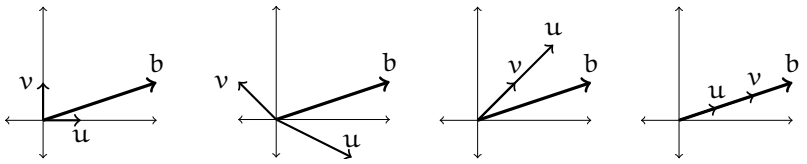
Zkusme si přiblížit několik typově různých systémů lineárních rovnic obrázkem. Uvažme soustavu dvou rovnic o dvou neznámých:

Zkusme si nyní také přiblížit grafické znázornění celé soustavy, kdy nebudeme uvažovat jednotlivé rovnice (řádky), ale zaměříme se na sloupce, výše uvedené vektory. Ty si označíme postupně u_1, u_2, u_3 a vektor pravé strany pak b. Řešení je opět zjevné.



$$Ax = \begin{bmatrix} | & | \\ u & v \\ | & | \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = b$$

Zaměřme se teď na následující čtyři příklady:



Ve všech případech se ptáme: „Kolikrát mám vzít vektor u a kolikrát v, aby jejich součet byl vektor b?“ Protože ale nemáme explicitně uvedeny hodnoty, místo konkrétních koeficientů nás bude zajímat, zdali lze vektor b vyjádřit pomocí vektorů u a v (tedy zdali existuje nějaká lineární kombinace, která by vedla na vektor pravé strany).

V prvním případě je to poměrně přímočaré, tam takovou kombinaci jistě najdeme. Tento systém bude mít právě jedno řešení.

Ve třetím a čtvrtém případě je v násobkem u (a naopak). Libovolná jejich lineární kombinace tedy bude vektor ležící na stejné přímce jako u a v. To znamená, že ve třetím případě se nám vektor b „nepodaří“ vyrobit, ve čtvrtém máme naopak nekonečně mnoho možností.

Druhý příklad může být na první pohled obtížnější. Zkusme si uvědomit, jak by vypadaly všechny lineární kombinace zadaných vektorů u a v (zejména ve srovnání s ostatními příklady). Protože oba vektory neleží v přímce, podaří se nám pomocí lineárních kombinací popsat libovolný jiný vektor v rovině, a tedy i vektor b. Tento systém bude mít jedno řešení.

Lineární (ne)závislost

Zkusme nyní pozorování z předchozích příkladů zobecnit a podívat se, jaké vlastnosti musí mít matice A, aby měl systém $Ax = b$ řešení pro libovolný vektor pravé strany b.

Vzpomeňme si, že stačilo, aby jeden sloupec matice A nebyl násobkem druhého, a věděli jsme že existuje lineární kombinace vedoucí na vektor b.

Pokud se přesuneme alespoň o dimenzi výše, situace se nám poněkud zkomplikuje. Uvažme následující matici A, kde žádný sloupec není násobkem jiného:

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 1 & 1 & 1 \\ 0 & 2 & -2 \end{bmatrix}$$

Ačkoliv to není možná na první pohled vidět, všechny vektory tvořící sloupce této matice, leží v jedné rovině v trojrozměrném prostoru. Proč tomu tak je? Všimněme si, že libovolný z vektorů můžu vyjádřit jako lineární kombinaci zbylých dvou, např.

$$2 \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} - 1 \cdot \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ -2 \end{bmatrix}$$

Jeden z vektorů je tedy z jistého pohledu „zbytečný“, ničím nám v rámci lineárních kombinací nepřispívá. To také znamená, že jistě nedokážeme vyjádřit každý vektor b jako lineární kombinaci sloupců matice A[26].

Abychom byli schopni vyjádřit libovolný vektor b v trojrozměrném prostoru, potřebujeme tedy tři vektory takové, že každý přispívá do lineární kombinace „něčím novým“. To znamená, že žádný z nich nelze vyjádřit jako lineární kombinace zbylých. O takových vektorech prohlásíme, že jsou *lineárně nezávislé*. Pokud to lze, označíme je za *lineárně závislé*.

[26] Vyjádřit se nám podaří pouze takový vektor b, který leží ve stejné rovině jako vektory z matice A.

Formální definice říká, že vektory jsou lineárně nezávislé, pokud jediná lineární kombinace, která vede na nulový vektor, je triviální (tj. všechny její koeficienty jsou nulové).

$$a_1 v_1 + \dots a_n v_n = 0 \rightarrow a_1 = \dots a_n = 0$$

Naopak pokud lze najít lineární kombinaci, která vede na nulový vektor, takovou, že alespoň jeden její koeficient je nenulový, vektory jsou lineárně závislé.[27]

[27] Zamyslete se, proč to pak znamená, že některý z vektorů dokážu vyjádřit jako lineární kombinaci zbylých.

Jak tedy určit, zdali jsou vektory lineárně závislé nebo nezávislé?

V jednoduchých příkladech to může jít vidět přímo (jeden vektor je násobkem druhého; jeden vektor je nulový; dva vektory jsou stejné, atd.), neplatí to ale vždy.

Standardní postup spočívá v tom, že si vektory napíšeme do matice (jako řádky)[28] a provedeme Gaussovu eliminaci, tedy úpravu na schodovitý tvar. Pokud po jejím skončení získáme nějaký nulový řádek, vektory byly lineárně závislé. Pokud ne, byly lineárně nezávislé.[29] [30]

[28] Pro sloupce to funguje stejně, ale tento způsob je intuitivnější pro pochopení.

[29] Proč ale tento postup funguje? Uvědomme si, že Gaussova eliminace je založena na přičítání nějakého násobku jednoho řádku k jinému. Intuitivně: Pokud se nám podaří některý z řádků „vynulovat“, znamená to, že jej šlo „vyrobit“ (ve smyslu lineární kombinace) z násobků řádků předešlých.

[30] Možná si pamatujete pojem *hodnost* matice. To je počet lineárně nezávislých řádků matice. A to je stejné číslo jako počet nenulových řádků matice upravené na schodovitý tvar.

Vraťme se nyní k původní otázce, tj. kdy je systém lineárních rovnic řešitelný pro libovolný vektor b, nyní však s využitím pojmů představených výše.

Systém n lineárních rovnic $Ax = b$ má řešení vždy (= pro každé b), pokud matice A obsahuje n lineárně nezávislých vektorů.

Homogenní systémy

Krátce zmíníme ještě speciální případ systému lineárních rovnic, pro který platí $b = 0$. Takový systém nazveme *homogenní*.

$$Ax = 0$$

V čem je tento systém zvláštní? Zkuste jej opět rozepsat jako lineární kombinaci sloupců matice A. Takový systém má zjevně vždycky řešení, a to je $x = 0$. [31]

[31] Zamyslete se, jak to souvisí s pojmem lineární nezávislosti.

Dále platí, že má-li takový systém další (= nenulové) řešení, má jich rovnou nekonečně mnoho.

Pro demonstraci využijme dříve zmíněnou matici A:

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 1 & 1 & 1 \\ 0 & 2 & -2 \end{bmatrix}$$

Systém $Ax = 0$ má řešení:

$$x = \begin{bmatrix} 2 \\ -1 \\ -1 \end{bmatrix}$$

Dalšími řešeními jsou pak všechny násobky tohoto vektoru, tj. kx pro libovolné reálné číslo k. Ověřte si to rozepsáním součinu $A(kx)$ jako lineární kombinace.

Pro úplnost dodejme, že pokud má homogenní systém dvě řešení x a y, pak je řešením i jejich libovolná lineární kombinace. Zkuste si to ověřit podobně jako v tomto případě.

Dosavadní souvislosti

Než se pustíme v tématech dále, ukažme si na jednom místě, jak spolu všechny dosud zmíněné informace souvisí.

Mějme matici $A_{n\times n}$. Tu nazveme *regulární*[32], pokud platí:

[32] V opačném případě mluvíme o tzv. *singulární* matici.

- počet lineárně nezávislých řádků matice A je n
- počet lineárně nezávislých sloupců matice A je n
- počet nenulových řádků matice A převedené na schodovitý tvar je n
- hodnost matice A je n
- soustava $Ax = 0$ má pouze triviální řešení ($x = 0$)
- soustava $Ax = b$ má právě jedno řešení

Ačkoliv se v tomto textu determinanty nezabýváme, tak zmiňme, že také $\det A \neq 0$.

Všimněme si, že všechny výše uvedené vlastnosti jsou ekvivalentní (pokud platí jedna, platí všechny ostatní) a vystihují všechno, co jsme si doposud v lineární algebře pověděli.

V tuto chvíli může být užitečné projít si celou kapitolu ještě jednou od začátku a zaměřit se na všechny souvislosti, které jsme uvedli později a nyní sumarizovali.

Metoda nejmenších čtverců

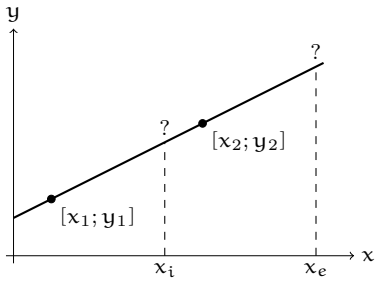
Následující sekce se bude zabývat soustavami lineárních rovnic, které nemají řešení. Ačkoliv to nezní úplně prakticky, ukážeme si, že jsou tyto příklady poměrně běžné a užitečné.

Představme si jednoduchý experiment, kdy měříme nějakou vlastnost (třeba teplotu) pozorovaného systému v čase. Víme-li, že mezi veličinami existuje např. lineární závislost, můžeme tento vztah vyjádřit jednoduše rovnicí

$$y = kx + q$$

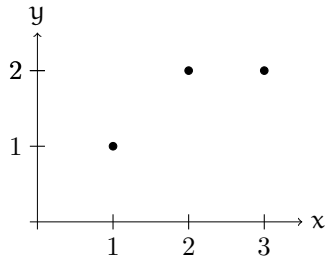
kde x je hodnota nezávislé veličiny, y hodnota měřené veličiny a k, q jsou pak nějaké parametry.

Abychom tyto parametry určili jednoznačně, budeme potřebovat dvojici měření $([x_1, y_1], [x_2, y_2])$. Zadaný problém pak vede na soustavu dvou rovnic o dvou neznámých (k, q).



S VYPOČTENÝMI HODNOTAMI PARAMETRŮ pak můžeme takový model použít k *interpolaci* (x_i na obrázku vpravo) a *extrapolaci* (x_e), tzn. k odhadu hodnot mezi jednotlivými měřeními, resp. mimo ně.

Protože jsou ale měření často zatíženy chybami, je užitečné jich provést více. Ukažme si zjednodušený příklad. Mějme pro jednoduchost tři měření, jak je znázorněno na následujícím obrázku.



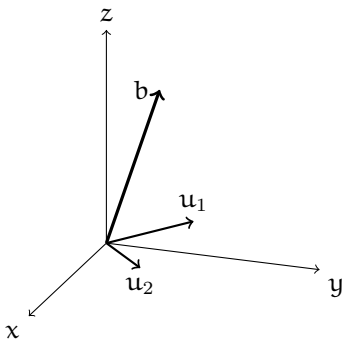
Poměrně snadno jde vidět, že neexistuje přímka, která by procházela všemi třemi body. V řeči lineárních systémů to znamená, že by naše soustava rovnic neměla žádné řešení.

Zkusme se tedy pokusit nalézt řešení, které by bylo v jistém smyslu nejlepší možné.[33,34]

Sestavme si nejprve tuto soustavu tří rovnic o dvou neznámých:

$$1k + q = 1$$
$$2k + q = 2$$
$$3k + q = 2$$

Ukažme si nyní pohled přes jednotlivé sloupce [35] na jinak analogickou soustavu (aby byl obrázek přehlednější, zvolili jsme jiné hodnoty vektorů):



[33] S nejmenší odchylkou od zadaných bodů. Co formálně znamená, si definujeme později.

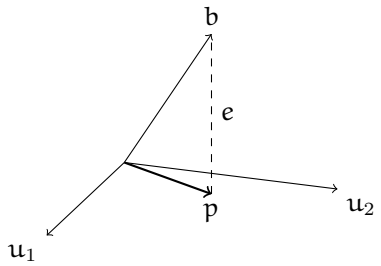
[34] Tuto úlohu můžeme nalézt i pod názvem *lineární regrese*.

[35] Soustava vyjádřená jako lineární kombinace:

$$k \cdot \begin{bmatrix}1\\2\\3\end{bmatrix} + q \cdot \begin{bmatrix}1\\1\\1\end{bmatrix} = \begin{bmatrix}1\\2\\2\end{bmatrix}$$

Uvážíme-li všechny možné hodnoty parametrů k a q, bude lineární kombinace $ku_1 + qu_2$ popisovat nějakou rovinu v trojrozměrném prostoru. Připomeňme, že aby měl náš systém rovnic řešení, musí vektor b ležet v této rovině (= být lineární kombinací vektorů u_1 a u_2).

Jak tedy vypadá vektor, který by ležel v této rovině a byl zadanému vektoru b nejblíže? Takovou vlastnost má projekce vektoru b do této roviny, což si zkusíme ukázat na následujícím obrázku:



Vektor p je naše hledaná projekce, vektor e pak označuje chybový vektor, tj. rozdíl mezi b a p, píšeme:

$$p = b - e$$

Zároveň platí, že vektor p už je nyní lineární kombinací vektorů u_1 a u_2 pro nějaké vhodně zvolené koeficienty $\widehat{x_1}$ a $\widehat{x_2}$, píšeme: [36]

$$\begin{aligned} p &= \widehat{x_1}u_1 + \widehat{x_2}u_2 \\ p &= \begin{bmatrix} u_1 & u_2 \end{bmatrix} \cdot \begin{bmatrix} \widehat{x_1} \\ \widehat{x_2} \end{bmatrix} \\ p &= A\widehat{x} \end{aligned}$$

[36] Držíme se standardního značení, kdy naše vektory u_1, u_2 tvoří sloupce matice A a vektor $\widehat{x} = \begin{bmatrix} \widehat{x_1} \\ \widehat{x_2} \end{bmatrix}$.

Protože je vektor p projekcí do roviny, znamená to, že je vektor e na tuto rovinu kolmý. Zejména je tedy kolmý na vektory u_1 a u_2. Víme také, že skalární součin dvou na sebe kolmých vektorů je 0:

$$\begin{aligned} u_1 \perp e &\Leftrightarrow u_1^T e = 0 \\ u_2 \perp e &\Leftrightarrow u_2^T e = 0 \end{aligned}$$

Všimněme si, že můžeme obě rovnosti sloučit do jedné, vektory u_1^T a u_2^T totiž představují *řádky* matice A^T. Navíc vyjádříme e užitím b a p:

$$\begin{aligned} A^T e &= 0 \\ A^T(b - p) &= 0 \\ A^T b &= A^T p \end{aligned}$$

Vyjádřením projekce p pak získáme nový systém lineárních rovnic, který už ale oproti původnímu systému $Ax = b$ bude mít řešení. Tato transformace se nazývá *metoda nejmenších čtverců*. [37]

[37] Naším cílem je minimalizace velikosti chybového vektoru e. Připomeňme si, že velikost vektoru e získáme jako $\sqrt{e^T e} = \sqrt{e_1^2 + e_2^2 + e_3^2}$. Název metody pak odkazuje na součet druhých mocnin jednotlivých složek chybového vektoru.

$$A^T A\widehat{x} = A^T b$$

Metoda nejmenších čtverců pro řešení stejného problému se objevuje i v matematické analýze, kde bychom pro řešení použili parciální derivace.

