3D számítógépes geometria 2

Lineáris algebra alapok

Várady Tamás, Salvi Péter / BME

September 6, 2018

A mátrix

- $m \times n$ skalár érték:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix}$$

- $m = 1 \Rightarrow$ sorvektor, $n = 1 \Rightarrow$ oszlopvektor
- Julia:

```
julia> a = [1 2 3; 4 5 6]
2x3 Array{Int64,2}:
1 2 3
4 5 6
```

- Indexelés 1-től, pl. `a[2,1]` $\rightarrow 4$
- Oszlopfolytonos (column-major) tárolás

GSL

- Külön struktúra és függvények vektorokra és mátrixokra
- Háttérben mindkettő mögött egy „block” struktúra (összefüggő memóriaterület)
- Egy blockra több nézet (view) készíthető
- Indexelés 0-tól, sorfolytonos (row-major) tárolás
- Kezelés:
  - `gsl_matrix *a = gsl_matrix_alloc(2, 3);`
  - `gsl_matrix_set(a, 1, 2, x);`
  - `y = gsl_matrix_get(a, 1, 2);`
  - `gsl_matrix_free(a);`
  - vagy: `gsl_matrix_view a = gsl_matrix_view_array(arr, 2, 3);`
- Vektorokra hasonlóan (`gsl_vector_...`)

Eigen

- Template osztály: `Matrix<típus, m, n, sorrend>`
- Speciális méret a `Dynamic`, ha fordítási időben nem ismert
- Tárolási sorrend: `ColMajor` (default) / `RowMajor`
- Indexelés 0-tól
- Hasznos előre definiált típusok:
  - `typedef Matrix<double, Dynamic, Dynamic> MatrixXd;`
  - `typedef Matrix<int, Dynamic, 1> VectorXi;`
  - hasonlóan `(Vector|Matrix)[1-4X][ifd]`
- Kezelés:
  - `MatrixXd a(2,3);`
  - `a << 1, 2, 3, 4, 5, 6;`
  - `a(1,2) = x;`
  - `y = a(1,2);`

Speciális mátrixok

- Négyzetes: $m = n$
- Szimmetrikus: $a_{ij} = a_{ji}$
- Átlós (diagonális): $i \neq j \rightarrow a_{ij} = 0$
- Egység (identitás I): $a_{ij} = \delta_{ij}$
- Felső háromszög: $i > j \rightarrow a_{ij} = 0$
- Alsó háromszög: $j > i \rightarrow a_{ij} = 0$
- Sávos: $|i - j| > b \rightarrow a_{ij} = 0$
  - b a sávszélesség (bandwidth)
  - $b = 0$: átlós mátrix
  - $b = 1$: tridiagonális mátrix

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix}$$

- Ritka: a legtöbb elem 0 (más tárolás)
  - Eigen: `SparseMatrix` osztály, GSL: `gsl_spmatrix_*`
  - Julia: `spzeros(m,n)` / `sparse(I,J,vals)` [`SparseArrays`]

Elemenkénti műveletek

- Összeadás, kivonás
  - Julia: `+`, `-` (és `.+`, `.-`)
  - GSL: `gsl_matrix_add(a, b)`, `gsl_matrix_sub(a, b)`
  - Eigen: `+`, `-`
- Szorzás, osztás
  - Julia: `.*`, `./`
  - GSL: `gsl_matrix_mul_elements(a, b)`, `gsl_matrix_div_elements(a, b)`
  - Eigen: nincs (ld. `Array` osztály)
- Műveletek skalárokkal
  - Julia: `.+`, `.-`, `.*`, `./` (és `*`, `/`)
  - GSL: `gsl_matrix_scale(a, x)`, `gsl_matrix_add_constant(a, x)`
  - Eigen: `*`, `/` (összeadás/kivonás nincs, ld. `Array` osztály)

Szorzás

- $C = AB$ esetén (A: $m \times n$, B: $n \times l$, C: $m \times l$)

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \qquad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix} = \begin{bmatrix} 1 \cdot 7 + 2 \cdot 8 + 3 \cdot 9 \\ 4 \cdot 7 + 5 \cdot 8 + 6 \cdot 9 \end{bmatrix}$$

- Asszociatív, disztributív, de *nem* kommutatív
- Simán elvégezve $O(n^3)$, de lehet ügyesebben
  - Strassen-algoritmus: $O(n^{2.8})$
    - 2×2-es mátrixok 7 műveletes szorzásával
  - Kicsit kevésbé stabil
  - Van $O(n^{2.3})$ módszer is, de nagy konstans szorzóval
  - [Kitérő: bignum szorzásra hasonló elvű a Karatsuba algoritmus, $xy = (x_1 B + x_2)(y_1 B + y_2)$ 3 szorzással, $O(n^{1.6})$]

