1 Vektory

Vektor = jednorozmerné pole

Definovanie je možné viacerými spôsobmi:

- explicitným vymenovaním zoznamu prvkov
- vygenerovaním pomocou zabudovaných matlabovských funkcií
- načítaním externého súboru dát

Základné konvencie:

- čiarka alebo medzera oddeľuje prvky v riadku
- bodkočiarka oddeľuje riadky, t.j. definujú sa stĺpce
- celý zoznam prvkov je ohraničený hranatými zátvorkami

1.1 Definovanie vektorov

1.1.1 Definovanie vymenovaním prvkov

`>> v = [3 1]`	Vytvorí riadkový vektor (a vypíše ho, lebo tam nie je bodkočiarka)
`>> v = [3, 1, 7, -21, 5, 6]`	Vytvorí príslušný riadkový vektor
`>> v = [3; 1; 7; -21; 5; 6]`	Vytvorí príslušný stĺpcový vektor
`>> v = [3 1 7 -21 5 6]'`	Transpozícia riadkového vektora na stĺpcový
`>> v`	Slúži na prezretie obsahu vektora `v`

Príklad 1.1 *Takýmto spôsobom je možné definovať vektory ľubovoľnej dĺžky (avšak iba po maximálnu dĺžku povolenú softvérom). Zistite, akú maximálnu dĺžku môže mať vektor v tej verzii Matlab-u, ktorú používate. Vo väčšine používaných verzií je toto číslo rovnaké. Čo ste sa dozvedeli? Návod: Použite príkaz* `>> [System,MaxArrayElements]=computer` *Pri narábaní s priveľkými poľami (vektormi alebo maticami) však môžeme naraziť na problémy s pamäťou.*

Príklad 1.2 *Zadefinujte (ľubovoľný) riadkový vektor* `c` *dĺžky 5. Zadefinujte (ľubovoľný) stĺpcový vektor* `d` *dĺžky 6.*

1.1.2 Definovanie pomocou zabudovaných funkcií

Matlab umožňuje generovať niektoré špeciálne typy vektorov pohodlným spôsobom pomocou zabudovaných funkcií.

`zeros(n,1)`	Stĺpcový vektor núl (resp. matica s rozmermi $n \times 1$)
`zeros(1,n)`	Riadkový vektor núl (resp. matica s rozmermi $1 \times n$)
`ones(n,1)`	Stĺpcový vektor jednotiek (resp. matica s rozmermi $n \times 1$)
`ones(1,n)`	Riadkový vektor jednotiek (resp. matica s rozmermi $1 \times n$)

1.1.3 Iné spôsoby definovania

Aritmetická postupnosť čísel (s jednotkovou, kladnou, zápornou, celočíselnou, alebo neceločíslenou diferenciou):

```
>> v = [1:1:8]            (to sa dá úspornejšie zapísať aj ako v = [1:8] )
v =
       1     2     3     4     5     6     7     8
>> v = [2:0.5:4]
v =
    2.0000    2.5000    3.0000    3.5000    3.7500    4.0000
```

1.2 Pristupovanie k prvkom vektora

Na prezretie prvku vektora zadáme názov vektora a zaň do okrúhlych zátvoriek poradové číslo želaného prvku.

Príklad 1.3 *Vytvorte si vektor* `v` *dĺžky aspoň 4. Potom vyskúšajte a zistite, čo robia nasledujúce príkazy:*

```
>> v(1)
>> v(1:3)
>> b = v([2,3,end])
```

Všimnite si tiež, kedy sa vytvorila nová premenná s názvom ans. Táto sa vytvorí vždy, keď vyvolávate akciu, ktorá vo výstupe nemá priradené meno. V našom prípade vektor síce meno má, ale jeho zložky mená nemajú. Premenná `ans` *je obyčajnou premennou ako ktorákoľvek iná, t.j. pamätá si hodnotu (pokiaľ ju samozrejme nepremažeme nejakou inou) a možno ju napríklad transponovať. Vyskúšajte zadať nasledovné príkazy:*

```
>> ans
>> ans'
```