Zkusme nyní aplikovat tuto metodu na náš původní příklad.

$$\begin{bmatrix}1 & 2 & 3\\ 1 & 1 & 1\end{bmatrix}\cdot\begin{bmatrix}1 & 1\\ 2 & 1\\ 3 & 1\end{bmatrix}\cdot\begin{bmatrix}\widehat{x_1}\\ \widehat{x_2}\end{bmatrix}=\begin{bmatrix}1 & 2 & 3\\ 1 & 1 & 1\end{bmatrix}\cdot\begin{bmatrix}1\\ 2\\ 2\end{bmatrix}$$

$$\begin{bmatrix}14 & 6\\ 6 & 3\end{bmatrix}\cdot\begin{bmatrix}\widehat{x_1}\\ \widehat{x_2}\end{bmatrix}=\begin{bmatrix}11\\ 5\end{bmatrix}$$

Tento systém už vyřešíme standardně Gaussovou eliminační metodou. Získáme řešení $\widehat{x_1} = 1/2, \widehat{x_2} = 2/3$, což odpovídá přímce:

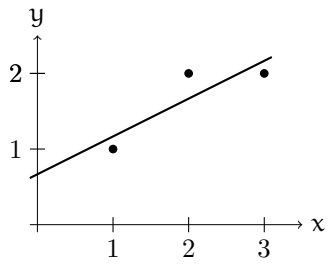
Vyjádřeme si ještě pro úplnost vektory p a e:

$$p = A\widehat{x} = \begin{bmatrix}1 & 1\\ 2 & 1\\ 3 & 1\end{bmatrix}\cdot\begin{bmatrix}1/2\\ 2/3\end{bmatrix} = \begin{bmatrix}7/6\\ 10/6\\ 13/6\end{bmatrix}$$

$$e = b - p$$

$$e = \begin{bmatrix}1\\ 2\\ 2\end{bmatrix} - \begin{bmatrix}7/6\\ 10/6\\ 13/6\end{bmatrix} = \begin{bmatrix}-1/6\\ 2/6\\ -1/6\end{bmatrix}$$

$$y = 1/2x + 2/3$$



Řešení v Pythonu je zde poměrně přímočaré, srovnejme dva přístupy. První používá přímo metodu `lstsq` (z angl. least squares), druhá pak představuje explicitní výpočet pomocí vzorce odvozeného výše.[38]

[38] Doporučeno je používat první možnost, která používá numericky stabilní algoritmus.

```
>>> A = np.array([[1, 1], [2, 1], [3, 1]])
>>> b = np.array([1, 2, 2])
>>> np.linalg.lstsq(A, b)[0]
array([0.5       , 0.66666667])
>>> np.linalg.solve(A.T @ A, A.T @ b)
array([0.5       , 0.66666667])
```

Podmíněnost systému lineárních rovnic

Tuto kapitolu zakončíme jednou zajímavou vlastností systémů lineárních rovnic.

Uvažme jednoduchý příklad:

$$\begin{aligned}10x + \quad y &= 11\\ x + 0{,}11y &= 1{,}11\end{aligned}$$

V Pythonu můžeme vyřešit tento systém takto:

```
>>> import numpy as np
>>> A = np.array([[10, 1], [1, 0.11]])
>>> b = np.array([11, 1.11])
>>> np.linalg.solve(A, b)
array([1., 1.])
```

Aniž bychom zkusili systém nějakou vhodnou metodou řešit, všimneme si rovnou, že řešení je jednoduché $x = y = 1$.

Zkusme nyní systém mírně modifikovat, pravou stranu druhé rovnice zmenšíme o 0,01:

$$\begin{aligned} 10x + \quad y &= 11 \\ x + 0{,}11y &= 1{,}1 \end{aligned}$$

Ačkoliv je změna pouze v jednom čísle a to dokonce menší než $1\,\%$, řešení tohoto systému se výrazně změní. Nově totiž získáme řešení $x = 1{,}1$ a $y = 0$.

Takový systém, který je velmi citlivý na změny, nazveme *špatně podmíněný*. Můžeme sepsat i opačnou definici.

SYSTÉM JE DOBŘE PODMÍNĚNÝ (intuitivně), pokud malá změna zadání způsobí pouze malou změnu ve výsledku.

Zkusme vzít systém rovnic z úvodu kapitoly a provést podobnou modifikaci i u něj. Abychom se vyhnuli zdlouhavým výpočtům, pomůžeme si Pythonem:

```
>>> import numpy as np
>>> A = np.array([[1, 2], [1,  -1]])
>>> b = np.array([0, 3])
>>> bb = np.array([0, 2.99])
>>> np.linalg.solve(A, b)
array([ 2., -1.])
>>> np.linalg.solve(A, bb)
array([ 1.99333333, -0.99666667])
```

V tomto případě k žádnému překvapení nedochází, výsledek pro modifikované zadaní se liší pouze minimálně.

Nabízí se přirozená otázka, v čem se tyto systémy lineárních rovnic liší. Zaměřme se na to, jak vypadají sloupce matice A. V obou případech jsou tvořeny lineárně nezávislými vektory (jeden není násobkem druhého). [39] U prvního z nich však tento rozdíl není velký:

$$\begin{bmatrix} 10 \\ 1 \end{bmatrix} \approx 10 \cdot \begin{bmatrix} 1 \\ 0{,}11 \end{bmatrix}$$

Jsou-li tedy sloupce vektory, které mají „blízko k lineární závislosti“, můžeme malá změna v zadání (např. chyba při měření, chyba reprezentace) výrazně ovlivnit výsledek.

Všimněme si, že podmíněnost je vlastnost problému, nikoliv algoritmu, který byl při výpočtu použit. V obou případech jsou totiž výsledky přesné vzhledem ke konkrétnímu zadání, nejde tak o chybu danou výpočtem.[40]

[39] Připomeňme, že kdyby se jednalo o lineárně závislé vektory, řešení by obecně neexistovalo.

[40] Jen pro úplnost dodejme, že užitím složitějšího matematického aparátu jsme schopni odhadnout velikost chyby, kterou může konkrétní změna zadání způsobit.

Násobení matic

Jedna z nejdůležitějších operací (vedle lineární kombinace), se kterou se setkáme, je násobení matic, které budeme hojně využívat v další kapitole. I zde existuje několik postupů, které můžeme zvolit, každý nám umožní nahlédnout na problém z trochu jiného pohledu.

Všechny postupy demonstrujme s použitím následujících matic:

$$A = \begin{bmatrix} 1 & 2 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}$$

Abychom mohli dvě matice vynásobit, musí platit, že počet sloupců první matice je roven počtu řádků druhé matice.

Násobení matic není komutativní, tzn. obecně neplatí, že bychom mohli matice při násobení prohodit (stejně jako je to možné u reálných čísel).

$$A \cdot B \neq B \cdot A$$

Výsledná matice má stejný počet řádků jako první matice a stejný počet sloupců jako druhá. Obecně:[41]

$$A_{m\times n} \cdot B_{n\times p} = C_{m\times p}$$

[41] Jako pomůcka může v tomto zápisu pomoci, že vnitřní čísla musí být stejná a vnější pak udávají rozměry výsledku.

Pro naše matice A a B je tato podmínka splněna. Rozměry výsledné matice pak budou 3×2. Nyní můžeme přejít k prvnímu způsobu výpočtu.

Pro násobení matic v Pythonu existuje operátor `@`:

```
>>> import numpy as np
>>> A = np.array([[1, 2],[0, 1],[1, 1]])
>>> B = np.array([[1, 1],[2, 1]])
>>> A @ B
array([[5, 3],
       [2, 1],
       [3, 2]])
```

Skalární součiny

Skalární součin je první a asi nejčastěji zmiňovaný způsob pro násobení matic. Konkrétně prvek v i. řádku a j. sloupci výsledné matice získáme jako skalární součin i. řádku matice A a j. sloupce matice B.

Například pro druhý řádek a první sloupec dostáváme:

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \end{bmatrix} = 0 \cdot 1 + 1 \cdot 2 = 2$$

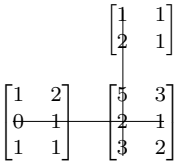
Stejným způsobem určíme všechny ostatní prvky:

$$A \cdot B = \begin{bmatrix} 1 & 2 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1\cdot 1 + 2 \cdot 2 & 1 \cdot 1 + 2 \cdot 1 \\ 0 \cdot 1 + 1 \cdot 2 & 0 \cdot 1 + 1 \cdot 1 \\ 1 \cdot 1 + 1 \cdot 2 & 1 \cdot 1 + 1 \cdot 1 \end{bmatrix} = \begin{bmatrix} 5 & 3 \\ 2 & 1 \\ 3 & 2 \end{bmatrix}$$

Tento postup je poměrně přímočarý na výpočet, nicméně nám neumožňuje přesněji nahlédnout do podstaty násobení matic. To nyní napravíme.

U větších matic může být obtížnější zorientovat se v tom, který řádek násobit s kterým sloupcem. Zkusme si druhou matici napsat „o řádek výše“. Pak virtuální průsečík každého řádku a sloupce udává místo, kam zapíšeme výsledek konkrétního skalárního součinu.

Jako bonus nemusíme přemýšlet ani nad rozměry výsledné matice, ty jsou zřejmé z tvaru tohoto schématu.



Lineární kombinace sloupců

Připomeňme si, že pokud násobíme matici vektorem, je to stejné, jedná se o lineární kombinaci sloupců této matice, jejíž koeficienty jsou jednotlivé prvky daného vektoru. Vezměme pro příklad matici A a první sloupec matice B:

$$\begin{bmatrix} 1 & 2 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \end{bmatrix} = 1 \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + 2 \cdot \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \\ 3 \end{bmatrix}$$

Jak vidíme, výsledný vektor je první sloupec výsledné matice. Analogicky pokud bychom vynásobili matici A druhým sloupcem B, dostali bychom druhý sloupec výsledné matice.

$$\begin{bmatrix} 1 & 2 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1 \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + 1 \cdot \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix}$$

Místo toho, abychom jako v případě skalárních součinů uvažovali nad jednotlivými prvky výsledku, dáváme dohromady celé sloupce.[42]

Lineární kombinace řádků

Poslední způsob, který si zde zmíníme, funguje analogicky, jen si teď obě matice „prohodí role“. Budeme uvažovat lineární kombinace řádků matice B, kde jako koeficienty budeme brát prvky z jednotlivých řádkových vektorů A.

Pro první řádek dostáváme tedy:

$$\begin{bmatrix} 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix} = 1 \cdot \begin{bmatrix} 1 & 1 \end{bmatrix} + 2 \cdot \begin{bmatrix} 2 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 3 \end{bmatrix}$$

Pro druhý:

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix} = 0 \cdot \begin{bmatrix} 1 & 1 \end{bmatrix} + 1 \cdot \begin{bmatrix} 2 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 \end{bmatrix}$$

A pro poslední:

[42] Pokud vyjádříme tuto myšlenku slovně pro náš příklad, tak pro výsledek platí:

1. *sloupec* = součet prvního sloupce a dvojnásobku druhého sloupce matice A
2. *sloupec* = součet prvního a druhého sloupce matice A

To nám umožní některé příklady spočítat snadněji, zejména pokud druhá matice obsahuje pouze čísla jako 0 a 1.

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix} = 1 \cdot \begin{bmatrix} 1 & 1 \end{bmatrix} + 1 \cdot \begin{bmatrix} 2 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 2 \end{bmatrix}$$

Opět zkusme i slovní vyjádření:

1. *řádek* = součet prvního řádku a dvojnásobku druhého řádku matice B
2. *řádek* = druhý řádek matice B
3. *řádek* = součet prvního a druhého řádku matice B

Pro některé příklady nám mohou tato alternativní vyjádření výrazně ušetřit práci. Jejich hlavní smysl ale uvidíme později, v následující kapitole.

Jednotková matice

Násobíme-li reálné číslo jedničkou[43], výsledek se nezmění.

[43] Formálně se bavíme o neutrálním prvku, v případě násobení pak také používáme označení jednotkový prvek.

$$x \cdot 1 = 1 \cdot x = x$$

Pro maticové násobení můžeme nalézt podobnou „jedničku“. Hledáme tedy matici I, pro kterou platí:

$$A \cdot I = I \cdot A = A$$

nezávisle na konkrétní matici A. Jakou má mít matice I podobu, aby výše zmíněná rovnost byla splněna, není těžké určit, podíváme-li se na toto násobení ($A \cdot I = A$) jako na lineární kombinaci sloupců matice A.

Aby první sloupec výsledku byl stejný jako první sloupec A, musí být v prvním sloupci I první prvek 1 a všechny ostatní 0. Pro druhý sloupec potřebujeme jedničku na druhé pozici a všude jinde nuly, atd.

Matice I[44] se třemi řádky a třemi sloupci pak vypadá takto:

[44] Označení I vychází z anglického slova *Identity*.

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Zobecnění pro jiné rozměry není jistě problém. V tuto chvíli si můžete ověřit, že i pro ostatní způsoby násobení matic, které jsme uvedli dříve, tento tvar vyhovuje.

V Pythonu vytvoříme jednotkovou matici pomocí funkce `np.eye(n)`, kde `n` označuje její velikost.

Všimněme si navíc, že to samé platí i při násobení jednotkové matice vektorem:

$$I \cdot v = v$$

Inverzní matice

Zůstaňme ještě chvíli u analogie s reálnými čísly a připomeňme si pojem převráceného čísla (hodnoty). Číslo y je převrácené k x, pokud platí, že $xy = 1$. Zjevně tedy musí platit, že $y = 1/x = x^{-1}$:

$$x \cdot \frac{1}{x} = x \cdot x^{-1} = 1$$

U matic je situace komplikovanější. Jednotkovou matici máme, nicméně dělení matic (jako I/A) definováno není. Musíme si pomoci jinou definicí.

INVERZNÍ MATICI k A označme A^{-1} a definujme vztahem:[45]

$$A \cdot A^{-1} = I$$

Je poměrně zjevné, že jednotková matice je sama sobě inverzí, platí $I^{-1} = I$.[46]

Jak určit obecně inverzní matici není úplně triviální. Zkusme postupně určit inverzní matice k těmto příkladům:[47]

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}$$

Nalézt A^{-1} není těžké. Snažíme se vlastně vyřešit následující rovnici pro neznámé a_1, a_2, a_3, a_4:

$$\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Ať už zvolíme libovolnou metodu (roz)násobení, snadno určíme:[48]

$$A^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1/2 \end{bmatrix}$$

Ve druhém případě už je situace složitější, protože máme nenulové prvky i mimo diagonálu:

$$\begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Jedním ze způsobů může být rozdělení této rovnosti na dvě jednodušší, využijeme-li lineárních kombinací.

$$\begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} b_2 \\ b_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Tyto dva systémy už vyřešíme třeba Gassovou eliminační metodou a získáme:[49]

$$B^{-1} = \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix}$$

[45] Pokud se bavíme o inverzních maticích, uvažujeme vždy pouze čtvercové matice.

[46] To vyplývá z vlastností násobení, přesvědčte se o tom.

[47] V Pythonu můžeme pro výpočet inverzní matice použít funkci `np.linalg.inv`.

[48] Obecně platí, že inverzní matice k *diagonální matici* (= má nenulové prvky pouze na diagonále), je opět diagonální matice – prvky diagonály jsou převrácené hodnoty původních.

[49] Ověřte, že platí $B \cdot B^{-1} = I$.

Třetím příklad pak začneme řešit analogicky jako druhý:

$$\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} c_2 \\ c_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Když se ovšem podíváme na sloupce matice C, můžeme si všimnout, že jeden je násobkem druhého. Oba vektory leží v jedné přímce a zcela jistě z nich nelze pomocí lineární kombinace vyrobit ani jeden ze sloupců jednotkové matice.[50] Inverzní matice C^{-1} tedy neexistuje.

Zkusme nyní zobecnit závěry, které jsme učinili na předchozích příkladech. Aby měla matice svou inverzi, musí její sloupce tvořit vektory, kterými lze vyjádřit libovolný sloupec jednotkové matice. A protože uvažujeme pouze čtvercové matice, musí podle poznatků z předchozí kapitoly platit, že takové vektory jsou lineárně nezávislé.