Szorzás (folyt.) & permutáció

- Szorzás könyvtárakkal:
  - Julia: `*`
  - GSL: BLAS függvényekkel, pl. $C = AB$:
    ```
    gsl_blas_dgemm(CblasNoTrans, CblasNoTrans, 1.0,
    &A.matrix, &B.matrix, 0.0, &C.matrix);
    ```
  - Eigen: `*`
- Permutációs mátrix: egységmátrix felcserélt sorokkal
  - Balról szorozva sorokat, jobbról szorozva oszlopokat cserél
  - Pl. a 2. és 3. sorokat ill. oszlopokat felcserélve:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a & b & c & d \\ k & l & m & n \\ w & x & y & z \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ w & x & y & z \\ k & l & m & n \end{bmatrix}$$

$$\begin{bmatrix} a & b & c \\ x & y & z \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} a & c & b \\ x & z & y \end{bmatrix}$$

Egyéb műveletek

- Inverz (négyzetes, nemszinguláris [$\det(A) \neq 0$] mátrixokra)
  - $AA^{-1} = I$ (számítása nem triviális, ld. később)
- Transzponált: A^T (átlóra tükrözött),
  Adjungált: A^* (konjugált transzponált)
  - Julia: `transpose(a)` és adjoint(a) / `a'`
  - GSL: `gsl_matrix_transpose(a)` [négyzetes mátrixra], `gsl_matrix_transpose_memcpy(b, a)`
  - Eigen: `a.transpose()` és `a.adjoint()`
- Nyom: az átlós elemek összege
  - Julia: `tr(a)` [`LinearAlgebra`]
  - Eigen: `a.trace()`
- Kiegészítés (augmentáció): egy mátrix „melléhelyezése"

```
julia> [[1 2 3; 4 5 6] [7 8; 9 10]]
2x5 Array{Int64,2}:
1 2 3 7 8
4 5 6 9 10
```

Ritka mátrixok tárolása

- Célok:
  - Kompakt tárolás
  - Hatékony vektorral való szorzás
- Triviális módszer
  - (i, j, a_{ij}) hármasok tárolása
  - C-ben érdemes három tömbbel
  - Tárolandó még:
    - Mátrix mérete (m, n)
    - Nem 0 elemek száma (NNZ)
  - Ax szorzat kiszámítása:

```
y = zeros(m)
for k = 1:nnz
    y[i[k]] += v[k] * x[j[k]]
end
```

Ritka mátrixok tárolása

- Tömörített sortárolás (Compressed Row Storage, CRS)
  - i helyett a sorok kezdőindexei + utolsó utáni index
- Példa:

$$A = \begin{bmatrix} 0 & a_1 & 0 & 0 \\ a_2 & 0 & a_3 & a_4 \\ 0 & a_5 & 0 & a_6 \\ 0 & a_7 & a_8 & a_9 \end{bmatrix}$$

$$\begin{aligned} \mathrm{row} &= 1, 2, 5, 7, 10 \\ \mathrm{j} &= 2, 1, 3, 4, 2, 4, 2, 3, 4 \\ v &= a_1, \ldots, a_9 \end{aligned}$$

- Ax szorzat kiszámítása:

```
y = zeros(m)
for i = 1:m
    for k = row[i]:row[i+1]-1
        y[i] += v[k] * x[j[k]]
    end
end
```

Vektornormák

- p-norma ($p \geq 1$):

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

  - Julia: `norm(v, p)` [`LinearAlgebra`]
  - GSL: `gsl_blas_dnrm2(v)` [2-es norma]
  - Eigen: `v.lpNorm<p>()` ill. `v.norm()` [2-es norma]

- Maximum norma:

$$\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|$$

  - Julia: `norm(v, Inf)`
  - Eigen: `v.lpNorm<Infinity>()`

Mátrixnormák

- Frobenius norma:

$$\|A\|_f = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}$$

  - Julia: `norm(a)` / `norm(a, 2)` [`LinearAlgebra`]
  - Eigen: `a.norm()`

- Oszlopösszeg és sorösszeg normák:

$$\|A\|_1 = \max_{1\le j\le n}\sum_{i=1}^{m}|a_{ij}|, \quad \|A\|_\infty = \max_{1\le i\le m}\sum_{j=1}^{n}|a_{ij}|$$

  - Julia: `opnorm(a, 1)` ill. `opnorm(a, Inf)` [`LinearAlgebra`]
  - Eigen: `a.lpNorm<1>()` ill. `a.lpNorm<Infinity>()`