Príklad 1.4 *Vykonajte nasledovné príkazy. Po každom kroku si uvedomte, čo sa udialo.*

```
>> v = [0:2:8]
>> v
>> v;
>> v'
```

Príklad 1.5 *Čo vykoná nasledovný príkaz? Ktoré členy vektora* `v` *sa zobrazia?*

```
>> v(1:2:4)
```

Skontrolujte si svoju odpoveď tak, že si necháte vypísať aj celý vektor. Odpovedali ste správne? Ako si pozriete obsah prvého, tretieho a piateho člena vektora `v`*? A ako si pozriete obsah druhého, štvrtého a šiesteho člena vektora* `v`*?*

1.3 Základné operácie na vektoroch

S vektormi môžeme robiť štandardné operácie, ako sú sčítanie, odčítanie, násobenie skalárom, delenie skalárom.

Príklad 1.6 *Vytvorte riadkové vektory:* `v = [0 2 4 6 8]`, `u = [0 -1 -2 -3 -4]`, `z = [9 8 7]`. *Vyskúšajte vykonať nasledovné operácie a sledujte, čo je ich výsledkom:*

```
>> u+v
>> u-v
>> -2*v
>> v/3
>> -2*u+v/3
>> v(1:3)-v(2:4)
```

Potom skúste aj tieto a zistite, v čom je problém:

```
>> u + z
>> u + v'
```

2 Matice

Matica = dvojrozmerné pole

Pripomeňme si základné konvencie:

- čiarka alebo medzera oddeľuje prvky v riadku
- bodkočiarka oddeľuje riadky, t.j. definujú sa stĺpce
- celý zoznam prvkov je ohraničený hranatými zátvorkami

2.1 Definovanie matíc

2.1.1 Definovanie vymenovaním prvkov

```
>> A = [1 2 3; 4 5 6]            Vytvorí maticu typu 2 × 3
>> B = [1 2; 3 4; 5 6; 7 8 ];    Vytvorí maticu typu 4 × 2
```

Príklad 2.1 *Zadefinute maticu A ako stĺpec riadkov:*
`>> A = [ 1 2 3; 3 4 5; 6 7 8]`
Zadefinute maticu B ako riadok stĺpcov :
`>> B = [ [1 2 3]' [2 4 7]' [3 5 8]']`
alebo
`>> B = [ [1; 2; 3] [2; 4; 7] [3; 5; 8]]`

2.1.2 Definovanie pomocou zabudovaných funkcií

Matlab umožňuje generovať niektoré špeciálne typy matíc pohodlným spôsobom pomocou zabudovaných funkcií.

`zeros(n)`	Štvorcová matica núl typu $n \times n$
`zeros(m,n)`	Matica núl ľubovoľného rozmeru $m \times n$
`ones(n)`	Matica jednotiek $n \times n$
`ones(m,n)`	Matica jednotiek ľubovoľného rozmeru $m \times n$
`eye(n)`	Štvorcová matica identity
`diag`	Vytvorí diagonálnu maticu zo zadaného vektora, extrahuje diagonálu zo zadanej matice
`rand(m,n)`	Matica náhodných čísel typu $m \times n$
`randn(m,n)`	Matica náhodných čísel typu $m \times n$
`magic(n)`	Magický štvorec rozmerov $n \times n$

Príklad 2.2 *Pomocou príkazu* `diag` *zadefinujte maticu D, ktorá bude na diagonále obsahovať prvky 1,2,3,4 a všade inde budú nuly.*

2.2 Pristupovanie k elementom v matici

Príklad 2.3 *Vygenerujme opäť maticu* `A=[1 2 3; 4 5 6; 7 8 10]`. *Vyskúšajte, čo spravia príkazy:*

```
>> A(2,3)
>> B = A([1 3], [1 2])
>> C = A([3 2 1], :)
```

2.3 Zväčšovanie matíc

V Matlabe je možné pridávať prvky k matici a tým ju zväčšovať.