Do souhrnu vlastností z předchozí kapitoly tak můžeme přidat následující větu:

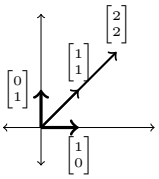
MATICE MÁ INVERZI, pokud je regulární. Jinými slovy, pokud je A tvořena lineárně nezávislými vektory, A^{-1} existuje.

Známe-li inverzní matici, můžeme ji použít při řešení systému lineárních rovnic, jak ukazuje následující odvození:[51]

$$\begin{aligned} Ax &= b \\ A^{-1}Ax &= A^{-1}b \\ Ix &= A^{-1}b \\ x &= A^{-1}b \end{aligned}$$

Určit inverzní matici je obecně časově náročné. Ve speciálních případech to může být ovšem jednodušší, jak ukáže následující kapitola. V té přiřkneme některým maticím speciální význam.

[50] Připomeňme si tuto situaci obrázkem:



[51] Obě strany rovnice vynásobíme zleva A^{-1}. Popřemýšlejte nad analogií s reálnými čísly.

Poznamenejme, že reálně v počítači se tento postup (s explicitním vyjádřením inverzní matice) nepoužívá kvůli potenciálním problémům s numerickou stabilitou.

Transformace

V minulé kapitole jsme se naučili násobit matice několika různými způsoby. Nyní si ukážeme, že matice mohou mít při násobení zvláštní význam – fungují jako speciální „funkce“, které nám vhodným způsobem transformují data.[52]

[52] Matice tedy můžeme využít na dvou úrovních – pro vyjádření konkrétní transformace a jako příhodnou reprezentaci vlastních dat.

V příkladech dále budeme nějakou vhodnou metodou transformovat matici A. Připomeňme, že násobíme-li ji nějakou transformační maticí T zleva, interpretuje násobení jako lineární kombinace řádků matice A, násobíme-li ji naopak zprava, pracujeme s lineární kombinací sloupců A.

Zaměřme se nejprve na první případ, kdy se vrátíme ke Gaussově eliminační metodě a odvodíme si známý postup pro výpočet inverzní matice.

Gaussova eliminace a maticové násobení

Gaussova eliminační metoda je založena na eliminaci prvků pod hlavní diagonálou. K tomu máme k dispozici tzv. elementární řádkové úpravy, tedy operace, které pracují na úrovni jednotlivých řádků a nemění řešení daného systému rovnic. Jsou to:

- prohození pořadí dvou řádků
- vynásobení libovolného řádku nenulovým číslem
- přičtení k-násobku jednoho řádku k jinému

Ukažme si nyní, jak lze všechny operace realizovat jako maticové násobení.

Prohození řádků

Chceme-li odvodit, jak má vypadat konkrétní matice realizující požadovanou transformaci, je vždycky užitečné začít maticí jednotkou a tu postupně modifikovat. Ukažme si to na příkladu matice o velikosti 3×3:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot A$$

Jak víme, budeme-li násobit jednotkovou maticí, dostaneme jako výsledek opět A. Zkusme si slovy vyjádřit, co toto násobení představuje. V tuto chvíli děláme lineární kombinace řádků matice A, takže první řádek jednotkové matice $\begin{bmatrix}1 & 0 & 0\end{bmatrix}$ vlastně říká: „opiš první řádek“. Analogicky pak funguje druhý a třetí.

V tuto chvíli by mělo být snadné odvodit, jak bude vypadat matice, která by v matici A prohodila např. první a druhý řádek. Stačí pouze prohodit tyto řádky v jednotkové matici a dostáváme požadovaný tvar.[53]

[53] Pokud vám to není intuitivně jasné, zkuste si vyzkoušet násobení na nějakém příkladu s konkrétní maticí A.

$$\begin{bmatrix}0 & 1 & 0\\1 & 0 & 0\\0 & 0 & 1\end{bmatrix}$$

Tímto způsobem jsme schopni realizovat libovolné prohození řádků. Matice, které toto prohození realizují se nazývají *permutační*.

Pro daný rozměr existuje $n!$ různých permutačních matic[54], pro 3×3 je jich tedy celkem 6:

[54] To by nemělo být překvapivé, vzpomeneme-li si na kombinatoriku a pojem permutace, tedy počet různých uspořádání. V tomto případě pořadí řádků jednotkové matice:

- 1, 2, 3
- 1, 3, 2
- 2, 1, 3
- 2, 3, 1
- 3, 1, 2
- 3, 2, 1

$$\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\0 & 0 & 1\\0 & 1 & 0\end{bmatrix}\begin{bmatrix}0 & 1 & 0\\1 & 0 & 0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}0 & 1 & 0\\0 & 0 & 1\\1 & 0 & 0\end{bmatrix}\begin{bmatrix}0 & 0 & 1\\1 & 0 & 0\\0 & 1 & 0\end{bmatrix}\begin{bmatrix}0 & 0 & 1\\0 & 1 & 0\\1 & 0 & 0\end{bmatrix}$$

Pojďme si nyní zamyslet, jak je to s inverzí k permutační matici. Inverzní matice realizuje inverzní transformaci, tedy v tomto případě „vrací“ (prohazuje) řádky A do původní podoby. Inverze permutační matice je tedy také permutační matice.[55]

[55] Přesvědčte se o tom sami a pro matice výše určete jejich inverze (samozřejmě bez počítání).

Zmíníme zde ještě poslední vlastnost, a to je součin dvou permutačních matic. Asi nás už nepřekvapí, že výsledkem bude opět permutační matice. Přesvědčit se o tom můžeme jednoduše několika způsoby. Uvažme následující součin:

$$P_2 \cdot P_1 \cdot A$$

kde P_2 a P_1 jsou nějaké permutační matice. Označme navíc A' jako výsledek prvního násobení s maticí A:

$$A' = P_1 \cdot A$$

A' je tedy A s nějak prohozenými řádky. Analogicky $A'' = P_2 \cdot A'$, tedy opět A s (obecně) jinak prohozenými řádky.

Protože je násobení matic asociativní (= nezávisí na pořadí uzávorkování), můžeme psát:

$$(P_2 \cdot P_1) \cdot A = A''$$

Odtud vidíme, že součin $P_2 \cdot P_1$ musí být opět matice, která prohazuje řádky v A, tedy permutační.[56]

[56] Všimněme si, že jsme při tomto odvození vůbec nemuseli vědět, jak permutační matice vypadají.

Vynásobení řádku

Využijeme-li poznatky z předchozí sekce, bude už velmi snadné vyrobit matici, která realizuje druhou ekvivalentní úpravu, tedy vynásobení libovolného řádku nenulovým reálným číslem.

Uveďme příklad, kdy chceme vynásobit třetí řádek dvěma:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

V rámci jedné matici můžeme vyjádřit i více těchto úprav najednou. To je dáno tím, že jednotlivé úpravy jsou na sobě nezávislé. Tedy například následující matice vynásobí každý řádek konkrétním hodnotou na diagonále:

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

Najít inverzi je podobně snadné:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/2 \end{bmatrix}$$

Přičtení k*-násobku řádku k jinému*

Poslední z úprav už by měla být opět poměrně jednoduchá. Ukažme příklad matice, která přičte dvojnásobek prvního řádku ke třetímu:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 0 & 1 \end{bmatrix}$$

Pokud modifikujeme řádky nezávisle na sobě, můžeme opět více operací snadno vyjádřit jednou maticí.

Inverzní transformace musí od třetího řádku dvojnásobek prvního naopak odečíst:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & 0 & 1 \end{bmatrix}$$

Operace nad sloupci

Ještě než pokročíme dále, je vhodné zmínit, že všechny uvedené postupy můžeme aplikovat nejen na řádky matice, ale i na její sloupce. Transformačními maticemi pak musíme ale násobit zprava.

Srovnejme například následující dva výrazy. V prvním získáme matici A s prohozeným prvním a druhým řádkem, ve druhém dojde k prohození prvního a druhého sloupce:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot A \qquad \text{vs.} \qquad A \cdot \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Algoritmus pro výpočet inverzní matice

V předchozích příkladech jsme si ukázali, že při určení inverzní matice nemusíme mnohdy počítat, uvědomíme-li si, jakou transformaci daná

matice reprezentuje. Tu pak pouze „otočíme“. Samozřejmě ne ve všech případech to ale půjde takto jednoduše.

Už v minulé kapitole jsme naznačili, jak určit inverzní matici pro libovolnou regulární matici. Představme si nyní alternativní vyjádření stejné myšlenky jako jednoduchý algoritmus.

Napíšeme do jedné „větší“ matice původní matici A a vedle ní navíc jednotkovou matici:

$$\left[\begin{array}{c|c} A & I \end{array}\right]$$

Nyní použijeme Gaussovu eliminaci na celou tuto rozšířenou matici. Postupně ji elementárními úpravami modifikujeme, aby v první části vznikla jednotková matice. Jakmile se nám to podaří, v druhé části bude původní jednotková matice transformována na A^{-1}:[57]

$$\left[\begin{array}{c|c} A & I \end{array}\right] \sim \ldots \sim \left[\begin{array}{c|c} I & A^{-1} \end{array}\right]$$

[57] Jedná se vlastně o rozšíření Gaussovy eliminace, kdy máme více pravých stran. Tyto pravé strany jsou jednotlivé sloupce jednotkové matice:

$$A \cdot X = I$$

Zkusme nyní tento postup vysvětlit s užitím transformačních matic realizujících elementární operace nad řádky. Uvažme posloupnost těchto operací, které musíme v průběhu transformace $A \sim \ldots \sim I$ použít. Každou z nich umíme vyjádřit pomocí nějaké matice, které označíme postupně $E_1, E_2, \ldots, E_n$. Pak celý postup můžeme vyjádřit jako:

$$E_n \cdot \ldots \cdot E_2 \cdot E_1 \cdot \left[\begin{array}{c|c} A & I \end{array}\right]$$

Protože zmíněná posloupnost operací transformuje A na I, platí (pro první část rozšířené matice):

$$(E_n \cdot \ldots \cdot E_2 \cdot E_1) \cdot A = I$$

A z toho nutně vyplývá, že součin těchto matic E_i musí být A^{-1}. Protože ale při tomto násobení aplikujeme stejné operace i na jednotkovou matici v druhé části, dostáváme naše zdůvodnění:

$$(E_n \cdot \ldots \cdot E_2 \cdot E_1) \cdot I = A^{-1} \cdot I = A^{-1}$$

Aby byla tato myšlenka úplně zřejmá, demonstrujme celý postup na následující matici:

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

Utvoříme rozšířenou matici a budeme ji postupně upravovat. V prvním kroku odečteme první řádek od druhého. Ve druhém pak druhý od prvního:

$$\left[\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 1 & 2 & 0 & 1 \end{array}\right] \to \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \cdot \left[\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 1 & 2 & 0 & 1 \end{array}\right] = \left[\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 1 \end{array}\right]$$

$$\left[\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 1 \end{array}\right] \to \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \cdot \left[\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 1 \end{array}\right] = \left[\begin{array}{cc|cc} 1 & 0 & 2 & -1 \\ 0 & 1 & -1 & 1 \end{array}\right]$$

A zde už přímo vidíme hledanou inverzi:[58]

$$A^{-1} = \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix}$$

[58] Uvědomte si, že tato inverze skutečně vznikla jako součin matic realizujících obě použité elementární operace.

LU dekompozice

Jak už jsme zmínili dříve, Gaussova eliminace se pro řešení systému lineárních rovnic v reálu přímo nepoužívá. Místo ní se využívají různé alternativní přístupy, které povětšinou staví na nějakém rozkladu matice A na součin jiných matic, které mají určité speciální vlastnosti.[59] V této části si představíme myšlenku asi nejjednodušší takové dekompozice.

Matici A vyjádříme jako součin dolní trojúhelníkové matice L a horní trojúhelníkové matice U.[60] Tyto matice mají nuly pouze nad, resp. pod hlavní diagonálou.

Cílem je najít matice, pro které platí:

$$A = L \cdot U$$

Máme-li tento rozklad, transformujeme $Ax = b$ na ekvivalentní systém:

$$LUx = b$$

Označme $y = Ux$. Pak řešení nového systému spočívá ve dvou krocích:

1. Výpočet „pomocného“ vektoru y:

$$Ly = b$$

2. Výpočet vlastního řešení, vektoru x:

$$Ux = y$$

Může se zdát, že nahradit řešení jednoho systému lineárních rovnic *dvěma* jinými systémy rovnic není úplně efektivní. Uvědomme si ale, že výpočet těchto systémů je už ale díky speciálnímu tvaru matic L a U výrazně jednodušší.[61]

Ukažme si nyní jednoduchý způsob, jak LU dekompozici provést pro následující matici A:

[59] Na takovém principu je založena i metoda `np.linalg.solve` z knihovny NumPy, kterou už jsme dříve několikrát použili pro řešení systému lineárních rovnic.

[60] Pojmenování L a U vycházejí z anglických slov *lower* a *upper*.

[61] Poznamenejme, že pro matici A o rozměrech $n \times n$ zabere Gaussova eliminace řádově n^3 kroků. Pokud je ale matice trojúhelníková, jde to výrazně rychleji, stačí už jich řádově pouze n^2.

V praxi není neobvyklé se setkat se soustavami rovnic, které mají tisíce neznámých.

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 2 & 4 & 2 \\ 1 & 3 & 1 \end{bmatrix}$$

Tuto matici budeme postupně upravovat na horní trojúhelníkovou pomocí elementárních řádkových úprav (jako jsme to dělali při výpočtu inverzní matice v předchozí části). Cílem je tedy vynulovat prvky pod hlavní diagonálou. Nejdříve eliminuje prvky v prvním sloupci:

$$E_1 \cdot A = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 \\ 2 & 4 & 2 \\ 1 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 2 & 2 \\ 0 & 2 & 1 \end{bmatrix}$$

Druhým krokem je pak eliminace prvku pod diagonálou v druhém sloupci:

$$E_2 \cdot (E_1 \cdot A) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 \\ 0 & 2 & 2 \\ 0 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 2 & 2 \\ 0 & 0 & -1 \end{bmatrix}$$

V tuto chvíli se nám podařilo A transformovat na horní trojúhelníkovou matici, máme tedy:

$$E_2 \cdot E_1 \cdot A = U$$

Druhým krokem je z této rovnosti vyjádřit A. To uděláme tak, že obě strany rovnice vynásobíme postupně inverzemi k E_2 a E_1. Zkusme tyto kroky rozepsat:

$$\begin{aligned} E_2 \cdot E_1 \cdot A &= U \\ E_1^{-1} \cdot E_2^{-1} \cdot E_2 \cdot E_1 \cdot A &= E_1^{-1} \cdot E_2^{-1} \cdot U \end{aligned}$$

Nicméně samozřejmě platí, že $E_i \cdot E_i^{-1} = I$. Navíc násobení jednotkovou maticí můžeme vypustit, takže rovnost postupně upravíme:

$$\begin{aligned} E_1^{-1} \cdot I \cdot E_1 \cdot A &= E_1^{-1} \cdot E_2^{-1} \cdot U \\ E_1^{-1} \cdot E_1 \cdot A &= E_1^{-1} \cdot E_2^{-1} \cdot U \\ I \cdot A &= E_1^{-1} \cdot E_2^{-1} \cdot U \\ A &= E_1^{-1} \cdot E_2^{-1} \cdot U \end{aligned}$$

Zbývá nám ještě určit inverze k maticím E_i. To je ale snadné, pokud si uvědomíme, co představují. U E_1 šlo o odečtení prvního řádku (v příslušných násobcích) od druhé a třetího, u E_2 pak o odečtení druhého řádku od třetího.