Determináns

- Négyzetes mátrixra

$$\det(A) = |A| = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} a_{i\sigma_i}$$

  - S_n az $\{1 \dots n\}$ permutációinak halmaza
  - $\operatorname{sgn}(\sigma)$ a permutáció előjele (páratlan inverziószámra negatív)
- Könyvtárakkal:
  - Julia: `det(a)` [`LinearAlgebra`]
  - GSL: `gsl_linalg_LU_det(lu, permutation_signum)`
    - LU felbontás után hívható (ld. később)
  - Eigen: `a.determinant()`

Lináris egyenletrendszerek mátrix alakban

- Lineáris egyenletrendszer:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \vdots & \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

- Pl. egyenesek metszése, illesztés pontokra stb.
- Mátrix alak: $Ax = b$
- Homogén, ha $b = \mathbf{0}$
- Megoldás: $x = Ix = A^{-1}Ax = A^{-1}b$

Lináris egyenletrendszerek megoldása

- Ha $m > n \Rightarrow$ túlhatározott (pl. LSQ)
- Ha $m < n \Rightarrow$ alulhatározott (pl. optimalizálás)
- Megoldó módszerek (részleges lista):
  - Cramer-szabály
    - Determinánsok alapján (csak nagyon kis mátrixokra hatékony)
  - Gauss-elimináció – ld. később
  - LU-felbontás – ld. később
  - Cholesky-felbontás – ld. később
  - QR-felbontás
    - Felbontás egy ortogonális és egy felső háromszög mátrixra
- Megfelelő algoritmus a mátrix stuktúrája szerint

```
julia> [-1 2; 3 2] \ [2; 18]
2-element Array{Float64,1}:
4.0
3.0
```

$$\begin{bmatrix} -1 & 2 \\ 3 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 18 \end{bmatrix}$$

Nem négyzetes esetek

- Visszavezetjük a négyzetes esetre
- Túlhatározott: legkisebb négyzetes megoldás (LSQ)

$$\|Ax - b\|_2^2 = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j - b_i \right)^2 \to \min$$

  - Megoldandó: $(A^T A)x = A^T b$
  - Geometriailag: $\mathbb{R}^m$-ben b vetítése A oszlopainak alterére
- Alulhatározott: legkisebb normájú megoldás

$$\|x\|_2 \to \min$$

  - Megoldandó: $(AA^T)y = b$, és ebből $x = A^T y$
    - Bizonyítás: $(\hat{x} - x) \perp x$ könnyen belátható $\Rightarrow$
      $\|\hat{x}\|^2 = \|\hat{x} - x + x\|^2 = \|\hat{x} - x\|^2 + \|x\|^2 \geq \|x\|^2$
  - Geometriailag: $\mathbb{R}^n$-ben 0 vetítése a megoldások alterére

Naív algoritmus

- Két lépés: elimináció + visszahelyettesítés
- i-edik elimináció eltünteti az i-edik oszlopot az i-edik sor alatt
  - A j-edik sorból ($j > i$) levonjuk az i-edik sor a_{ji}/a_{ii}-szeresét
- Az eredmény ilyen alakú lesz:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a'_{22}x_2 + \cdots + a'_{2n}x_n &= b'_2 \\ \vdots & \vdots \\ a_{nn}^{(n-1)}x_n &= b_n^{(n-1)} \end{aligned}$$

- Ebből alulról felfele meghatározhatóak az x_i értékek (visszahelyettesítés)
- Julia implementáció – $O(n^3)$ ★

Példa – Elimináció

$$\begin{aligned} 2x_1 - 4x_2 + x_3 &= 12 \\ x_1 + 2x_2 - 3x_3 &= -12 \\ 6x_1 - 4x_2 + 5x_3 &= 36 \end{aligned}$$

Első sor félszeresét (1/2) ill. háromszorosát (6/2) levonjuk:

$$\begin{aligned} 2x_1 - 4x_2 + x_3 &= 12 \\ 4x_2 - 3.5x_3 &= -18 \\ 8x_2 + 2x_3 &= 0 \end{aligned}$$

Második sor kétszeresét (8/4) levonjuk:

$$\begin{aligned} 2x_1 - 4x_2 + x_3 &= 12 \\ 4x_2 - 3.5x_3 &= -18 \\ 9x_3 &= 36 \end{aligned}$$

Példa – Visszahelyettesítés

$$\begin{aligned} 2x_1 - 4x_2 \quad + x_3 &= 12 \\ 4x_2 - 3.5x_3 &= -18 \\ 9x_3 &= 36 \end{aligned}$$