Príklad 2.4 *Vygenerujme opäť maticu A:*
```
>> A=[1 2 3; 4 5 6; 7 8 10]
```
Príkaz `A(4,4)` *vráti chybu. Avšak príkaz* `A(4,4)=12` *je možné vykonať. Výsledkom bude matica rozmerov* 4×4*, ktorá bude vo štvrtom riadku a vo štvrtom stĺpci obsahovať okrem posledného prvku samé nuly.*

2.4 Operátor dvojbodka

Operátor dvojbodka je jeden z najdôležitejsích operátorov v Matlabe. Vyskytuje sa v rôznych situáciách, ako napríklad:

`1:10`	Znamená postupnosť prirodzených čísel od 1 po 10
`A(:)`	Vytvorí vektorovú verziu matice A
`A(:,2)=[]`	Druhý stĺpec matice A bude vymazaný
`A(1:3,2)`	Stĺpcový vektor pozostávajúci z druhého stĺpca matice A
`sum(A(1:3,3))`	Vypočíta súčet prvkov tretieho stĺpca

2.5 Základné operácie na maticiach

Pri operáciách medzi vektormi a maticami je potrebné dbať na to, aby sedeli ich rozmery (tak, ako sme zvyknutí z lineárnej algebry).

Príklad 2.5 *Vykonajte nasledujúce príkazy:*
```
>> v = [0:2:8]
>> A*v(1:3)
```
Zvyknite si čítať **error messages** *a naučte sa rozumieť im!*

Rovnako ako v prípade vektorov, môžete pracovať aj s rôznymi časťam matíc. Opäť treba dávať pozor, či je operácia rozmerovo legálna.

Príklad 2.6 *Vyskúšajte si a porozumejte nasledujúcim príkazom:*
```
>> A(1:2,3:4)
>> A(1:2,2:3)
>> B = A(1:2,2:3)'
```

Syntax základných operácií s maticami:

`A+B`	
`A-B`	
`A*B`	`A.*B`
`A/B`	`A./B`
`A\B`	`A.\B`
`A^B`	`A.^B`
`A'`	`A.'`

Opis:

`A.*B`	Súčin po zložkách.
`A./B`	Delenie po zložkách. Prvky `A(i,j)/B(i,j)`
`A.\B`	Delenie po zložkách. Prvky `B(i,j)/A(i,j)`
`X^p`	Umocňovanie matice na skalár, mocnina matice.
`A.^B`	Umocňovanie matíc po zložkách. Prvky sú `A(i,j)` na `B(i,j)`.
`A'`	Transponovanie matice.

2.6 Maticové funkcie

`size`	rozmer matice
`eig`	vlastné čísla, vlastné vektory
`inv`	inverzná matica
`det`	determinant matice
`norm`	Euklidovská norma
`lu`	LU rozklad matice, $A = LU$, kde L je horná a U je dolná trojuh.mat.
`chol`	Choleského rozklad
`svd`	Singular-value rozklad matice A typu $m \times n$, resp. spektrálny rozklad symetrickej štvorcovej kladne definitnej matice
`qr`	QR rozklad matice

Príklad 2.7 *Inverzná matica k matici* `A = [1 2 3; 4 5 6; 7 8 9]`*:*

```
>> inv(A)
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 4.565062e-18
ans =
   1.0e+15 *

   -2.7022    4.5036   -1.8014
    5.4043   -9.0072    3.6029
   -2.7022    4.5036   -1.8014
```

Čo znamená vypísané varovanie a na čo teda treba dávať pozor pri počítaní inverznej matice?

Musíte byť opatrní, nakoľko tieto operácie sú numerické manipulácie vykonávané na digitálnom počítači. V tomto príklade matica `A` *nie je regulárna matica, ale Matlab aj napriek tomu vráti k nej „inverznú“ (ale nesprávnu, samozrejme).*

Príklad 2.8 *Vlastné čísla matice* `A`*:*

```
>> cisla = eig(A)
>> [vektory,cisla] = eig(A)
```

2.7 Riešenie systémov lineárnych rovníc

Maticové operácie sa uplatňujú napríklad pri riešení systémov lineárnych rovníc. Pri systéme $Ax = b$ so zadanou maticou A a vektorom b hľadáme taký vektor x, ktorý spĺňa zadanú rovnosť. Matica A môže byť pritom regulárna alebo singulárna.