Inverzní matice tedy musí realizovat opak těchto operací:

$$E_1^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \qquad E_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Posledním krokem je už pouze tyto matice vynásobit, což je jednoduché, protože obě obsahují poměrně hodně nul. Dostáváme hledanou matici L.[62]

[62] Všimněme si, že všechny matice E_i byly samy o sobě dolní trojúhelníkové. Inverze dolní trojúhelníkové matice je opět dolní trojúhelníková matice. To stejné platí pro součin těchto matic.

$$E_1^{-1} \cdot E_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} = L$$

Pro úplnost ještě napišme kompletní rozklad matice A:

$$A = L \cdot U$$

$$\begin{bmatrix} 1 & 1 & 0 \\ 2 & 4 & 2 \\ 1 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 \\ 0 & 2 & 2 \\ 0 & 0 & -1 \end{bmatrix}$$

V závěru ještě poznamenejme, že v praxi se používají vzorce na přímý výpočet jednotlivých prvků L a U. Zde uvedený postup je spíše demonstrační.

Připojme ještě pár rozšiřujících poznámek.

- Dekompozice matic se používají napříč lineární algebrou, nemusí jít nutně o LU dekompozici a řešení systému lineárních rovnic.
- Má-li matice A nějaký speciální tvar (např. je symetrická), můžeme využít dekompozici, která bude efektivnější pro výpočet.
- LU dekompozici lze provést pro každou regulární matici A. Pokud by bylo potřeba v průběhu výpočtu prohodit nějaké řádky, bude obecný tvar vypadat takto:

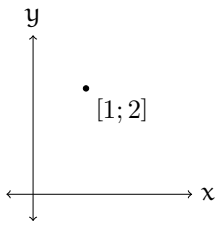
$$P \cdot A = L \cdot U$$

kde P je nějaká permutační matice.

Geometrické transformace

Druhou část této kapitoly věnujeme geometrickým aplikacím násobení matic. Ukážeme si několik jednoduchých transformací, které tvoří základ pro operace například v počítačové grafice, úpravě fotografií, ale i jinde.

Mějme bod v rovině o souřadnicích $[1; 2]$:



Ať už půjde o to vyjádřit libovolnou transformaci, je vždy vhodné zvolit pro začátek jednotkovou matici a tu nějakým vhodným způsobem upravit. Jednotková matice totiž představuje i zde identickou transformaci. Uložíme-li si tedy souřadnice našeho bodu do vektoru v[63], zjevně platí:

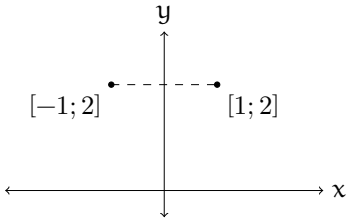
[63] Budeme-li chtít pracovat s více body najednou, můžeme každý z nich reprezentovat jako jeden sloupec matice. Všechny operace pak zůstávají v nezměněné podobě.

$$I \cdot v = v$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Zrcadlení

Chceme-li například realizovat zrcadlení (= osovou souměrnost) podle osy y, musíme si uvědomit, jak se mají složky našeho vektoru změnit.



Z obrázku je zřejmé, že potřebujeme pouze zařídit, aby se u první složky otočilo znaménko. To už v tuto chvíli vede na triviální úpravu jednotkové matice:

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$$

Takovou matici nazvěme M_y.[64] Pokud se zamyslíme nad její inverzí, tak ta je triviální:

[64] Z angl. *mirroring*.

$$M_y^{-1} = M_y$$

Škálování

Škálování ve směru osy x nebo y už opět není nic nového, příslušnou matici a její inverzi už zde píšeme pouze pro úplnost.

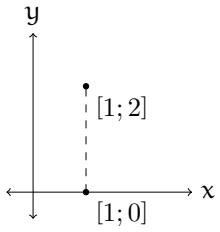
$$S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}, \qquad S^{-1} = \begin{bmatrix} 1/s_x & 0 \\ 0 & 1/s_y \end{bmatrix}$$

Všechny transformace lze samozřejmě zobecnit i pro vyšší dimenze. Pro pochopení hlavních myšlenek nám ovšem stačí pohybovat se ve dvourozměrném prostoru (= rovině).

Projekce

U metody nejmenších čtverců jsme se setkali s pojmem projekce. Šlo tehdy o projekci vektoru do roviny generované zadanými vektory. Zde si zmíníme pouze jednodušší příklad projekce na souřadné osy.

Schematicky jde o velmi jednoduchou operaci. Ukažme si projekci na osu x:



Dochází tedy k „vynulování“ souřadnice y, což realizuje následující matice P_x:

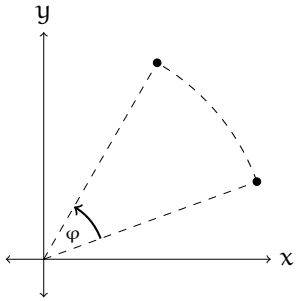
$$P_x = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

Jak je to s inverzí v tomto případě? Zmíněná projekce „zahazuje“ hodnotu jedné souřadnice, z nových souřadnic tak nemůžeme odvodit ty původní. Řečeno neformálně jinak, projekce je *ztrátová* transformace. Inverze tedy neexistuje.[65]

[65] Možností, jak toto ukázat formálně, máme více. Postačí si třeba uvědomit, že P_x je singulární matice, protože má lineárně závislé sloupce.

Rotace

Poslední z transformací, kterou si zde popíšeme, bude rotace okolo počátku soustavy souřadnic. Tu si můžeme znázornit následovně:



Odvodit matici pro rotaci o úhel φ už by bylo poměrně netriviální,[66] uvedeme pouze její konečnou podobu:

[66] Pro její určení je potřeba si opět rozepsat, co se děje s jednotlivými souřadnicemi a využít při úpravách využít některé goniometrické vztahy pro sinus a cosinus součtu.

$$R_\varphi = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}$$

Z předchozích částí je zřejmé, že můžeme transformace aplikovat postupně. Intuitivně už vidíme, že například platí:[67]

[67] Všimněme si, že jsme opět vůbec nepotřebovali znát, jak konkrétně matice R vypadají, využili jsme pouze informaci o tom, co dané matice představují.

$$R_\varphi \cdot R_\psi = R_{\varphi+\psi}$$

Konečně inverzní matice je také snadná, triviálně platí:[68,69]

[68] V poslední úpravě využijeme toho, že $\cos\varphi$ je sudá funkce a naopak $\sin\varphi$ je lichá.

[69] Přesvědčte se násobením, že platí:

$$R_\varphi \cdot R_{-\varphi} = I$$

$$R_\varphi^{-1} = R_{-\varphi} = \begin{bmatrix} \cos(-\varphi) & -\sin(-\varphi) \\ \sin(-\varphi) & \cos(-\varphi) \end{bmatrix} = \begin{bmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{bmatrix}$$

Homogenní souřadnice

Možná jste si všimli, že jsme nezmínili možná nejjednodušší operaci, prosté posunutí ve směru daném nějakým vektorem t (*translace*). Je to proto, že ji nelze přímočaře vyjádřit pomocí násobení matic, jako tomu bylo u předchozích příkladů.

Samozřejmě, že můžeme použít prosté sečtení $v + t$, ale to má dvě nevýhody. Prvně ji nemůžeme přímo aplikovat na více bodů najednou. Druhý důvod je ale závažnější. Pokud skládáme transformace

za sebou, chceme, abychom jako výsledek dostali jednu matici, která realizuje všechny transformace „najednou“. To by se nám ovšem nepodařilo, pokud by všechny operace nebyly vyjádřeny jako násobení matic.

Řešení je v použití tzv. *homogenních souřadnic*. Každý bod o původních souřadnicích $[x; y]$ můžeme reprezentovat vektorem s o jedna větší dimenzí:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Chceme-li odvodit, jak by měla vypadat matice translace o $[t_x; t_y]$, znamená to vyřešit tento systém rovnic:

$$T \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

Rozepišme si tuto rovnost jako lineární kombinaci sloupců matice T, ty označme třeba u, v, w:

$$x \cdot \begin{bmatrix} | \\ u \\ | \end{bmatrix} + y \cdot \begin{bmatrix} | \\ v \\ | \end{bmatrix} + 1 \cdot \begin{bmatrix} | \\ w \\ | \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

Protože se na pravé straně objevuje x pouze v první složce a y ve druhé, zvolme u a v následovně:[70]

[70] Jde samozřejmě o první sloupce jednotkové matice.

$$x \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + y \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + 1 \cdot \begin{bmatrix} | \\ w \\ | \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

V tuto chvíli už je jasně vidět, jak musí vypadat poslední vektor:

$$w = \begin{bmatrix} t_x \\ t_y \\ 1 \end{bmatrix}$$

Celkově dostáváme tedy matici posunutí:[71]

[71] Inverzi už explicitně zmiňovat nebudeme, ta by měla být v tuto chvíli zřejmá.

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Ostatní operace

S výjimkou translace jsme všechny dřívější operace uvedli ve standardních souřadnicích. Nabízí se přirozená otázka, jak budou vypadat tyto transformace v homogenních souřadnicích.

Uvažme nějakou matici transformace ve standardních souřadnicích. Obecně ji vyjádřeme jako:

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}$$

Pak její ekvivalent v homogenních souřadnicích bude tento:

$$\begin{bmatrix} a_1 & a_2 & 0 \\ a_3 & a_4 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Zdůvodnění už si můžete provést sami, stačí se zamyslet, jaký bude výsledek násobení vektorem ve standardních souřadnicích v prvním případě, resp. v homogenních ve druhém.

Navíc nyní zvládneme snadno reprezentovat i rotaci kolem libovolného bodu, nejen počátku. Získáme ji součinem tří matic, které postupně provedou:

1. posunutí bodu do počátku
2. vlastní rotace o daný úhel
3. posunutí zpátky

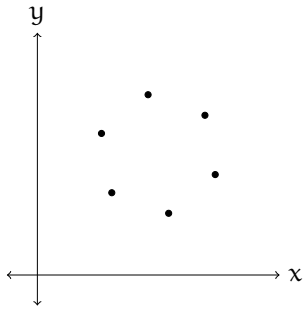
Redukce dimenzí

Při zpracování a analýze dat je často velmi užitečné je umět vhodným způsobem vizualizovat. Uvažme například následující matici:

$$\begin{bmatrix} 2{,}77 & 1{,}83 & 1{,}06 & 1{,}23 & 2{,}17 & 2{,}94 \\ 2{,}64 & 2{,}98 & 2{,}34 & 1{,}36 & 1{,}02 & 1{,}66 \end{bmatrix}$$

Na první pohled náhodná čísla však při vhodném způsobu zobrazení mohou odhalit nějakou skrytou strukturu, která nám pomůže pochopit, co data vyjadřují. Pokud bude každý sloupec zmíněné matice vyjadřovat souřadnice jednoho bodu, dostáváme tento obrázek:

Že je vizuální vnímání pro nás přirozenější není překvapivé. To se u (předchůdců) člověka formovalo miliony let, symbolické vyjadřování (třeba využití číslic) používáme naproti tomu „pouze“ několik tisíc let.

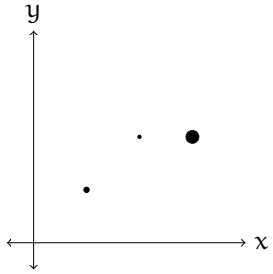


Situace se ovšem zkomplikuje, pokud bychom měli data s vyšší dimenzionalitou. 3D data můžeme ještě zakreslit v rámci nějakého promítnutí prostoru, ale je otázka, do jaké míry bude podobný obrázek přehledný.

Zmiňme ještě několik možností. Například jeden rozměr můžeme vyjádřit jako velikost bodu:

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 2 \\ 5 & 3 & 12 \end{bmatrix}$$

V tomto případě první dva rozměry zobrazíme jako souřadnice x, y, třetí rozměr pak udává poloměr bodu:



Pro kategorické atributy[72] bychom mohli zvolit odlišení užitím různých barev nebo symbolů pro body, nicméně je zřejmé, že tady pomalu naše možnosti končí.

V praxi totiž není neobvyklé pracovat s daty, která mají desítky, ale klidně i stovky dimenzí. V této kapitole si představíme myšlenku základního postupu[73], který dokáže snížit dimenzionalitu dat při zachování co největšího množství informace (= struktury, charakteristik dat).

[72] Kategorické atributy nabývají pouze konečného množství diskrétních hodnot. Objekty tak můžeme rozdělit do skupin na základě hodnot těchto atributů.

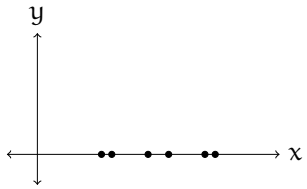
[73] Celý postup není úplně triviální, my se zaměříme jen na jednu jeho část.

Triviální způsob redukce dimenzí je prosté vynechání některých souřadnic.[74] To může být užitečné, pokud víme, že vynecháváme souřadnici, která nemá vliv na to, co zkoumáme. To ale samozřejmě vždy nemusí platit a tímto způsobem bychom mohli ztratit potenciálně velké množství informace.

Uvažme úvodní matici po vynechání druhé souřadnice (projekce na osu x). Vizualizujeme-li si ji nyní, získáme místo dříve zjevného pravidelného šestiúhelníku pouze několik bodů na ose x:

[74] Ačkoliv to v tomto případě není nutné, i toto lze vyjádřit užitím maticového násobení. Například redukce 3D →2D vynecháním druhé souřadnice:

$$\begin{bmatrix}1 & 0 & 0\\ 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{bmatrix}=\begin{bmatrix}1 & 2 & 3\\ 7 & 8 & 9\end{bmatrix}$$

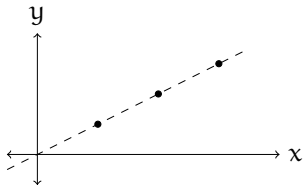


Alternativní reprezentace

Pro některé případy může být užitečné, pokud bychom našli pro zadaná data alternativní vyjádření. Uvažme následující data:

$$\begin{bmatrix}1 & 2 & 3\\ 0{,}5 & 1{,}0 & 1{,}5\end{bmatrix}$$

Vztah mezi nimi je poměrně zjevný, z obrázku navíc můžeme vidět, že všechny leží na přímce procházející počátkem.



Směr této přímky je udán vektorem, který označme v:

$$v=\begin{bmatrix}2\\ 1\end{bmatrix}$$

Nyní je vidět, že stejná data můžeme snadno reprezentovat jako násobky tohoto vektoru. Dostáváme jednodušší vyjádření stejných bodů, dokonce bez ztráty informace:[75]

Jen zmíníme, že někdy může být užitečné reprezentovat data např. v *polárních souřadnicích*. Připomeňme, že u polárních souřadnic používáme místo souřadnic x a y vzdálenost od počátku r a úhel φ.

Základ pro náš „šestiúhelník“ by pak mohl být popsán takto:

$$\begin{bmatrix}1 & 1 & 1 & 1 & 1 & 1\\ 0^\circ & 60^\circ & 120^\circ & 180^\circ & 240^\circ & 300^\circ\end{bmatrix}$$

Abychom dostali vyjádření plně ekvivalentní, museli bychom ještě všechny body vhodně orotovat a posunout.

[75] Všimněme si, že každý bod je nyní reprezentován pouze jednou souřadnicí místo dvou. To může být významné zejména tehdy, máme-li takových bodů mnoho.

$$\begin{bmatrix}2\\1\end{bmatrix} \cdot \begin{bmatrix}1/2 & 1 & 3/2\end{bmatrix}$$

Ukažme si nyní, jak se dají naše úvahy zobecnit. K tomu si ale musíme vysvětlit pojem *báze*.

Báze

Máme-li nějaký bod v rovině, můžeme zapsat jeho souřadnice do vektoru. Např.:

$$w = \begin{bmatrix}2\\3\end{bmatrix}$$

To nevypadá jako nějaká nová informace, klíčové je ale si uvědomit, že dané souřadnice se vztahují vždy k nějakému souřadnému systému, který používáme.[76]

[76] Dříve jsme v poznámce vzpomněli polární souřadnice. Např. vektor $\begin{bmatrix}1 & 1\end{bmatrix}^T$ má v polárních souřadnicích úplně jiný význam.