Az utolsó sor alapján

$$x_3 = 36/9 = 4$$

Ekkor a második sor

$$4x_2 - 3.5 \cdot 4 = -18 \Rightarrow x_2 = (-18 + 14)/4 = -1$$

Végül az első sor

$$2x_1 - 4 \cdot (-1) + 4 = 12 \Rightarrow x_1 = (12 - 4 - 4)/2 = 2$$

Pivotálás

- Mindig az a_{ii} elemmel dolgoztunk
- Ha ez 0-közeli, akkor elszállhat
- Megoldás: megcseréljük a sorokat
- Mindig az oszlop legnagyobb abszolutértékű elemét vesszük (részleges pivotálás)

```
pivot = findmax(abs.(ab[i:n,i]))[2] + i - 1
if pivot != i
    ab[[i,pivot],:]  = ab[[pivot,i],:]
end
```

- Lehet oszlopok *és* sorok szerint nézni (teljes pivotálás)
  - De ritkán éri meg

Példa – részleges pivotálás

$$\begin{aligned} 2x_2+3x_3 &= 8 \\ 4x_1+6x_2+7x_3 &= -3 \\ 2x_1-3x_2+6x_3 &= 5 \end{aligned}$$

Az első oszlopban a maximum a második sorban levő 4

$$\begin{aligned} 4x_1+6x_2 \quad +7x_3 &= -3 \\ 2x_2 \quad +3x_3 &= 8 \\ -6x_2+2.5x_3 &= 6.5 \end{aligned}$$

A második oszlopban a maximum a harmadik sorban levő -6

$$\begin{aligned} 4x_1+6x_2 \quad +7x_3 &= -3 \\ -6x_2+2.5x_3 &= 6.5 \\ 3\frac{5}{6}x_3 &= 10\frac{1}{6} \end{aligned}$$

Egyéb felhasználás

- Determináns számítás
  - Háromszögmátrixnál az átlós elemek szorzata
  - Sorok egymáshoz adása nem változtatja meg
  - Használjuk a Gauss-eliminációból kapott háromszöget!
  - Pivotálásnál a cserék paritása szerint változik az előjel

$$D = a_{11} a'_{22} \cdots a^{(n-1)}_{nn} (-1)^p$$

- Inverz számítás
  - b vektor helyett az I oszlopaira oldjuk meg ($AA^{-1} = I$)
  - LU-felbontással egyszerűbb lesz
  - Julia: `inv(a)`, Eigen: `a.inverse()`, GSL: `gsl_linalg_LU_invert(lu, permutation, inv)`
- Sávos mátrixoknál egyszerűsödik az algoritmus
  - Tridiagonális különösen gyakori és egyszerű ★

Ismételt megoldás

- Ha több b-re kell megoldani ugyanazt a rendszert
  - Pl. görbeillesztésnél ugyanazt kell megoldani x, y, z-re
- $A = LU$
- L alsó háromszög (átlóban 1-esekkel), U felső háromszög
- Szokás egy mátrixban tárolni
- $Ax = LUx = b \Rightarrow$ megoldandó $Ld = b$, majd $Ux = d$
- Az U ugyanaz, amit a Gauss-elimináció ad
- Az L elemei az elimináció a_{ji}/a_{ii} szorzói
- Pivotálás itt is kell
- Ha a permutációs mátrix P:

$$PA = LU \;\Rightarrow\; Ld = Pb \;\Rightarrow\; Ux = d$$

LU a gyakorlatban

- Julia: `lu(a)` [LinearAlgebra]
  - Visszatérési érték: L, U, permutáció (pl. [3,1,2])
  - Megoldáshoz a \-t érdemes használni
- GSL: `gsl_linalg_LU_decomp(a, permutation, signum)`
  - $PA = LU$, `signum` -1 vagy 1 (cserék paritása alapján)
  - Ezután `gsl_linalg_LU_solve(lu, permutation, b, x)`
  - Vagy: `gsl_linalg_LU_svx(lu, permutation, bx)`
    (bx egyben bemeneti (b) és kimeneti (x) változó)
- Eigen: `a.fullPivLu()`
  - Visszatérési érték egy `FullPivLu` objektum
  - Ennek van egy `solve(b)` függvénye