Ak je matica A regulárna, možno riešenie nájsť ako $x = A^{-1}b$, teda pomocou príkazu
`>> x = inv(A)*b`
Matlab vie túto úlohu riešiť aj pomocou „opačného delenia“ `\` , a to spôsobom
`>> x = A \ b` Tento spôsob rieši úlohu pomocou metódy najmenších štvorcov, teda minimalizuje výraz $||Ax - b||^2$. Keďže v oboch prípadoch sa jedná o numerické algoritmy, je prirodzené, že získané výsledky sa môžu jemne líšiť (na vzdialených desatinných miestach).

Matlab vám umožní previesť uvedené príkazy aj v prípade, ak matica A nie je regulárna. V tom prípade však treba dať pozor na to, čo vlastne vypočítané výsledky predstavujú...

Výhodou druhého uvedeného spôsobu výpočtu je, že je výrazne rýchlejší ako počítanie inverznej matice v prvom spôsobe. To hrá dôležitú úlohu najmä pri veľkorozmerných úlohách. Ďalšou výhodou druhého postupu je aj to, že aj ak riešený systém nemá presné riešenie, týmto dostanete aspoň najbližšie možné riešenie v zmysle najmenších štvorcov.

Tretím spôsobom riešenia systému lineárnych rovníc v Matlabe je použitie zabudovanej funkcie linsolv:

```
>> x = linsolve(A,b)
```

Porovaniu rýchlostí všetkých troch uvedených spôsobov sa budeme venovať v Príklade 2.27.

2.8 Riedke matice

Riedka matica (angl. sparse matrix) je matica, ktorá vzhľadom na svoj rozmer obsahuje iba veľmi málo nenulových prvkov. Vedomosť o tom, že matica je riedka, môže najmä pri veľkorozmerných úlohách výrazne urýchliť niektoré výpočty a tiež znížiť ich pamäťovú náročnosť. Na definovanie riedkej matice sa používa príkaz

```
S = sparse(i,j,v)
```

kde i je vektor riadkových súradníc, j je vektor stĺpcových súradníc a v sú hodnoty, ktoré chceme na príslušné dvojice súradníc umiestniť. Pozri Príklad 2.28.

Riedka matica sa dá prekonvertovať na plnú pomocou príkazu

```
S = full(S)
```

Treba však mať na pamäti, že plná matica vdžy zaberá zbytočne viac pamäte, než jej riedka reprezentácia.

Ak z nejakého dôvodu máme riedku maticu zadefinovanú ako plnú, ale chceme Matlabu dať informáciu, že je riedka, môžeme ju prekonvertovať pomocou príkazu

```
S = sparse(S)
```

2.9 Keď treba poupratovať...

Niekedy je potrebné (alebo si želáme) vymazať všetky naše dáta, alebo vymazať iba nejakú konkrétnu premennú. Príkazom `clear` sa tak stane. Buďte opatrní, Matlab sa po zadaní tohto príkazu nepýta, či to chcete naozaj urobiť, a vymazanie premenných a ich hodnôt je definitívne.

Príklad 2.9 *Vyskúšajte urobiť nasledovné a všimnite si, ktoré premenné sa vymažú:*

```
>> clear A
>> clear all
```

2.10 Ďalšie príklady

Príklad 2.10 *Nech $A = magic(3)$. Čo vráti príkaz* `diag(diag(A))`*?*

Príklad 2.11 *Sčítajte prvky v jednotlivých stĺpcoch matice A. Použite príkaz* `sum` *a ak potrebujete, jeho použitie si overte v Helpe. Ako funguje príkaz* `sum` *použitý na maticu?*

Príklad 2.12 *Sčítajte prvky v riadkoch matice A.*

Príklad 2.13 *Z matice A vyberte do nového vektora diagonálu a potom opačnú diagonálu (z pravého horného do ľavého dolného rohu).*

Príklad 2.14 *Zadefinujte maticu*

```
1     2     3     1
5     4     2     4
1     7     4     0
2     1     5     3
```

a pomenujte ju M. Extrahujte z nej maticu N pozostávajúcu z prvých troch riadkov a prvých troch stĺpcov matice M. Vypočítajte vlastné čísla a vlastné vektort matice N a uložte ich do premenných v (vektor vlastných čísel) a V (matica, ktorej stĺpce sú vlastné vektory).

