2-4-Linearna_algebra-S

K. Kumerički, verzija 1.1, 2011-10-25

Linearna algebra

O nizovima objekata (brojeva, simbola, ...) biti će više govora u slijedećem poglavlju, a ovdje nam treba samo nekoliko osnovnih ideja. Često ćemo sretati dvije vrste nizova objekata: *liste* (engl. list) i *tuplove* (engl. tuple). Liste su nizovi elemenata odvojeni zarezom u uglatim zagradama, a tuplovi su to isto samo u okruglim zagradama. Razlike između liste i tupla će biti razjašnjene kasnije; u ovom poglavlju će obje strukture funkcionirati bez razlika.

```
a = [1,1,2]      # lista
b = (2, 2, 4)    # tupl
```

Pristup pojedinim elementima se izvodi indeksom u uglatim zagradama (i za liste i za tuplove!), gdje je brojanje kao u C-u, tj. počinje od nule: prvi element liste a je a[0]:

```
print a[2]
b[0]
```

```
2
2
```

Ukoliko želimo napraviti pridruživanje gdje se vrijednosti uzimaju iz liste na slijedeći način je to moguće izvesti za cijelu listu odjednom. (Tzv. raspakiravanje.)

```
b1, b2, b3 = b; b2
```

```
2
```

Naravno, elementi listi mogu biti druge liste:

```
c = [a, b, [a]]
avec, bvec, [cvec] = c; cvec    # ovo je korisno za donji
```

```
c = [a, b, [a]]
avec, bvec, [cvec] = c; cvec    # ovo je korisno za donji
zadatak sa svojstvenim vrijednostima matrice
```

```
[1, 1, 2]
```

Vektori i matrice se konstruiraju pomoću funkcija vector() i matrix() kojima kao argument dajemo listu elemenata, odnosno listu listi elemenata (listu redak-vektora)

```
vec1 = vector([1,1,2])
vec2 = vector(b)   # moze i tupl
```

Množenje skalarom je prirodno:

```
3*vec1
```

```
(3, 3, 6)
```

Skalarni produkt vektora ...

```
vec1.inner_product(vec2)
```

```
12
```

... se može zapisivati kao i obično množenje:

```
vec1*vec2
```

```
12
```

Vektorski produkt (nema zapisa s $\times$!)

```
vec1.cross_product(vec2)
```

```
(0, 0, 0)
```

Norma ("duljina") vektora:

```
vec2.norm()
```

```
2*sqrt(6)
```

Svi vektori iste vrste i dimenzije su elementi apstraktnog vektorskog prostora koji je dostupan putem metode parent():

```
vecspace=vec1.parent(); vecspace
```

```
Ambient free module of rank 3 over the principal ideal domain
Integer Ring
```

Ovdje je važno uočiti da je je prostor definiran na "prstenu" cijelih brojeva. Naravno, vektori mogu biti i nad drugim prstenovima/poljima:

```
print vector([1/3, 1/4]).parent()
vector([1., 2.]).parent()
```

```
Vector space of dimension 2 over Rational Field
Vector space of dimension 2 over Real Field with 53 bits of
precision
```

Baza vektorskog prostora:

```
vecspace.basis()
```

```
[
(1, 0, 0),
(0, 1, 0),
(0, 0, 1)
]
```

Množenje matrica te množenje matrice i vektora ide na prirodan način:

```
mat = matrix([[1, 2, 1], [4, 3, 3], [9, 1, 7]]); mat
```

```
[1 2 1]
[4 3 3]
[9 1 7]
```

```
mat*mat
```

```
[18  9 14]
[43 20 34]
[76 28 61]
```

```
mat*vec1
```

```
(5, 13, 24)
```

```
~mat      # inverz matrice, moze i m.inverse() ili m^-1
```

```
[-18/7  13/7  -3/7]
[  1/7   2/7  -1/7]
[ 23/7 -17/7   5/7]
```

```
~mat*mat    # provjera
```

```
[1 0 0]
[0 1 0]
[0 0 1]
```

Matrice isto imaju svoje apstraktne prostore kojima pripadaju:

```
print mat.parent()
matinv.parent()
```

```
Full MatrixSpace of 3 by 3 dense matrices over Integer Ring
Traceback (click to the left of this block for traceback)
...
NameError: name 'matinv' is not defined
```

```
mat2 = matrix([[1., 2.], [3., 4.]])
print mat2.parent(); mat2
```

```
Full MatrixSpace of 2 by 2 dense matrices over Real Field with
53
bits of precision
[1.00000000000000 2.00000000000000]
[3.00000000000000 4.00000000000000]
```

Ovo defaultno polje realnih brojeva nije sasvim pogodno za numeričke račune pa je potrebno matrice s realnim koeficijentima kreirati uz eksplicitnu deklaraciju polja RDF ("real double field"):

```
mat2 = matrix(RDF, [[1., 2.], [3., 4.]])
print mat2.parent(); mat2
```

```
Full MatrixSpace of 2 by 2 dense matrices over Real Double Field
[1.0 2.0]
[3.0 4.0]
```

Pristup pojedinim elementima matrice se isto izvodi indeksiranjem:

```
mat[0, 0] = 0; mat
```

```
[0 2 1]
[4 3 3]
[9 1 7]
```

 Zadatak 2-4.1: Za datu matricu A definiramo svojstvene vektore (eigenvectors) v i njima pripadajuće svojstvene vrijednosti λ (eigenvalues) kao rješenja matrične jednadžbe

$$A\mathrm{v} = \lambda\mathrm{v} \ .$$

Odredite svojstvene vrijednosti i svojstvene vektore matrice

$$\begin{pmatrix} 2.3 & 4.5 \\ 6.7 & -1.2 \end{pmatrix},$$

i provjerite da dobivena rješenja zaista zadovoljavaju gornju jednadžbu.