Zatím vždy (aniž bychom o tom věděli) jsme se pohybovali v tzv. *standardní bázi*, to znamená, že každý vektor byl vyjádřen jako lineární kombinace vektorů, které jsou sloupci jednotkové matice (označujeme je jako $e_1, e_2, \ldots$). Konkrétně pro předchozí vektor to znamená:

$$2e_1 + 3e_2 = 2 \cdot \begin{bmatrix}1\\0\end{bmatrix} + 3 \cdot \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}2\\3\end{bmatrix}$$

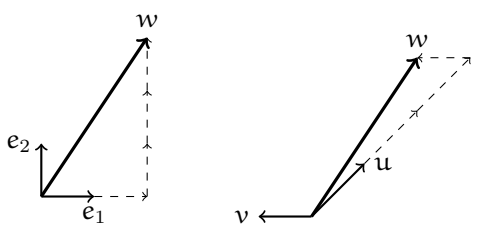
Pokud bychom ale zvolili náš souřadný systém jinak, měl by uvedený vektor souřadnice jiné. Zkusme třeba použít jako bázi následující vektory:

$$u = \begin{bmatrix}1\\1\end{bmatrix}, \quad v = \begin{bmatrix}-1\\0\end{bmatrix}$$

Pak stejný vektor bude mít v této nové bázi souřadnice 3 a 1, protože platí:

$$3u + v = 3 \cdot \begin{bmatrix}1\\1\end{bmatrix} + 1 \cdot \begin{bmatrix}-1\\0\end{bmatrix} = \begin{bmatrix}2\\3\end{bmatrix}$$

Srovnání obou vyjádření pak nabídněme v následujících obrázcích, kde navíc explicitně znázorníme příslušné lineární kombinace bázových vektorů:



Pro vyjádření libovolného vektoru musí být bázové vektory lineárně nezávislé. To už nás asi nepřekvapí, zejména vzpomeneme-li si na diskuzi k řešitelnosti systému lineárních rovnic.

Najít souřadnice vektoru v nové bázi (reprezentované vektory u, v) tedy znamená vyřešit následující systém rovnic. x, y představují souřadnice ve standardní bázi, x', y' jsou naopak ty hledané:[77]

$$\begin{bmatrix} | & | \\ u & v \\ | & | \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

[77] Jinak řečeno hledáme vhodnou lineární kombinaci těchto nových bázových vektorů.

Alternativně můžeme k přímému vyjádření nových souřadnic použít ekvivalentní formu, která vznikne vynásobením obou stran rovnice inverzní maticí zleva:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} | & | \\ u & v \\ | & | \end{bmatrix}^{-1} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

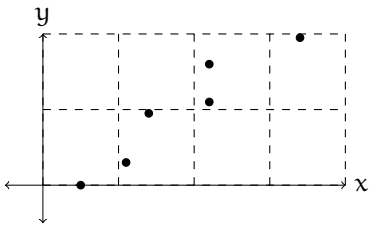
Ztrátové transformace

Tuto kapitolu jsme začali úvahami o vizualizaci vícerozměrných dat. Abychom snížili dimenzi těchto dat, ale zároveň zachovali co nejvíce informace, pomůžeme si změnou báze.

Mějme následující data:

$$D = \begin{bmatrix} 0{,}5 & 1{,}1 & 1{,}4 & 2{,}2 & 2{,}2 & 3{,}4 \\ 0{,}0 & 0{,}3 & 0{,}95 & 1{,}1 & 1{,}6 & 1{,}95 \end{bmatrix}$$

Opět si zkusíme data vykreslit do obrázku a navíc si do něj přidáme mřížku:



Všimněme si, že všechny body leží blízko pomyslné přímky $y = x/2$. Tuto přímku lze reprezentovat pomocí vektoru v:

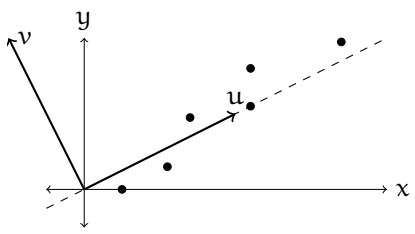
$$v = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

Zvolme ještě vektor u takový, aby byl na v kolmý. Například následující volbou:[78]

[78] Potřebujeme, aby platilo $u^T v = 0$. Stačí tedy prohodit souřadnice a jednu znegovat.

$$u = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

Přidejme nyní vektory u a v k původnímu obrázku:



Naším cílem bude teď vyjádřit tyto body v nové bázi. To znamená, že musíme najít koeficienty lineární kombinace vektorů u a v pro každý bod z D, což odpovídá řešení následujícího systému rovnic:

$$\begin{bmatrix} 2 & 1 \\ 1 & -2 \end{bmatrix} \cdot X = D$$

Pomozme si v tuto chvíli Pythonem, abychom nemuseli tyto výpočty dělat ručně:

```
>>> import numpy as np
>>> D = np.array([[0.5, 1.1, 1.4, 2.2, 2.2, 3.4],\
...               [0.0, 0.3, 0.95, 1.1, 1.6, 1.95]])
>>> A = np.array([[2, 1],[1, -2]])
>>> X = np.linalg.solve(A, D)
>>> X
array([[ 0.2 ,  0.5 ,  0.75,  1.1 ,  1.2 ,  1.75],
       [ 0.1 ,  0.1 , -0.1 , -0.  , -0.2 , -0.1 ]])
```

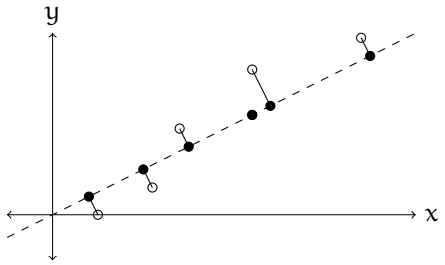
Když se nyní podíváme na souřadnice v nové bázi, můžeme si všimnout, že druhá složka je oproti první výrazně marginální.[79]

[79] To asi není překvapivé, když se znovu podíváme na obrázek výše.

Budeme-li chtít snížit dimenzi dat, může se „vypuštění“ této složky jevit jako rozumný nápad, který nám zachová velké množství původní informace. Opět výpočet realizujme v Pythonu:

```
>>> X[1] = 0
>>> X
array([[0.2 , 0.5 , 0.75, 1.1 , 1.2 , 1.75],
       [0.  , 0.  , 0.  , 0.  , 0.  , 0.  ]])
>>> # Převod do standardní báze
>>> A @ X
array([[0.4 , 1.  , 1.5 , 2.2 , 2.4 , 3.5 ],
       [0.2 , 0.5 , 0.75, 1.1 , 1.2 , 1.75]])
```

Vykresleme si tyto nové body k těm z původního obrázku:



Všimněme si, že jsme vlastně určili projekci všech bodů na přímku $y = x/2$ reprezentovanou vektorem u.[80]

[80] Dodejme, že tímto postupem jsme schopni udělat projekci pouze na přímku procházející počátkem.

V obecném případě by bylo nutné data nejdříve vycentrovat (= zjistit průměrnou hodnotu každé ze souřadnic a tyto průměry pak odečíst od vlastních dat).

Vzpomeňme si nyní na metodu nejmenších čtverců, která je na projekci založena. I zde totiž řešíme systém lineárních rovnic, o kterém víme, že nemá řešení:

$$u \cdot X = D$$

Ověřme, že užitím této metody získáme stejné výsledky:

```
>>> v = np.array([[2, 1]]).T
>>> np.linalg.lstsq(v, D)[0]
array([[0.2 , 0.5 , 0.75, 1.1 , 1.2 , 1.75]])
```

Vektor v, který jsme použili, sice není nejlepší možný (také jsme jej určili „od oka“), ale oproti jednoduché projekci třeba na osu x poskytuje i tak výrazně lepší výsledky.

Množství zachované informace záleží tedy na tom, jakým způsobem zvolíme bázi a kterou souřadnici v reprezentaci podle této nové báze vypustíme.

Jako vektory nové báze tak dává smysl vybírat postupně ty směry, ve kterých mají data největší variabilitu (= nejvíce se mění) a vypouštět je v opačném pořadí, od nejméně významných. Jak tyto směry určit už však výrazně přesahuje zamýšlenou úroveň tohoto textu.[81]

[81] Celý tento proces se pak nazývá *analýza hlavních komponent*, anglicky *principal component analysis* (PCA).

Přidejme ještě odkaz na jedno vynikající vysvětlení celé myšlenky (zejména několik prvních částí).

Dodejme ještě, že obdobné postupy jsou základem některých algoritmů ztrátové komprese.

Základní pojmy matematické analýzy

Doposud jsme se v tomto textu zabývali lineární algebrou, nyní je čas zaměřit se na druhou velkou oblast matematiky – matematickou analýzu. Ta se zabývá studiem funkcí a jejich vlastností. V této kapitole neformálně připomeneme některé základní pojmy z této oblasti.[82]

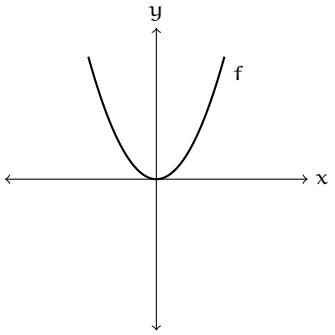
[82] Záměrně zde neuvádíme konkrétní formalismy, ani vzorce a postupy pro řešení různých typů příkladů, ty lze nalézt v materiálech úvodních kurzů.

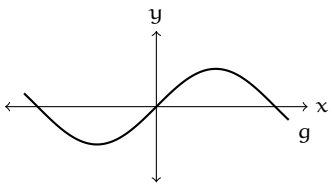
Funkce

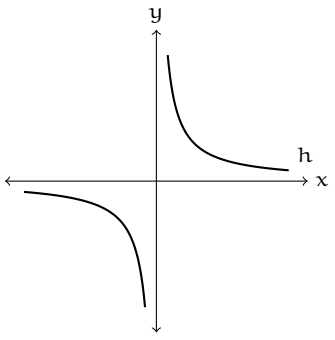
Funkce v matematické analýze je předpis, který nám říká, jakým způsobem závisí hodnota proměnné y na proměnné x. Značíme ji typicky písmenem f (nebo také g, h atp.). Například:

$$\begin{aligned} f:&\quad y = x^2 \\ g:&\quad y = \sin x \\ h:&\quad y = \frac{1}{x} \end{aligned}$$

Pro připomenutí si ukažme grafy těchto funkcí:







Definiční obor funkce je množina všech hodnot, které mohu dosadit za x, aniž by bylo porušeno nějaké matematické pravidlo (typicky například dělení nulou, odmocnina ze záporného čísla,...). Definiční obor se značí písmenem D. Pro naše funkce výše platí:

$$\begin{aligned} D_f &= D_g = \mathbb{R} \\ D_h &= \mathbb{R} \setminus \{0\} \end{aligned}$$

Obor hodnot pak udává, jakých hodnot může nabývat proměnná y, budeme-li do funkce dosazovat x z definičního oboru. Obor hodnot značíme H. Dostáváme:

$$\begin{aligned} H_f &= \mathbb{R}_0^+ \\ H_g &= [-1; 1] \\ H_h &= \mathbb{R} \setminus \{0\} \end{aligned}$$

Grafy některých funkcí mohou být symetrické, ať už okolo osy y nebo počátku soustavy souřadnic. V prvním případě hovoříme o funkcích *sudých*, ve druhém jde o funkce *liché*. Pro takové funkce platí:

$$\begin{aligned} &\text{sudá:} && f(-x) = f(x) \\ &\text{lichá:} && f(-x) = -f(x) \end{aligned}$$

Funkce f je tedy zjevně sudá, funkce g a h jsou pak liché.[83]

[83] Ověřte si dosazením a vizuálně načrtnutím v obrázku.

Limita funkce

Limita je základním stavebním kamenem matematické analýzy. Jedná se o formalismus, který nám umožňuje pracovat s nekonečně malými vzdálenostmi.[84]

[84] Matematické analýze se také říká infinetizimální počet (infinetizimální = nekonečně malý).

Limita popisuje, jak se funkce chová *v okolí* daného bodu. To je pro naše účely užitečné, zejména jedná-li se o body nespojitosti. Uvažme funkci h za začátku kapitoly $\left(y = \frac{1}{x}\right)$. Protože bod $x = 0$ není součástí definičního oboru, nemůžeme v něm určit funkční hodnotu. Můžeme se ale ptát, co se děje, budeme-li se k tomuto bodu přibližovat.[85]

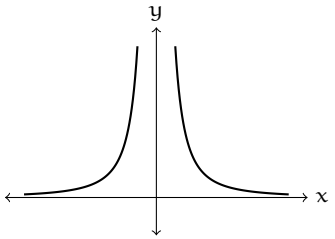
[85] Podívejte se na obrázek na předchozí straně.

Pokud půjdeme zleva, funkční hodnota bude klesat k $-\infty$, naopak při přiblížení zprava roste funkční hodnota nade všechny meze. Tyto skutečnosti můžeme vyjádřit jednostrannými limitami.

$$\begin{aligned} \lim_{x \to 0^-} \frac{1}{x} &= -\infty \\ \lim_{x \to 0+} \frac{1}{x} &= \infty \end{aligned}$$

Protože se tyto jednostranné limity nerovnají, „oboustranná“ limita $\lim_{x \to 0} \frac{1}{x}$ neexistuje.[86]

[86] Srovnejte s funkcí $y = \frac{1}{x^2}$:



Dalším využitím limit je pak asymptotické chování funkcí, tedy informace o tom, jak se mění funkční hodnota pro $x \to \pm\infty$. Mohou nastat tři případy, jak si ukážeme na funkcích f, g a h (zde pouze pro $x \to \infty$).[87]

[87] Podívejte se na obrázky.

$$\begin{aligned} \lim_{x \to \infty} f(x) &= \lim_{x \to \infty} x^2 = \infty \\ \lim_{x \to \infty} g(x) &= \lim_{x \to \infty} \sin x \text{ neexistuje} \\ \lim_{x \to \infty} h(x) &= \lim_{x \to \infty} \frac{1}{x} = 0 \end{aligned}$$

V prvním případě roste funkční hodnota nade všechny meze. Ve třetím dochází k ustálení na hodnotě 0. Ve druhém funkce osciluje, k ustálení nedochází, a limita tak neexistuje.

Derivace funkce

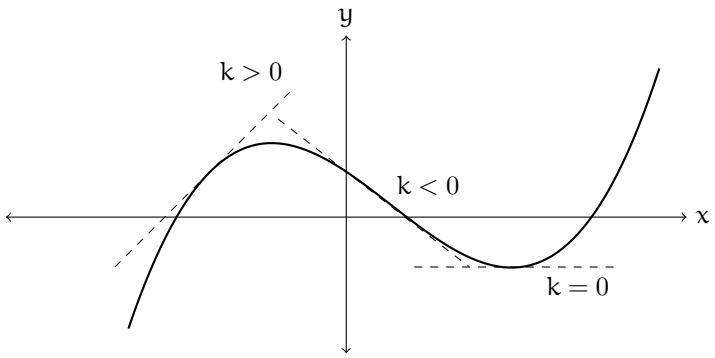
Dalším z klíčových pojmů celé matematické analýzy je derivace. Derivace funkce určuje, jakým způsobem se mění funkční hodnota. Je-li

derivace (v konkrétním bodě) kladná, funkce je tam rostoucí, je-li záporná, funkce klesá.

Pokud je derivace v některém bodě nulová, ke změně funkční hodnoty nedochází. Takovým bodům říkáme stacionární a některé z nich mohou být minimem nebo maximem.