Príklad 2.15 *Zadefinujte matice A, B, C, D rozmerov* 4×4 *nasledovným spôsobom. Matica A sa rovná matici M z predošlého príkladu. Prvky matice B sú druhými mocninami prvkov matice A. Matica C vznikne transponovaním matice B. Matica D je inverzná k A. Napokon vytvorte vektor v, ktorý obsahuje diagonálne prvky matice* $A * B * C * D$.

Príklad 2.16 *Predstavte si, že potrebujete vygenerovať tabuľku, v ktorej v prvom stĺpci budú čísla 1 až 10, v druhom čísla* `1^2` *až* `10^2` *a v poslednom* `2^1` *až* `2^{10}`. *Skúste sami. Návod: definujte postupne všetky potrebné vektory a napokon ich „pozliepajte“ do jednej matice (tabuľky). Potom sa zamyslite, či sa váš postup dá ešte zjednodušiť.*

Príklad 2.17 *Matice môžeme vytvárať aj pomocou relácií a logických operátorov. Napríklad, nech*

```
>> x=[2.7 1.7 1.7 1.5 NaN 1.9 1.8 1.5 5.1 2.2 1.4 1.6 1.8];
```

V tomto prípade NaN označuje chýbajúce dáta. Aby sme odstránili chýbajúce dáta z vektora x, použijeme príkaz

```
>> x = x(finite(x))
```

Príklad 2.18 *Použijeme (očistený od NaN) vektor z predchádzajúceho príkladu, t.j.*

```
>> x=[2.7 1.7 1.7 1.5 1.9 1.8 1.5 5.1 2.2 1.4 1.6 1.8];
```

Chceme vylúčiť tie pozorovania, ktoré sa zdajú byť príliš odlišnými od ostatných, tzv. outlier-e. Nasledujúci príkaz odstraňuje tie prvky, ktoré sú väčšie alebo rovné 2:

```
>> x = x(x < 2)
```

Príklad 2.19 *Vytvorte riadkový vektor dĺžky 10, ktorého všetky prvky sa rovnajú dvom. Číslicu dva pritom môžete použiť najviac dvakrát.*

Príklad 2.20 *Vytvorte riadkový vektor* `a=[1 2 3 4 5]`. *Vytvorte vektor b, ktorý sa skladá z tých istých prvkov ako vektor a, len v obrátenom poradí.*

Príklad 2.21 *Nech A je štvorcová matica. Vytvorte maticu B, ktorá má tie isté prvky ako matica A, okrem prvkov na diagonále. Diagonála matice B pozostáva zo samých jednotiek.*

Príklad 2.22 *Nech* `A=round(10*rand(6))` *a matica B je definovaná ako* `B=A'`. *Zistite a vysvetlite, čo sa stane, keď zadáte príkaz:*

```
A(:,6)=-sum(B(1:5,:))'
```

Príklad 2.23 *Daný je vektor*
`x= [3,15,9,12,-1,0,-12,9,6,1]`. *Zostrojte vektor y, ktorý obsahuje prvky vektora x, ktoré sú väčšie ako 10. Použite pritom logické adresovanie (písanie podmienky do okrúhlych zátvoriek na miesto definovania vybraných pozícií).*

Príklad 2.24 *Singulárny rozklad obdĺžnikovej matice* `A = [1 2 3 4; 5 6 7 8; 9 5 2 6]` *na súčin* $A = USV$, *kde S obsahuje singulárne čisla matice:*

```
>> [U, S, V] = svd(A)
```

Všimnite si rozmery výstupných matíc a overte, či sú matice U a V naozaj ortogonálne. Ďalej si overte, že ak spravíme SVD rozklad matice AA^T

```
>> [K,L,M] = svd(AA')
```

tak platí, že matica $\sqrt{L}$ *a matica S majú v absolútnej hodnote rovnaké diagonálne prvky.*