 Zadatak 2-4.2: Kreirajte 3x3 matricu sa slučajnim realnim brojevima između 0 i 10. Invertirajte je i pomnožite s originalnom matricom te se uvjerite da dobijete jediničnu matricu.

Elementi vektora i matrica mogu biti i simboli:

```
var('x y z')
```

```
(x, y, z)
```

```
vec3 = vector([x, y, z])
vec3.norm()
```

```
sqrt(abs(x)^2 + abs(y)^2 + abs(z)^2)
```

Međutim, ukoliko pokušamo već stvorenom vektoru nad poljem nekih brojeva zamijeniti neki element simbolom, to ne ide:

```
vec1[2] = x
```

```
Traceback (click to the left of this block for traceback)
...
TypeError: unable to convert x (=x) to an integer
```

Riječ je o tome da su npr. čisto realni vektori (nad poljem RDF) interno reprezentirani kao C-polja radi optimizacije. Da bismo mogli napraviti ovo

što želimo trebamo prvo konvertirati vektor u simbolički vektor. Tu konverziju radi odgovarajući vektorski prostor nad simboličkim prstenom (SR, symbolic ring). Taj prostor dobijemo pomoću metode parent() vektora istog ranga, ali koji već jest simbolički. (Za detalje o takvim konverzijama vidi prvih par odjeljaka ovdje.)

```
vec3.parent()
```

```
Vector space of dimension 3 over Symbolic Ring
```

```
vec1_symb = vec3.parent()(vec1)  # konverzija vec1 u
simbolicki vektor
```

```
vec1_symb[1] = x; vec1_symb    # sad ide
```

```
(1, x, 2)
```

Zadatak 2-4.3 (*): Isto kao zadatak 2-4.3, ali prije invertiranja matrice zamijenite njen središnji element simbolom x. Nakon invertiranja i množenja uvrstite za x slučajni broj (random()) i uvjerite se da dobivate jediničnu matricu.

Dijagonalizacija matrice A je pronalaženje njenog rastava oblika

$$A = PDP^{-1}$$

gdje je D dijagonalna matrica. To se izvodi metodom eigenmatrix_right() koja vraća matrice P i D:

```
A = matrix(QQ, [[3, 1], [1, 3]])  # metoda is_diagonalizable
ne radi nad ZZ
print A.is_diagonalizable()
D, P = A.eigenmatrix_right(); (D, P)
```

```
True
(
[4 0]  [ 1  1]
[0 2], [ 1 -1]
)
```

```
A == P*D*(~P) # provjera
```

```
True
```

Treba uočiti da su elementi dijagonalne matrice D i stupci matrice P upravo svojstvene vrijednosti odnosno svojstveni vektori od A.

```
A.eigenvectors_right()
```

```
[(4, [
(1, 1)
], 1), (2, [
(1, -1)
], 1)]
```

Neke matrice nisu dijagonalizabilne, u slučaju čega će matrice P i D koje vraća eigenmatrix_right() i dalje zadovoljavati

$$AP = PD$$

ali P neće biti invertibilna:

```
A = matrix(QQ, [[1, 1], [0, 1]])
print A.is_diagonalizable()
D, P = A.eigenmatrix_right(); (D, P)
```

```
False
(
[1 0]  [1 0]
[0 1], [0 0]
)
```

```
A*P == P*D
```

```
True
```

```
~P
```

```
Traceback (click to the left of this block for traceback)
...
ZeroDivisionError: input matrix must be nonsingular
```

U takvim slučajevima od koristi može biti i vrlo popularni rastav na singularne vrijednosti (singular value decomposition, SVD):

$$A = USV^\dagger$$

gdje su U i V unitarne, a S dijagonalna matrica. (Jedino treba imati na umu da je odgovarajuća metoda SVD() trenutno implementirana samo za

matrice nad realnim RDF poljem pa je po potrebi potrebno prvo provesti konverziju matrice.)

```
U, S, V = matrix(RDF, A).SVD()
```

```
U*S*V.conjugate_transpose()
```

```
[              1.0               1.0]
[9.44340092235e-17               1.0]
```

Zadatak 2-4.4: Stupci matrice U u SVD rastavu su svojstveni vektori matrice $AA^\dagger$, a odgovarajuće svojstvene vrijednosti su kvadrati elemenata dijagonale matrice S. Uvjerite se u to eksplicitno na gornjem primjeru.

Literatura

Robert A. Beezer, A First Course in Linear Algebra, (Sage-enhanced textbook)