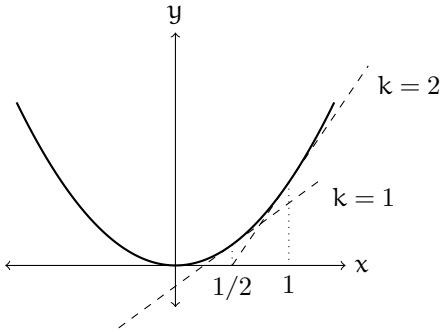
Hodnota derivace v daném bodě pak přesně odpovídá směrnici tečny[88] v tomto bodě.

[88] Připomeňme, že tečna je přímka, lze ji tedy popsat rovnicí $y = kx + q$, kde k je směrnice udávající sklon přímky a q pak označuje posunutí ve směru osy y.



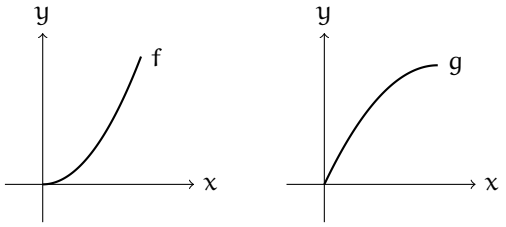
Navíc čím vyšší hodnota derivace (v absolutní hodnotě), tím funkce rychleji roste/klesá. Demonstrujme na příkladu funkce $y = x^2$. První derivaci značíme jako y', konkrétně zde platí $y' = 2x$.

Například v bodě $x = 1/2$ je hodnota derivace $y'(1/2) = 1$, v bodě $x = 1$ dostáváme $y'(1) = 2$. To znamená, že funkce v druhém bodě roste rychleji, což můžeme vidět i na následujícím obrázku:



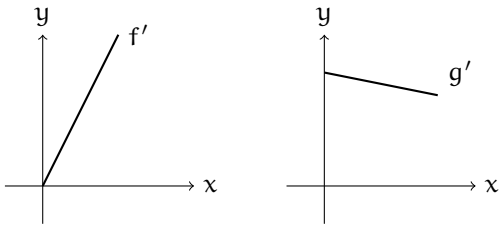
Druhá derivace

Uvažme nyní tyto dvě funkce f a g:

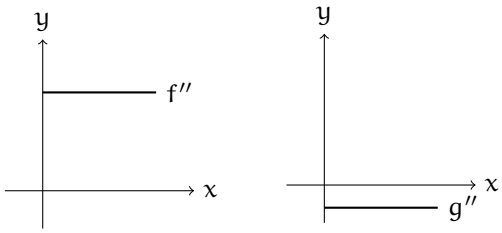


Obě funkce jsou rostoucí, takže jejich první derivace musí být kladná. Nicméně zatímco v případě funkce f se rychlost růstu zvyšuje, u funkce g naopak růst zpomaluje. Geometricky vidíme, že se liší „prohnutí“ obou funkcí.

Pokud funkce f roste čím dál rychleji, znamená to, že první derivace musí být rostoucí funkce.[89] Analogicky, první derivace funkce g musí být klesající funkce. Mohly by vypadat například takto:



Jak vidíme, obě první derivace jsou kladné funkce (jejich grafy leží nad osou x). Protože i první derivace je funkce, můžeme ji opět zderivovat, čímž dostáváme tzv. *druhou derivaci*. Opět si načrtněme obrázky. Víme, že f' je rostoucí funkce, takže její derivace musí být kladná. g' je naproti tomu klesající, její derivace tedy bude záporná.[90]



Funkce, jejichž druhá derivace je kladná, nazýváme *konvexní*, zápornou druhou derivaci mají funkce *konkávní*. Body, ve kterých se mění konvexnost na konkávnost (nebo naopak) nazýváme *inflexní body*.

Asymptoty

Asymptoty funkce jsou přímky, ke kterým se graf funkce blíží. Rozlišujeme asymptoty bez směrnice a se směrnicí. První z nich vznikají v bodech nespojitosti funkce, tedy tam, kde funkce není definována. Jsou to přímky kolmé na osu x, nejde tedy o funkce, protože je nedokážeme vyjádřit ve tvaru $y = kx + q$. Zapisujeme je tedy například jako $x = p$, kde p je bod nespojitosti.

Asymptoty se směrnicí nám pomáhají určit, jak se chová funkce pro $x \to \pm\infty$. Tyto umíme vyjádřit ve tvaru $y = kx + q$. Abychom určili konkrétní koeficienty, musíme spočítat dvě sady limit. Zde uvedeme pouze ty pro $x \to \infty$ (pro $x \to -\infty$ jsou analogické):

$$k = \lim_{x\to\infty} \frac{f(x)}{x}$$
$$q = \lim_{x\to\infty} f(x) - kx$$

Pokud některá z limit neexistuje, nebo její výsledek není reálné číslo, pak příslušná asymptota neexistuje.

Pro funkce f, g a h ze začátku kapitoly má asymptoty pouze funkce h. Asymptota bez směrnice je $x = 0$ (osa y), asymptota se směrnicí je pro $x \to \pm\infty$ shodná, jedná se o přímku $y = 0$, tj. osu x.

[89] Neformálně řečeno, hodnota derivace určuje přírůstek funkční hodnoty, pokud se posuneme o malý kousek na ose x. V případě funkce f se tyto přírůstky zvětšují.

[90] Konkrétní tvary prvních a druhých derivací mohou být samozřejmě různé. V tomto příkladě byly funkce f a g kvadratické, jejich derivace jsou tedy lineární funkce a derivace lineárních funkcí jsou funkce konstantní.

Průběh funkce

V předchozí kapitole jsme si ve stručnosti připomněli několik základních konceptů matematické analýzy. Abychom ukázali, jak spolu tyto pojmy navzájem souvisí, uvedeme v této kapitole jeden příklad, v rámci kterého všechny tyto nástroje využijeme. Zvolený typ příkladu se nazývá *průběh funkce*. Naším cílem bude určit o zadané funkci co nejvíce informací, na jejichž základě budeme schopni načrtnout její graf.

Zvolme pro ukázku následující funkci f:

$$f: \quad y = \frac{e^x}{x}$$

Základní vlastnosti

V první části naší analýzy se zaměříme na jednoduché charakteristiky, které můžeme odvodit přímo z předpisu funkce.

Definiční obor

Nejdříve ze všeho určíme definiční obor. Protože funkce e^x je definována pro libovolné x, jediným problematickým bodem funkce f bude $x = 0$, protože bychom při dosazení dostali nepovolené dělení nulou. Platí tedy:

$$D_f = \mathbb{R} \setminus \{0\}$$

Obor hodnot

Mohli bychom v tuto chvíli zkusit určit i obor hodnot, ale to je výrazně snazší, máme-li načrtnut obrázek. Tuto část tedy necháme na konec.

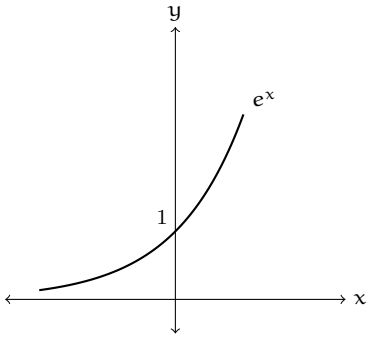
Parita funkce

Abychom zjistili, jestli je funkce sudá nebo lichá (nebo ani jedno), dosadíme do funkčního předpisu všude $-x$ místo původního x. Dostáváme:

$$f(-x) = \frac{e^{-x}}{-x}$$

Zjevně můžeme říct, že $f(-x) \neq f(x)$, ani $f(-x) \neq -f(x)$. Funkce f tedy není ani sudá, ani lichá.

Abychom mohli určit vlastnosti naší zadané funkce, musíme znát vlastnosti jednotlivých elementárních funkcí, jako je třeba e^x. Připomeňme si její graf:



Průsečíky s osami

Průsečíky se souřadnými osami nám mohou pomoci při kreslení grafu. Zatímco průsečíků s osou x může být obecně více, průsečík s osou y může být pouze jeden.[91]

[91] Zamyslete se proč.

Abychom určili průsečíky s osou x (značíme P_x), musíme vyřešit rovnici[92]:

[92] Průsečíky s osou x mají tu vlastnost, že jejich y-ová souřadnice má hodnotu 0, proto položíme $y = 0$.

$$0 = \frac{e^x}{x}$$

Tato rovnice zjevně nemá řešení, protože funkce e^x nikdy nemůže být nula.[93] Průsečíky s osou x tedy neexistují.

[93] Platí, že pro libovolné x je $e^x > 0$. Zkontrolujte s obrázkem výše.

Analogicky, průsečík s osou y je takový bod (P_y), jehož x-ová souřadnice je 0. Protože ale 0 není součástí definičního oboru, ani P_y pro naši funkci neexistuje.

Znaménko funkce

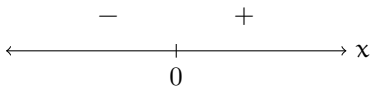
V závěru první části ještě určíme, kdy je graf funkce f nad, resp. pod osou x. Vyjádřeno jinak, ptáme se, pro která x platí, že $\frac{e^x}{x} > 0$, resp. $\frac{e^x}{x} < 0$.

Klíčové je si uvědomit, ve kterých bodech může měnit funkce znaménko. Obecně to jsou průsečíky s osou x a potom také body, ve kterých není zadaná funkce definována.[94]

[94] Teoreticky by mohly nastat ještě jiné případy, ale ne pro funkce, se kterými se setkáme v tomto textu.

Průsečíky s osou x nemáme žádné, jediný bod, kde může f měnit znaménko, je tedy $x = 0$. Ten nám rozdělí reálnou osu na dvě části (= dva intervaly). Poté už stačí vzít z každého intervalu pouze jeden bod a dosadit jej do f. Výsledné znaménko pak bude shodné na celém intervalu. Vždy je dobré zvolit bod, který se nám bude do funkčního předpisu snadno dosazovat. V tomto případě se nabízí dvojice $x = -1$ a $x = 1$.[95] Načrtněme si obrázek.

[95] Všimněme si, že pokud nám jde pouze o to určit znaménko funkce, tak vůbec nemusíme tyto hodnoty dosazovat do čitatele. Jak už jsme uvedli výše, platí totiž, že $e^x > 0$, takže čitatel bude kladný vždy. O znaménku funkce rozhoduje pouze jmenovatel.



Monotonnost funkce

V druhé části určíme, kde je funkce f klesající a rostoucí, případně má-li nějaké lokální extrémy. Z předchozí kapitoly víme, že tyto informace můžeme zjistit skrze první derivaci funkce f.

$$f'(x) = \left(\frac{e^x}{x}\right)' = \frac{e^x x - e^x \cdot 1}{x^2} = \frac{e^x(x-1)}{x}$$

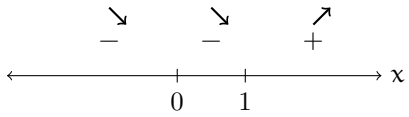
Zopakujme, že pokud $f'(x) > 0$, pak f je v bodě x rostoucí, obrácená nerovnost indikuje v daném bodě klesání. Opět nám tedy půjde o to určit znaménko funkce, tentokrát ovšem f'. [96]

[96] Body, ve kterých se znaménko f' může měnit, jsou opět průsečíky f' s osou x a body nespojitosti f'.

Bod nespojitosti f' je opět $x = 0$. Průsečíky f' s osou x jsou pak body, pro které platí, že $f'(x) = 0$. Tedy:

$$0 = \frac{e^x(x-1)}{x^2}$$

Tato rovnice má zjevně jediné řešení $x = 1$ (stacionární bod). Reálná osa se nám tedy rozpadá na tři intervaly. Z každého opět vybereme jeden bod a určíme znaménko f':[97]



[97] V tomto případě nemá smysl pro urychlení výpočtu dosazovat do jmenovatele, protože ten je vždy kladný.

Jak vidíme, v bodě $x = 1$ má funkce f lokální minimum. Určeme ještě jeho y-souřadnici dosazením do předpisu funkce f:

$$f(1) = \frac{e^1}{1} = e$$

Dostáváme tedy bod $\mathrm{LMIN}[1; e]$.

Konvexnost a konkávnost funkce

Ve třetí části určíme pomocí druhé derivace intervaly, na kterých je funkce f konvexní, resp. konkávní. Nejprve vlastní derivace:

$$\begin{aligned} f''(x) &= \left(\frac{e^x(x-1)}{x^2}\right)' = \frac{[e^x(x-1)+e^x]\,x^2 - e^x(x-1)2x}{x^4} \\ &= \frac{xe^xx^2 - 2xe^x(x-1)}{x^4} = \frac{x^2e^x - 2xe^x + 2e^x}{x^3} = \\ &= \frac{e^x(x^2-2x+2)}{x^3} \end{aligned}$$

Funkce je konvexní v daném bodě x, pokud platí $f''(x) > 0$, konkávní pokud platí $f''(x) < 0$. Znaménko f'' určíme opět zcela stejně jako znaménko f a f'.

Bodem nespojitosti f'' je zjevně $x = 0$. Průsečíky f'' s osou x jsou pak řešením rovnice:

$$0 = \frac{e^x(x^2-2x+2)}{x^3}$$

Tuto rovnici si můžeme zjednodušit:[98]

[98] Opět díky tomu, že víme, že $e^x > 0$, tedy zejména platí $e^x \neq 0$.

$$0 = x^2 - 2x + 2$$

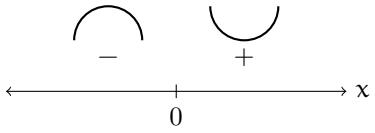
Tuto kvadratickou rovnici můžeme řešit standardním způsobem vzorci pro výpočet kořenů. Zjistíme, že diskriminant je záporný, takže řešení neexistuje.[99]

[99] Elegantněji si můžeme všimnout, že platí následující:

$$0 = x^2 - 2x + 2$$
$$0 = x^2 - 2x + 1 + 1$$
$$0 = (x-1)^2 + 1$$

Po poslední úpravě máme na pravé straně výraz, který je určitě větší než 0. Rovnice tedy nemá řešení.

Reálnou osu tedy rozdělíme pouze na dva intervaly okolo bodu 0. Dosadíme libovolný bod z každého intervalu a určíme znaménko f''. Zakreslíme vše do obrázku:



Asymptoty funkce

Poslední výpočetní částí je určení asymptot. Asymptoty bez směrnice nám vznikají v bodech nespojitosti funkce f, tedy v bodě $x = 0$. Jak se bude graf funkce k této asymptotě blížit, určíme výpočtem jednostranných limit:

$$\lim_{x\to 0^-} \frac{e^x}{x} = -\infty$$
$$\lim_{x\to 0^+} \frac{e^x}{x} = \infty$$

Asymptoty se směrnicí jsou přímky tvaru $y = kx + q$, zde zjišťujeme chování funkce pro $x \to \pm\infty$. Nejprve určeme pomocí vzorců koeficienty asymptoty pro $x \to \infty$:[100]

[100] Pro $x \to \infty$ platí, že $e^x \gg x^2$.

$$k_1 = \lim_{x\to\infty} \frac{f(x)}{x} = \lim_{x\to\infty} \frac{e^x}{x^2} = \infty$$

Protože výsledkem není reálné číslo, tato asymptota neexistuje. Pro druhý případ dostáváme:[101]

[101] Uvědomme si (například z grafu funkce na začátku této kapitoly), že platí $e^x \to 0$ pro $x \to -\infty$.

$$k_2 = \lim_{x\to-\infty} \frac{f(x)}{x} = \lim_{x\to\infty} \frac{e^x}{x^2} = 0$$
$$q_2 = \lim_{x\to-\infty} f(x) - k_2 x = \lim_{x\to-\infty} \frac{e^x}{x} = 0$$

Asymptotou pro $x \to -\infty$ je tedy přímka $y = 0$ (= osa x).