Príklad 2.25 *Zadefinujte si štvorcovú symetrickú a kladne semidefinitnú maticu (tzn. má nezáporné vlastné hodnoty) a overte, že SVD rozklad je totožný so spektrálnym rozkladom tejto matice, teda že diagonálna matica z SVD rozkladu bude obsahovať vlastné hodnoty vstupnej matice.*

Príklad 2.26 *LU rozklad. Vyskúšajte si LU rozklad na štvorcovej matici* `A = [3 4; 1 2]`*:*
`>> [L,U] = lu(A)`
a všimnite si, že matica L je naozaj dolná trojuholníková a U horná trojuholníková, a tiež že platí $A = LU$*. Potom si vyskúšajte LU rozklad matice* `B = [1 1; 2 3]`*:*
`>> [L,U] = lu(B)` *a všimnite si, že L už nie je dolná trojuholníková, ale je permutovaná, k čomu došlo v priebehu faktorizácie Matlabom z dôvodu zabezpečenia stability výpočtu. Aj v tomto prípade však platí* $B = LU$*.*

Príklad 2.27 *Porovnanie časovej náročnosti troch spôsobov riešenia systému lineárnych rovníc. Zvoľte si regulárnu maticu A a pravú stranu b a vypočítajte riešenie systému* $Ax = b$ *troma rôznymi spôsobmi:*

```
>> x = inv(A)*b
>> y = A\ b
>> z = linsolve(A,b);
```

Nastavte si zobrazovanie čísel na long a porovnajte časové trvanie jednotlivých výpočtov pomocou umiestnenia dvojice príkazov

```
>> t1 = cputime;
>> t2 = cputime - t1
```

pred a za každý zo spôsobov (prvý príkaz pred, druhý za časť kódu, trvanie ktorej chcete merať). Skúste si meniť veľkosť matice od 100×100 *po* 2000×2000 *a sledujte rozdiel v časovej náročnosti použitých metód. Mali by ste vidieť, že výpočet inverznej matice je časovo oveľa náročnejší než druhé dve metódy. Preto sa na riešenie najmä veľkých systémov rovníc, alebo pri iterovanom opakovaní riešenia systému rovníc odporúča používať spôsob pomocou opačného lomítka prípadne pomocou príkazu linsolve, a vyhnúť sa tým počítaniu inverznej matice. Výpočet inverznej matice totiž vyžaduje oveľa väčší počet matematických operácií.*

Tip na flexibilné vytvorenie regulárnej matice a pravej strany s ľahko meniteľným rozmerom:

```
>> n = 2000;
>> A = ones(n,n) + diag(ones(n,1));
>> riesenie = 1:n;  % zvolene riesenie
>> b = A*riesenie'; % nastavenie pravej strany
```

a teraz sa môžete tváriť, že máte zadané A, b *a nepoznáte* x*, pričom viete, ako má riešenie* x *vyjsť.*

Príklad 2.28 *Riedke matice. Zadefinujte riedku maticu nasledujúcim spôsobom:*

```
>> riadky = [1, 3, 1, 7, 9];
>> stlpce = [1, 2, 3, 4, 5];
>> hodnoty = [ 10, 100, 12, 13, 14];
>> A = sparse(riadky, stlpce, hodnoty)
```

a všimnite si, ako sa spárovali riadkové a stĺpcové súradnice s hodnotami. Taktiež si všimnite, že matica sa nezobrazuje ako plná, ale iba ako zoznam nenulových hodnôt s ich pozíciami. Ak chcete zobraziť (ale nie predefinovať) maticu ako plnú, zadajte príkaz

```
>> full(A)
```

Na vyskúšanie príkazu `sparse` *si zadefinujte* `>> B = full(A)` *a následne môžete túto novovzniknutú plnú maticu znova previesť na riedku pomocou* `>> B = sparse(B)`*.*

Mnohé matlabovské zabudované funkcie (algoritmy) sú upravené aj na možnosť využiť riedkosť matice na urýchlenie výpočtov. Preto pokiaľ je matica riedka, určite sa oplatí ju mať aj uloženú ako riedku.