Cholesky-felbontás & kondíciószám

- Cholesky-felbontás:
  - Speciális LU-felbontás szimmetrikus poz. definit mátrixokra
    - Pozitív definit: $\forall z \neq 0 : z^T Az > 0$ ($\Leftrightarrow$sajátértékek pozítívak)
    - Pl. $A^T A$, ha A oszlopai lin. függetlenek ($\Rightarrow$pl. LSQ)
  - $A = U^T U \Rightarrow U^T d = b$ és $Ux = d$
  - U nagyon könnyen számolható:

$$u_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} u_{ki}^2}, \quad u_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} u_{ki} u_{kj}}{u_{ii}}$$

- Kondíciószám:
  - $\|A\| \cdot \|A^{-1}\|$ (mindig ≥ 1) [konkrét érték függ a norma típustól]
  - A megoldás normájának relatív hibája arányos a kondíciószámmal: $\|\Delta x\|/\|x\| \leq \mathrm{Cond}(A) \cdot \|\Delta A\|/\|A\|$
  - Pl. A t jegyig pontos és $\mathrm{Cond}(A) \approx 10^c \Rightarrow x$ kb. 10^{c-t} jegyig

Gauss-Seidel iteráció

- Közelítő megoldás keresése finomítással ($O(n^2)$ / iteráció)
- Kezdőértékek kellenek (pl. $x_i^0 = 0$)
- Mindig a legfrisebb értékeket használja: ★

$$x_i^k = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^k - \sum_{j=i+1}^{n} a_{ij} x_j^{k-1} \right)$$

- Jacobi-iteráció: mindig az előző iteráció értékeit használja:

$$x_i^k = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{k-1} \right)$$

  - A Gauss-Seidel általában hatékonyabb

- Elégséges konvergencia-feltétel:
  diagonális dominancia $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$

Példa – Gauss-Seidel iteráció

$$\begin{aligned} 3x_1 - 0.1x_2 - 0.2x_3 &= 7.85 \\ 0.1x_1 + 7x_2 - 0.3x_3 &= -19.3 \\ 0.3x_1 - 0.2x_2 + 10x_3 &= 71.4 \end{aligned}$$

A pontos megoldás $x_1 = 3$, $x_2 = -2.5$, $x_3 = 7$.
Legyenek a kezdőértékek 0-k. Ekkor

$$\begin{aligned} x_1^{(1)} &= (7.85 + 0.1 \cdot 0 + 0.2 \cdot 0)/3 = 2.617 \\ x_2^{(1)} &= (-19.3 - 0.1 \cdot 2.617 + 0.3 \cdot 0)/7 = -2.795 \\ x_3^{(1)} &= (71.4 - 0.3 \cdot 2.617 + 0.2 \cdot (-2.795))/10 = 7.006 \\ x_1^{(2)} &= (7.85 + 0.1 \cdot (-2.795) + 0.2 \cdot 7.006)/3 = 2.991 \\ x_2^{(2)} &= (-19.3 - 0.1 \cdot 2.991 + 0.3 \cdot 7.006)/7 = -2.5 \\ x_3^{(2)} &= (71.4 - 0.3 \cdot 2.991 + 0.2 \cdot (-2.5))/10 = 7 \\ x_1^{(3)} &= (7.85 + 0.1 \cdot (-2.5) + 0.2 \cdot 7)/3 = 3 \end{aligned}$$

Relaxáció

- Az új értékeket összesúlyozzuk a régikkel

$$x_i^{\text{new}} = \lambda x_i^{\text{new}} + (1 - \lambda) x_i^{\text{old}}$$

- Alulrelaxálás: $0 < \lambda < 1$
  - Még nem konvergáló rendszernél
  - Oszcillálás csökkentésére
- Túlrelaxálás: $1 < \lambda < 2$
  - Már konvergáló rendszernél gyorsít
  - Successive Over-Relaxation (SOR)
- A λ beállítása nagyon feladatfüggő
  - Csak $\lambda \in (0, 2)$ értékekre konvergál

Konjugált gradiens

- Ritka mátrixú egyenletrendszerek megoldására
  - A mátrix csak vektorral való szorzásban (ritkára gyors)
  - De csak szimmetrikus pozitív definit mátrixokra
- Elv: $f(x) = \frac{1}{2}x^T Ax - x^T b \to \min \equiv \nabla f = Ax - b = 0$
- A k-adik iterációban p_k keresőirány (konjugált az előzőekre)
  - u és v konjugált A-ra nézve, ha $u^T Av = 0$
  - Keressük α_k-t, hogy $f(x_k + \alpha_k p_k) \to \min$
    - A minimumban p_k merőleges ∇f-re, ezt meg akarjuk tartani
  - $r_k = b - Ax_k$ az x_k approximáció hibavektora (reziduálisa)

$$\begin{aligned} r_0 = p_0 &= b - Ax_0 \\ \alpha_k &= r_k^T r_k / (p_k^T Ap_k) \\ r_{k+1} &= r_k - \alpha_k Ap_k \\ \beta_k &= r_{k+1}^T r_{k+1} / (r_k^T r_k) \\ p_{k+1} &= r_{k+1} + \beta_k p_k \end{aligned}$$