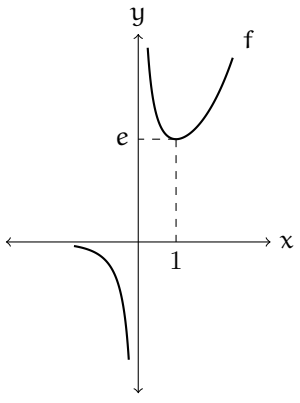
Graf funkce

Poslední částí je už jen nakreslit graf. Při tom využijeme všechny dříve zjištěné informace. Po souřadných osách má smysl jako první zakreslit asymptoty a významné body, kterými graf funkce prochází (průsečíky s osami, lokální extrémy). Dále už pouze kombinujeme získané informace pro nakreslení výsledného obrázku:[102]

[102] Z opačného pohledu můžeme také tuto fázi použít jako kontrolu, že jsme počítali správně. Všechny zjištěné vlastnosti si musí odpovídat.

Obor hodnot už nyní určíme snadno z obrázku:

$$H_f = (-\infty; 0) \cup [e; \infty)$$



Funkce více proměnných

Doposud jsme v rámci matematické analýzy pracovali s funkcemi jedné proměnné. Pro každou hodnotu nezávislé proměnné x z definičního oboru této funkce, jsme dle předpisu $f(x)$ získali hodnotu závislé proměnné y. Takovou funkcí je třeba $f : y = 2x^2 - 3$.

U funkcí můžeme explicitně vyjádřit jejich typ. V případě funkce jedné proměnné f vypadá takto:

$$f: \quad \mathbb{R} \to \mathbb{R}$$

Tento zápis nám říká, že funkce f má jako argument jedno reálné číslo a jejím výsledek je také reálné číslo.

Mějme nyní jinou funkci, která bude mít dva argumenty, např. $g(x, y)$ a napišme i její typ:[103]

$$g: \quad z = 2x^2 + y$$
$$g: \quad \mathbb{R}^2 \to \mathbb{R}$$

[103] Zde jsme použili explicitnější označení s oběma argumenty abychom zdůraznili, že má funkce dvě nezávislé proměnné. Pokud je to z kontextu zřejmé, stačí psát g.

Samozřejmě můžeme pracovat s funkcemi, které mají i více argumentů. Uvažme obecně funkci $h(x_1, \ldots, x_n)$ s n argumenty[104], její typ pak bude:

[104] Pokud máme funkci více než dvou proměnných, označujeme obecně její argumenty jako x_i.

$$h: \quad \mathbb{R}^n \to \mathbb{R}$$

Na funkci h se můžeme dívat tak, že má n reálných argumentů, nebo analogicky, že jejím argumentem je n-složkový vektor reálných čísel.

Ačkoliv se může zdát, že funkce více proměnných už jsou příliš abstraktní koncept, setkáváme se s nimi poměrně běžně. Třeba vzorec pro objem válce V je vlastně funkce dvou proměnných – poloměru podstavy a jeho výšky:

$$V(r, v) = \pi r^2 v$$

Jiným příkladem může být třeba Coulombův zákon, tedy výpočet síly působící mezi dvěma elektricky nabitými částicemi. Zde vyjádřený jako funkce tří proměnných F:[105,106]

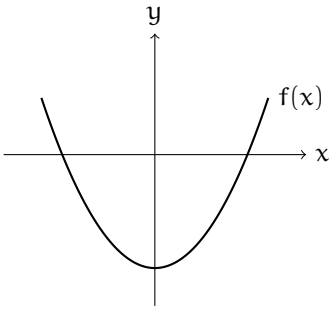
$$F(q_1, q_2, r) = k_e \frac{q_1 q_2}{r^2}$$

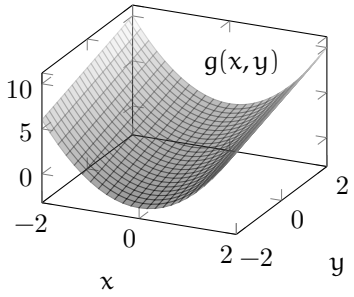
[105] k_e je Coulombova konstanta.

[106] Všimněme si, že pokud velikosti nábojů „zafixujeme“, dostáváme funkci pouze jedné proměnné $F(r)$.

VYKRESLIT GRAF funkce více proměnných je komplikovanější. Protože pro jeho zobrazení potřebujeme vždy o jeden rozměr více, než je počet argumentů funkce, zůstaneme v tomto textu u dvou proměnných.[107]

Níže srovnejme grafy funkcí f a g ze začátku kapitoly:





I u funkcí více proměnných můžeme určovat vlastnosti podobně, jako jsme to dělali dříve, nicméně většina výpočtů by byla výrazně složitějších.[108] Zaměříme se tedy dále jen na jednu oblast – hledání extrémů.

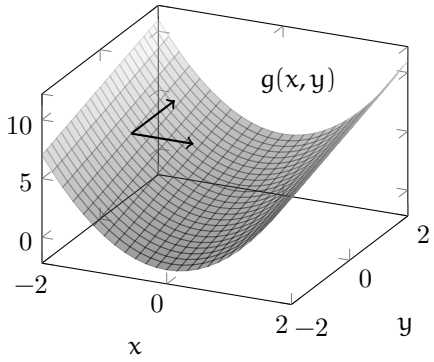
[107] To je asi i maximum toho, co si je schopen člověk představit. U vyšších dimenzí už si musíme pomoci různými projekcemi.

[108] Například místo průsečíků s osou x máme u funkce dvou proměnných průnik s rovinou xy.

Parciální derivace

Zopakujme si, že aby nějaký bod mohl být extrém u funkce jedné proměnné, musí být hodnota první derivace v tomto bodě nula. První derivace určuje změnu funkční hodnoty „při pohybu po ose x“. Pro funkci dvou proměnných budou existovat derivace dvě – ve směru osy x a ve směru osy y. Takovým derivacím říkáme *parciální*.

Ukažme si nyní, jak zmíněné směry vypadají v nějakém konkrétním bodě, např. $[-3/2; 0]$ na příkladu funkce g.



Jak můžeme vidět, ve směru osy x funkční hodnota $z = g(x, y)$ klesá, zatímco ve směru osy y roste. Hodnota první parciální derivace podle x je tedy záporná, podle y naopak kladná.

Než určíme přesně výpočtem velikost zmíněných parciálních derivací, musíme si povědět něco o tom, jak se parciální derivace zapisují. Uvedeme dva běžně používané způsoby zápisu. Oproti funkcím jedné proměnné zde musíme explicitně vyznačit podle které proměnné derivujeme. Oba tvary pro obě parciální derivace vypadají takto:[109,110]

[109] Čteme jako „první parciální derivace z (nebo $g(x, y)$) podle x (nebo y)“.

[110] Symbol pro parciální derivaci ∂ není řecké písmeno δ.

$$z'_x = \frac{\partial g(x,y)}{\partial x}$$
$$z'_y = \frac{\partial g(x,y)}{\partial y}$$

Výpočet parciálních derivací je poměrně jednoduchý, protože se od derivace funkcí jedné proměnné liší pouze v jednom pravidle. Při derivaci podle zadané proměnné na všechny ostatní proměnné pohlížíme jako na konstanty. Uveďme pro ilustraci pár příkladů:

$$\frac{\partial}{\partial x}(x^2y) = 2xy \qquad \frac{\partial}{\partial y}(x^2y) = x^2$$
$$\frac{\partial}{\partial x}(3x+y) = 3 \qquad \frac{\partial}{\partial y}(3x+y) = 1$$
$$\frac{\partial}{\partial x}(e^x) = e^x \qquad \frac{\partial}{\partial y}(e^x) = 0$$

V tuto chvíli už můžeme spočítat parciální derivace funkce g z příkladu výše:

$$\frac{\partial g(x,y)}{\partial x} = \frac{\partial}{\partial x}(2x^2+y) = 4x$$
$$\frac{\partial g(x,y)}{\partial y} = \frac{\partial}{\partial y}(2x^2+y) = 1$$

Konkrétní hodnotu derivací v bodě $[-3/2;0]$ pak získáme prostým dosazením:[111]

$$\left.\frac{\partial g(x,y)}{\partial x}\right|_{x=-3/2;y=0} = -6$$
$$\left.\frac{\partial g(x,y)}{\partial y}\right|_{x=-3/2;y=0} = 1$$

[111] Zkontrolujte s grafem funkce g. Ve směru osy x funkce v bodě $[-3/2;0]$ poměrně rychle klesá, naopak ve směru osy y mírně roste.

Druhé parciální derivace se opět počítají analogicky. Druhá derivace je první derivace první derivace. Máme-li dvě proměnné, dostáváme tak celkem čtyři různé druhé parciální derivace. Na příkladu funkce g si ukažme způsob značení:[112]

$$\frac{\partial^2 g(x,y)}{\partial x^2} \qquad \frac{\partial^2 g(x,y)}{\partial y^2}$$
$$\frac{\partial^2 g(x,y)}{\partial x \partial y} \qquad \frac{\partial^2 g(x,y)}{\partial y \partial x}$$

[112] Čteme například „druhá parciální derivace g dvakrát podle x“, nebo „druhá parciální derivace g podle x a y“.

Pro spojité funkce navíc platí, že nezáleží na pořadí derivování, můžeme tedy psát:[113]

$$\frac{\partial^2 g(x,y)}{\partial x \partial y} = \frac{\partial^2 g(x,y)}{\partial y \partial x}$$

[113] A s jinými se zde v tomto textu nesetkáme.

Extrémy funkcí dvou proměnných

Aby měla funkce dvou proměnných lokální extrém (minimum nebo maximum) v nějakém bodě, musí platit, že jsou obě parciální derivace v tomto bodě nulové. Například funkce g určitě žádný extrém nemá, protože $\frac{\partial g}{\partial y}$ nemůže být nikdy nula.[114]

[114] Opět ověřte s obrázkem výše.

Zkusme zvolit jinou funkci:

$$h: \quad z = x^3 + y^3 - 3xy + 6$$

Spočítáme obě parciální derivace:

$$\frac{\partial h}{\partial x} = 3x^2 - 3y$$
$$\frac{\partial h}{\partial y} = 3y^2 - 3x$$

Ty položíme rovny nule a vyřešíme jako soustavu dvou rovnic o dvou neznámých.

$$3x^2 - 3y = 0 \quad \rightarrow \quad x^2 - y = 0 \quad \rightarrow \quad x^2 = y$$
$$3y^2 - 3x = 0 \quad \rightarrow \quad y^2 - x = 0$$

Vyjádřením proměnné y z první rovnice a dosazením do druhé získáme po úpravách výsledek:

$$(x^2)^2 - x = 0$$
$$x^4 - x = 0$$
$$x(x^3 - 1) = 0$$
$$x_1 = 0 \qquad x_2 = 1$$
$$y_1 = 0 \qquad y_2 = 1$$

Tímto výpočtem jsme určili souřadnice dvou stacionárních bodů. Označme je například A a B:

$$A[0;0] \qquad B[1;1]$$

K tomu, abychom zjistili, zdali je některý ze stacionárních bodů lokálním minimem nebo maximem, využijeme test pomocí druhých parciálních derivací.

Definujme nejdříve *Hessián*, determinant z matice druhých parciálních derivací funkce f:

$$H = \det \begin{bmatrix} \frac{\partial^2 f(x,y)}{\partial x^2} & \frac{\partial^2 f(x,y)}{\partial x \partial y} \\ \frac{\partial^2 f(x,y)}{\partial y \partial x} & \frac{\partial^2 f(x,y)}{\partial y^2} \end{bmatrix}$$

Dosadíme-li pak za x a y souřadnice nějakého bodu S, pak platí:

$$H(S)\begin{cases} > 0 & \text{S je lokální extrém} \\ = 0 & \text{nelze rozhodnout} \\ < 0 & \text{S není lokální extrém} \end{cases}$$

Zajímavé jsou poslední dva případy. Pokud je hodnota Hessiánu v nějakém bodě záporná, znamená to, že se nejedná o extrém, ale o tzv. *sedlový bod*. Jde o bod, který se chová v jednom směru jako maximum a v jiném jako minimum.[115]

Pokud je naopak hodnota Hessiánu v zadaném bodě nula, tento postup nám s rozhodnutím, jestli jde o lokální extrém, nepomůže. Pro účely tohoto textu se s tímto závěrem spokojíme.[116]

Spočítejme nejdříve Hessián pro funkci h:

$$H = \det \begin{bmatrix} \frac{\partial^2 h(x,y)}{\partial x^2} & \frac{\partial^2 h(x,y)}{\partial x \partial y} \\ \frac{\partial^2 h(x,y)}{\partial y \partial x} & \frac{\partial^2 h(x,y)}{\partial y^2} \end{bmatrix} = \det \begin{bmatrix} 6x & -3 \\ -3 & 6y \end{bmatrix} = 36xy - 9$$

Dosadíme nyní dva stacionární body:

$$H(A) = 36 \cdot 0 \cdot 0 - 9 = -9 < 0$$
$$H(B) = 36 \cdot 1 \cdot 1 - 9 = 27 > 0$$

Bod A extrém tedy není, jedná se o sedlo. Bod B je lokální extrém. Zbývá už pouze určit, zdali se jedná o minimum nebo maximum, což zjistíme dosazením do druhé parciální derivace dvakrát podle x. Pro nějaký bod S obecně platí:

$$\left.\frac{\partial^2 f(x,y)}{\partial x^2}\right|_{S[x_0;y_0]} = \begin{cases} > 0 & \text{S je lokální minimum} \\ < 0 & \text{S je lokální maximum} \end{cases}$$

V našem případě získáváme:

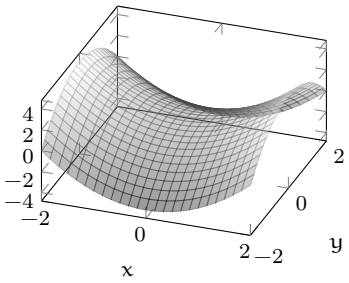
$$\left.\frac{\partial^2 h(x,y)}{\partial x^2}\right|_{B[1;1]} = 6x|_{B[1;1]} = 6 > 0$$

Bod B je tedy lokální minimum. Navíc můžeme určit ještě jeho z-ovou souřadnici dosazením do původního předpisu funkce h:

$$h(1,1): \quad z = 1^3 + 1^3 - 3 \cdot 1 \cdot 1 + 6 = 5$$

Závěrem si ukažme ještě graf funkce h pohledem ve směru osy z:[117]

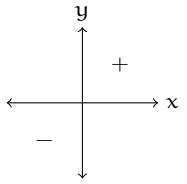
[115] Pro lepší představu si ukažme graf jednoduché funkce $z = x^2 - y^2$, která má sedlový bod $[0;0]$:



[116] Jako rozšíření rozeberme na ukázku alespoň jeden příklad, funkci $z = x^3 + y^3$. Funkce má jediný stacionární bod $[0;0]$, ve kterém je Hessián ovšem nulový.

Zkusme si nyní představit, jakých funkčních hodnot nabývá tato funkce v okolí bodu $[0;0]$. Rozdělme rovinu xy na čtyři kvadranty podle toho, jestli je hodnota x a y kladná, resp. záporná.

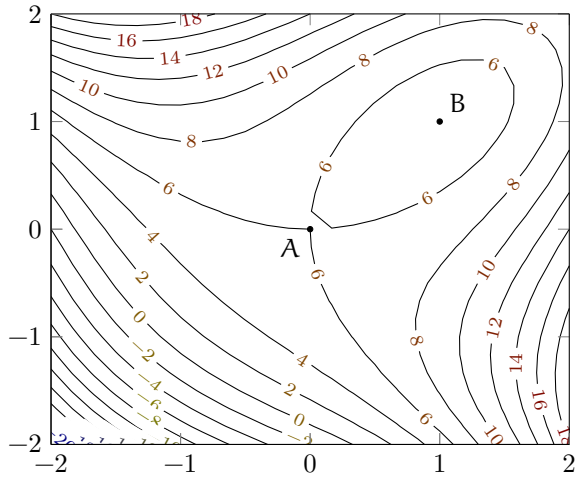
Pokud vezmeme libovolný bod, pro který platí $x > 0$ a $y > 0$, nutně vyplývá $z > 0$. Analogicky pro $x < 0$ a $y < 0$ platí $z < 0$. Graficky si to můžeme znázornit následovně:



Odtud vidíme, že bod $[0;0]$ nemůže být extrém. Pokud by měl být minimum, musela by být jeho funkční hodnota (zde 0) vždy nižší než hodnota bodů v okolí (analogicky vyšší pro maximum). Zde jsme ovšem ukázali, že v jednom z kvadrantů jsou funkční hodnoty nižší a ve druhém vyšší, takže se o extrém nejedná.