- $x_{k+1} = x_k + \alpha_k p_k$

Bikonjugált gradiens ★

- Tetszőleges mátrixra működik (nem csak szimm. poz. def.)
- Hasonló, de A^T alapú egyenletekkel kiegészített
- Négy vektormennyiség: r_k, $\bar{r}_k$, p_k, $\bar{p}_k$
  - $r_0 = \bar{r}_0 = p_0 = \bar{p}_0 = b - Ax_0$
- Iterációnként:

$$\begin{aligned}
\alpha_k &= \bar{r}_k^T r_k / (\bar{p}_k^T A p_k) \\
r_{k+1} &= r_k - \alpha_k A p_k \\
\bar{r}_{k+1} &= \bar{r}_k - \alpha_k A^T \bar{p}_k \\
\beta_k &= \bar{r}_{k+1}^T r_{k+1} / (\bar{r}_k^T r_k) \\
p_{k+1} &= r_{k+1} + \beta_k p_k \\
\bar{p}_{k+1} &= \bar{r}_{k+1} + \beta_k \bar{p}_k
\end{aligned}$$

- $x_{k+1} = x_k + \alpha_k p_k$

Definíciók

- A négyzetes mátrix; $Av = \lambda v$ esetén λ sajátérték
  - v a λ-hoz tartozó (jobboldali) sajátvektor [konstansszoros is]
- Az $(A - \lambda I)x = 0$ egyenlet nem triviális megoldását keressük

$$|A - \lambda I| = 0$$

- Karakterisztikus polinom gyökei a sajátértékek

$$\begin{bmatrix} 10 & -5 \\ -5 & 10 \end{bmatrix} \Rightarrow \begin{vmatrix} 10-\lambda & -5 \\ -5 & 10-\lambda \end{vmatrix} = \lambda^2 - 20\lambda + 75$$

$$\lambda_{1,2} = \frac{20 \pm \sqrt{20^2 - 4 \cdot 75}}{2} = \begin{cases} \lambda_1 & = 15 \\ \lambda_2 & = 5 \end{cases}$$

- Ez általában nem egy jó módszer a sajátértékek keresésére

Definíciók

- Ermitikus (Hermite-)mátrix: $A = A^*$ (konjugált transzponált), azaz $a_{ij} = a_{ji}^*$ (komplex konjugált) $\Rightarrow$ minden sajátérték valós
  - Speciális eset: valós szimmetrikus mátrix
- Ortogonális mátrix: $A^T = A^{-1}$
- Unitér mátrix: $A^* = A^{-1}$
  - Speciális eset: valós ortogonális mátrix
- Normál mátrix: $AA^* = A^*A$ (ermitikus/unitér$\rightarrow$normál)
  - A sajátvektorok ortogonálisak és kifeszítik a teret
  - Azonos sajátértékeknél nem feltétlenül, de található ilyen
    - Gram–Schmidt ortogonalizáció
- Megoldás könyvtárakkal:
  - Julia: `eigen(a)` [`LinearAlgebra`], Eigen: `a.eigenvalues()`
  - GSL: `gsl_eigen_symm(a, eigen, w)`
    + `gsl_eigen_symm_alloc(n)` / `gsl_eigen_symm_free(w)`
    - Sajátvektorokra `gsl_eigen_symmv*`

Hatvány-iteráció

- Közelítő algoritmus a legnagyobb sajátértékre
- Ha a legkisebb kell, az inverz mátrixra végezzük ($1/\lambda$ sajátértékei vannak)
- Kell egy kezdőérték a v-nek (pl. csupa 1, vagy random)
- Ez alapján kiszámoljuk a jobboldalt és normalizáljuk (úgy, hogy a max. érték 1 legyen)
- Megszorozzuk ezzel a mátrixot, tehát mindig $v^{k+1} = Av^k$ ★

$$\begin{bmatrix} 40 & -20 & 0 \\ -20 & 40 & -20 \\ 0 & -20 & 40 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 20 \\ 0 \\ 20 \end{bmatrix} \rightarrow 20 \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 40 & -20 & 0 \\ -20 & 40 & -20 \\ 0 & -20 & 40 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 40 \\ -40 \\ 40 \end{bmatrix} \rightarrow 40 \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$$