Nakonec zkuste použít stejný postup pro funkci $z = x^2 + y^2$, která má v bodě $[0;0]$ lokální minimum.

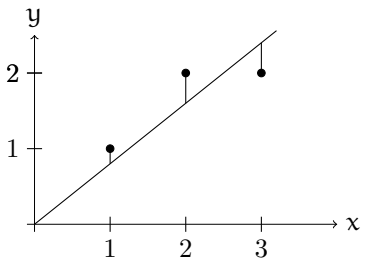
[117] Jedná se vlastně o analogii map se zobrazenými vrstevnicemi



Metoda nejmenších čtverců

Vraťme se nyní k příkladu, který jsme řešili v rámci lineární algebry. Cílem bylo nalézt přímku, která nejlépe aproximuje zadané body v rovině. Nyní si ukážeme, jak lze stejný příklad vyřešit s pomocí matematické analýzy, konkrétně parciálních derivací.

Zkusme si nyní vzít stejné zadání a do obrázku zakreslit libovolnou přímku ($y = kx + q$) a navíc chyby, kterých jsme se takto zvolenou aproximací dopustili:



Zjevně platí, že čím menší je součet délek těchto „chybových“ úseček, tím je zvolená aproximace přesnější. Zkusme pro ukázku vyjádřit velikost první chyby. To je vzdálenost bodu měření $[1; 1]$ od zvolené přímky, kde $x = 1$. Rozdíl (označme zde e_1) je tedy:

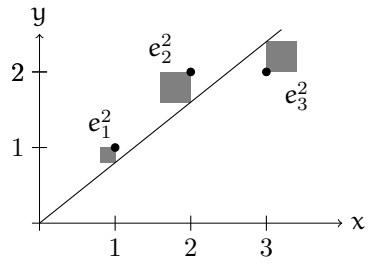
$$e_1 = 1 - (kx + q) = 1 - (k \cdot 1 + q) = 1 - k - q$$

Analogicky bychom mohli vyjádřit i ostatní. Pro třetí bod by nám ale tímto postupem vyšla záporná velikost, což samozřejmě nechceme. Mohli bychom jednoduše vzít každý z rozdílů v absolutní hodnotě, ale to má jistá úskalí[118], místo toho použijeme druhou mocninu rozdílu, která je také vždy nezáporná:

$$e_1^2 = (1 - (k \cdot 1 + q))^2$$
$$e_2^2 = (2 - (k \cdot 2 + q))^2$$
$$e_3^2 = (2 - (k \cdot 3 + q))^2$$

[118] Jedná se vlastně o technické omezení, absolutní hodnota totiž nemá definovanou derivaci, kterou budeme dále potřebovat, pro všechny hodnoty x.

Druhou mocninu nějakého čísla můžeme graficky vyjádřit jako čtverec, upravený obrázek pak bude vypadat takto:



Celková velikost chyby je tedy určena součtem $e_1^2 + e_2^2 + e_3^2$. Naším cílem je tuto chybu minimalizovat. Odtud plyne název *metoda nejmenších čtverců*.

Velikost této chyby závisí na zvolené přímce, konkrétně na jejích parametrech k a q. Zapišme nyní velikost chyby jako funkci dvou proměnných $E(k, q)$:

$$E(k, q) = e_1^2 + e_2^2 + e_3^2$$

Konkrétně v našem případě:

$$\begin{aligned} E(k, q) &= e_1^2 + e_2^2 + e_3^2 = \\ &= (1 - (k \cdot 1 + q))^2 + (2 - (k \cdot 2 + q))^2 + (2 - (k \cdot 3 + q))^2 = \\ &= (1 - k - q)^2 + (2 - 2k - q)^2 + (2 - 3k - q)^2 = \end{aligned}$$

Naším úkolem je minimalizovat funkci $E(k, q)$, tzn. nalézt bod (= dvojici hodnot) $[k; q]$, pro který výše uvedený výraz nabývá nejnižší hodnoty. V tomto bodě musí platit, že jsou obě parciální derivace nulové:

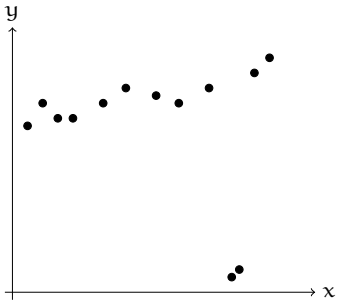
$$\begin{aligned} \frac{\partial E(k, q)}{\partial k} &= 0 \\ \frac{\partial E(k, q)}{\partial q} &= 0 \end{aligned}$$

Jednotlivé parciální derivace vypadají takto:

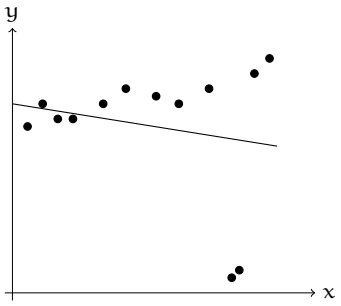
$$\begin{aligned} \frac{\partial E(k, q)}{\partial k} &= 2(1 - k - q) \cdot (-1) + 2(2 - 2k - q) \cdot (-2) + 2(2 - 3k - q) \cdot (-3) = \\ &= (-2 + 2k + 2q) + (-8 + 8k + 4q) + (-12 + 18k + 6q) = \\ &= 28k + 12q - 22 \\ \frac{\partial E(k, q)}{\partial q} &= 2(1 - k - q) \cdot (-1) + 2(2 - 2k - q) \cdot (-1) + 2(2 - 3k - q) \cdot (-1) = \\ &= (-2 + 2k + 2q) + (-4 + 4k + 2q) + (-4 + 6k + 2q) = \\ &= 12k + 6q - 10 \end{aligned}$$

Poznamenejme, že způsob, jakým jsme problém zadefinovali ovlivní podobu výsledku. Protože používáme druhé mocniny chyb, bude výpočet *nejlepší* aproximace výrazně ovlivněn odlehlými hodnotami (angl. *outliers*).

Zkuste se zamyslet nad následujícími daty:



S výjimkou dvou bodů můžeme vidět v datech mírný růst. I tyto dva body ovšem stačí k tomu, aby vypočtený trend metodou nejmenších čtverců byl přesně opačný.



To není chyba metody, ale její vlastnost. Jestli je vhodné ji použít v této podobě či nikoliv, pak záleží na konkrétní aplikaci, datech. Může se jednat třeba o chyby měření, pak dává smysl tyto hodnoty před výpočtem vypustit. Na druhou stranu mohou být skutečně *správně*, pak bychom se ale měli zamyslet, co v kontextu dané aplikace znamenají.

Položíme je nyní obě rovny 0 a upravíme:

$$\begin{aligned} 28k + 12q - 22 &= 0 \\ 12k + 6q - 10 &= 0 \\ &\downarrow \\ 14k + 6q &= 11 \\ 6k + 3q &= 5 \end{aligned}$$

Všimněme si, že jsme dostali úplně stejný systém rovnic jako dříve, i když jsme použili zcela jiný postup.

NYNÍ SI PŘEDSTAVÍME OBECNOU PODOBU celé metody. Uvažme množinu n bodů, které označíme jako $[x_1; y_1]$, $\ldots$, $[x_n; y_n]$. Těmito body budeme chtít proložit nějakou funkci s m parametry $p_1, \ldots, p_m$, kterou zde označujeme jako *model* $M(x, p_1, \ldots, p_m)$. Pak velikost chyby v závislosti na hodnotě parametrů p_i můžeme vyjádřit jako:[119]

$$E(p_1, \ldots, p_m) = \sum_{i=1}^{n} (y_i - M(x_i, p_1, \ldots, p_m))^2$$

[119] Tento výraz vypadá možná na první pohled složitě, ale vyjadřuje pouze součet druhých mocnin (= čtverců) odchylek mezi skutečnou a modelem predikovanou hodnotou.

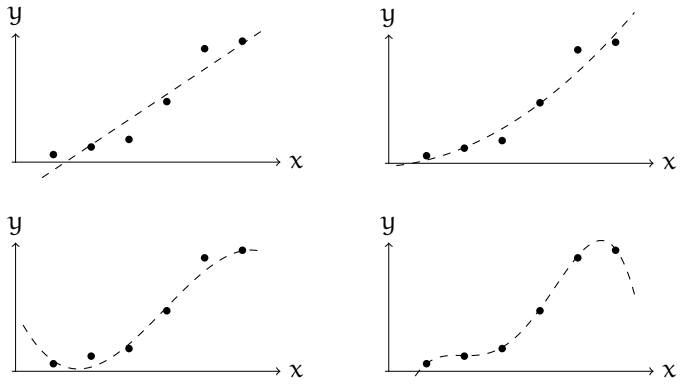
Pokud by modelem byla například kvadratická funkce, platilo by:[120]

$$\begin{aligned} M(x, a, b, c) &= ax^2 + bx + c \\ E(a, b, c) &= \sum_{i=1}^{n} \left(y_i - (ax_i^2 + bx_i + c)\right)^2 \end{aligned}$$

[120] Srovnejte s lineární funkcí, kterou jsme použili v úvodním příkladu.

V tomto případě bychom v dalším kroku potřebovali tři parciální derivace (podle parametrů a, b, c) a pro ně pak vyřešit soustavu tří rovnic o třech neznámých.

VOLBA VHODNÉHO MODELU je klíčový parametr celého procesu. Model by měl odpovídat charakteru dat[121], jak si ukážeme na následujícím příkladu. Pro stejná data jsme použili čtyři různé modely – polynomy prvního až čtvrtého stupně.



[121] Pokud bychom třeba sledovali dráhu padajícího míčku v závislosti na čase, vhodný model by byl kvadratický. Pro jeho rychlost by se pak hodil lineární (vzpomeňme si na vzorce $s = 1/2gt^2$, resp. $v = gt$).

Ačkoliv v posledním případě je chyba téměř nulová, predikční schopnost modelu klesá, protože neodráží trend ve vstupních datech. Takovému stavu se říká *overfitting* a většinou je to signál toho, že je zvolený model příliš složitý.[122] Lineární nebo kvadratický polynom se zdá být lepší volbou.

[122] Podrobnější diskuze už je však nad zamýšlený rámec tohoto textu.

Pár slov k aplikacím

V této poslední kapitole zmíníme pár příkladů, kde se můžeme s dříve probíranými koncepty setkat. Vezměme to postupně dle prezentovaných kapitol formou krátkého shrnutí.

Reprezentace reálných čísel

Protože se dnes zpracovávají data už prakticky výhradně na počítači, je dobré mít alespoň základní znalost toho, jak jsou v něm tato data reprezentována, a jaké případné problémy může tato reprezentace přinést.

Využitím vhodných knihoven nebo přímo nějakého softwaru pro analýzu dat můžeme vliv těchto problémů potlačit na minimum. Jistě není dobrý nápad implementovat zaběhlé metody „svépomocí“, při návrhu nových metod se pak snažíme v největší míře využít těch standardních.[123]

[123] To platí zejména pro zamýšlené čtenáře tohoto textu, tedy „běžné uživatele“ matematiky. Máte-li například doktorát z matematiky, toto doporučení se na vás nevztahuje.

Systémy lineárních rovnic

Jak jsme v předchozích kapitolách viděli, řešení systému lineárních rovnic se objevuje napříč všemi probíranými tématy. Jedná se o takový pomyslný základní stavební kámen celé lineární algebry.

Uveďme jednoduchý příklad. Mějme nějakou surovinu, která obsahuje konkrétní množství třech účinných látek. Otázka zní, jestli tuto surovinu dokážeme nahradit jinými surovinami s obecně jiným obsahem těchto látek tak, aby jejich celkové množství zůstalo v součtu zachováno.

Jinými slovy hledáme *lineární kombinaci* nových surovin, která by byla ekvivalentní té původní. Její koeficienty navíc musí být nezáporná čísla (záporné množství zde nedává smysl).

Mnohé z takových příkladů zvládneme vyřešit intuitivně postupy ze základní nebo střední školy, aniž bychom potřebovali formálně aparát lineární algebry. Máme-li soustavu dvou rovnic o dvou neznámých, kterou jsme často používali v příkladech, mohou se zdát zmíněné metody zbytečně komplikované. Pokud bychom ovšem měli desítky nebo stovky takových rovnic, začnou tyto postupy nabývat na významu. V těchto případech už je nutné použít počítač a lineární algebra nám poskytuje množství nástrojů, které můžeme pro řešení použít.

Metoda nejmenších čtverců

Zde jsme motivaci zmiňovali už v příslušné kapitole. Jde o základní způsob, jak „nafitovat“[124] nějakou křivku na zadaná data. Konkrétní tvar dané křivky je určen hodnotami jejích parametrů (koeficientů).

V původním příkladu jsme uvažovali přímku, ale situace je analogická třeba i pro parabolu[125].

$$y = ax^2 + bx + c$$

Cílem je opět najít takové hodnoty a, b, c, pro které výsledná parabola nejlépe aproximuje data.

Jestli ale použít přímku, parabolu, nebo jinou funkci je už závislé na charakteru dat. Většinou víme, co data představují, máme tedy nějakou představu o hledané závislosti.

Transformace

S využitím matic pro reprezentaci konkrétních transformací jsme se potkali také v několika případech. První z nich představují „abstraktnější“ koncepty, které jsme využili při odvození různých postupů (výpočet inverze, změna báze).

Druhý pak odkazuje na geometrické transformace, které nacházejí velké využití v počítačové grafice. Dobrým příkladem může být třeba změna perspektivy při úpravě fotek.[126] Takovou transformaci lze vyjádřit násobením vhodnou maticí.

Užití mohou nalézt třeba při studiu symetrie molekul, kdy jednotlivé operace symetrie lze opět popsat maticemi.

Redukce dimenzí

Redukce dimenzí je užitečná tam, kde pracujeme s mnohodimenzionálními daty. Cílem je snížit dimenzi dat při zachování co možná největšího množství informace.[127]

Jedním ze základních nástrojů je PCA, které původní charakteristiky dat[128] transformuje na vhodně zvolené nové (změna báze). Tyto nové vlastnosti jsou určovány v pořadí od nejdůležitějších (= dokáží nejlépe popsat variabilitu v původních datech) až po nejméně důležité. Při následné analýze se pak můžeme zaměřit pouze na několik prvních komponent.

Jedním z využití může být i odstranění šumu v původních datech, ten může být totiž identifikován jako některá z minoritních složek.

Uveďme jeden příklad, kdy se PCA využívá při studiu vnitřních pohybů proteinů. Zatímco pohyby jednotlivých atomů nemusí být samy o sobě významné[129], dohromady mohou realizovat změnu celé struktury proteinu, což pak může mít vliv na jeho biologickou funkci. Cílem PCA pak může být tyto pohyby popsat a oddělit je od těch, které nejsou pro jeho funkci relevantní.

[124] Tento výraz nemá asi v češtině lepší ekvivalent.

[125] Obecně můžeme zvolit libovolnou funkci, která je lineární vzhledem k hledaným koeficientům.

[126] Zkuste se zamyslet, jak reprezentovat obrázek jako nějakou matici.

[127] Uvědomme si, že některé původní vlastnosti mohou být např. korelované, a tedy v jisté směru redundantní.

[128] U člověka by takové charakteristiky mohly být třeba výška, váha, věk, atd.

[129] Navíc jsou vysoce korelované, protože jsou atomy spojeny vazbami.