Hatvány-iteráció

$$\begin{bmatrix} 40 & -20 & 0 \\ -20 & 40 & -20 \\ 0 & -20 & 40 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 60 \\ -80 \\ 60 \end{bmatrix} \rightarrow -80 \begin{bmatrix} -0.75 \\ 1 \\ -0.75 \end{bmatrix}$$

$$\begin{bmatrix} 40 & -20 & 0 \\ -20 & 40 & -20 \\ 0 & -20 & 40 \end{bmatrix} \begin{bmatrix} -0.75 \\ 1 \\ -0.75 \end{bmatrix} = \begin{bmatrix} -50 \\ 70 \\ -50 \end{bmatrix} \rightarrow 70 \begin{bmatrix} -0.714 \\ 1 \\ -0.714 \end{bmatrix}$$

$$\begin{bmatrix} 40 & -20 & 0 \\ -20 & 40 & -20 \\ 0 & -20 & 40 \end{bmatrix} \begin{bmatrix} -0.714 \\ 1 \\ -0.714 \end{bmatrix} = \begin{bmatrix} -48.517 \\ 68.517 \\ -48.517 \end{bmatrix} \rightarrow 68.517 \begin{bmatrix} -0.708 \\ 1 \\ -0.708 \end{bmatrix}$$

- Valódi megoldás: $\lambda = 68.284$ és $v = \begin{bmatrix} -\frac{\sqrt{2}}{2} & 1 & -\frac{\sqrt{2}}{2} \end{bmatrix}$

A megoldás elve

- Átlós mátrix sajátértékei az átlóban vannak:

$$\det(A-\lambda I)=\begin{vmatrix} a_{11}-\lambda & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & a_{nn}-\lambda \end{vmatrix}=(a_{11}-\lambda)\cdots(a_{nn}-\lambda)$$

  - Alsó/felső háromszögnél szintén
- Hasonlósági transzformáció:

$$A \to Z^{-1}\cdot A\cdot Z$$

- Nem változtatja meg a sajátértékeket
- Cél: átlós alakra hozni (Z-ben lesznek a sajátvektorok)
  - pl. Jacobi és Householder transzformációk
  - Ha csak sajátértékek kellenek, elég az alsó/felső háromszög
    - QR-felbontás $\Rightarrow A^{(k+1)}=RQ=Q^T A^{(k)} Q \to$ felső háromszög

Jacobi-transzformáció

- Forgatási mátrixokkal (forgatás/tükrözés numerikusan stabil)

$$P_{pq} = \begin{bmatrix} 1 & & & & & & & & & & \\ & \ddots & & & & & & & & & \\ & & 1 & & & & & & & & \\ & & & \cos\phi & & & & \sin\phi & & & \\ & & & & 1 & & & & & & \\ & & & & & \ddots & & & & & \\ & & & & & & 1 & & & & \\ & & & -\sin\phi & & & & \cos\phi & & & \\ & & & & & & & & 1 & & \\ & & & & & & & & & \ddots & \\ & & & & & & & & & & 1 \end{bmatrix}$$

Jacobi-transzformáció

- Csak néhány sor/oszlop változik ($c = \cos\phi$, $s = \sin\phi$):

$$\begin{aligned}
a'_{rp} &= ca_{rp} - sa_{rq}, \quad r \notin \{p, q\} \\
a'_{rq} &= ca_{rq} + sa_{rp}, \quad r \notin \{p, q\} \\
a'_{pp} &= c^2 a_{pp} + s^2 a_{qq} - 2sca_{pq} \\
a'_{qq} &= s^2 a_{pp} + c^2 a_{qq} + 2sca_{pq} \\
a'_{pq} &= (c^2 - s^2)a_{pq} + sc(a_{pp} - a_{qq})
\end{aligned}$$

- Az átlón kívüli elemeket tudjuk törölni: $a'_{pq} = 0$-hoz

$$\theta = \cot 2\phi = \frac{c^2 - s^2}{2sc} = \frac{a_{qq} - a_{pp}}{2a_{pq}}$$

Jacobi-transzformáció

- A $t = \tan\phi = s/c$ jelölés bevezetésével

$$t^2 + 2t\theta - 1 = 0$$

- Az (abszolutértékben) kisebb gyök kell

$$t = \frac{\operatorname{sgn}(\theta)}{|\theta| + \sqrt{\theta^2 + 1}}$$

- Ha θ túl nagy (θ^2 túlcsordul), akkor legyen $t = 1/(2\theta)$
- Ebből pedig

$$\begin{aligned} c &= 1/\sqrt{t^2+1} \\ s &= tc \end{aligned}$$

- Konvergens ($\sum_{r\neq s} |a_{rs}|^2$ monoton csökken)

Jacobi-transzformáció

- Milyen sorrendben eliminálják az elemeket?
- Eredeti algoritmus (1846): mindig a legnagyobbat
  - Kézi számoláshoz jó
  - A keresés $O(n^2)$, míg maga a forgatás $O(n)$
- Szisztematikus, ciklikus bejárás
  - Pl. $P_{12}, P_{13}, \ldots, P_{1n}, P_{23}, P_{24}, \ldots$
- Javítások:
  - Az első három bejárásnál csak egy ϵ feletti elemeket eliminál

$$\epsilon = \sum_{r<s} |a_{rs}|/(5n^2)$$

  - Utána ha $|a_{pq}| \ll \min(|a_{pp}|, |a_{qq}|) \Rightarrow$ forgatás nélkül $a_{pq} = 0$

Householder-transzformáció

- Tridiagonális alakra hoz $n-2$ ortogonális transzformációval
- Householder mátrix: $P = I - 2w \cdot w^T$ (w valós egységvektor)
  - Ortogonális, mivel szimmetrikus és $P^2 = I$
  - Alternatív felírás: $P = I - u \cdot u^T/H$, ahol $H = |u|^2/2$, u tetsz.
- Ha $u = x \mp |x|e_1$, akkor $Px = \pm|x|e_1$
  - e_1 az első bázisvektor: $[1, 0, \ldots, 0]^T$
  - P töröl minden elemet, kivéve az elsőt
- Lépések:
  1. Az A első oszlopának alsó $n-1$ eleme alapján P_{n-1}
  2. Ezt kiegészítjük egy egységmátrix-darabbal $n \times n$-esre
  3. Eliminálunk az első sor/oszlop $n-2$ elemét
  4. Az A második oszlopának alsó $n-2$ eleme alapján P_{n-2}
  5. Ezt kiegészítjük egy egységmátrix-darabbal $n \times n$-esre
  6. stb.

Householder-transzformáció – Példa

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

$$P_3 = \begin{bmatrix} -1/3 & 2/3 & -2/3 \\ 2/3 & 2/3 & 1/3 \\ -2/3 & 1/3 & 2/3 \end{bmatrix}, \quad Q_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1/3 & 2/3 & -2/3 \\ 0 & 2/3 & 2/3 & 1/3 \\ 0 & -2/3 & 1/3 & 2/3 \end{bmatrix}$$

$$A_1 = Q_1 A Q_1 = \begin{bmatrix} 4 & -3 & 0 & 0 \\ -3 & 10/3 & 1 & 4/3 \\ 0 & 1 & 5/3 & -4/3 \\ 0 & 4/3 & -4/3 & -1 \end{bmatrix}$$

Householder-transzformáció – Példa

$$A_1 = \begin{bmatrix} 4 & -3 & 0 & 0 \\ -3 & 10/3 & 1 & 4/3 \\ 0 & 1 & 5/3 & -4/3 \\ 0 & 4/3 & -4/3 & -1 \end{bmatrix}$$

$$P_2 = \begin{bmatrix} -3/5 & -4/5 \\ -4/5 & 3/5 \end{bmatrix}, \quad Q_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -3/5 & -4/5 \\ 0 & 0 & -4/5 & 3/5 \end{bmatrix}$$

$$A_2 = Q_2 A_1 Q_2 = \begin{bmatrix} 4 & -3 & 0 & 0 \\ -3 & 10/3 & -5/3 & 0 \\ 0 & -5/3 & -33/25 & -68/75 \\ 0 & 0 & -68/75 & 149/75 \end{bmatrix}$$

Householder-transzformáció

- Az $A' = Q \cdot A \cdot Q$ szorzás elvégzése helyett legyen

$$\begin{aligned} p &= (Au)/H \\ K &= (u^T p)/(2H) \\ q &= p - Ku \end{aligned}$$

- Ekkor

$$A' = A - qu^T - uq^T,$$

  ami hatékonyabban számolható
- Érdemes a transzformációt megcsinálni a QR használata előtt, sokkal gyorsabb lesz, iterációnként $O(n^3)$ helyett $O(n)$
  - pl. Eigen: `a.fullPivHouseholderQr()`

Szinguláris érték szerinti felbontás (SVD)

- A sajátérték fogalmának bővítése tetszőleges alakú mátrixokra

$$A = U\Sigma V^*$$

- U és V unitér mátrixok ($U^* = U^{-1}$), méretük $m \times m$ ill. $n \times n$
  - U az AA^*, V az A^*A sajátvektoraiból áll
- Σ egy $m \times n$ átlós mátrix nemnegatív valós értékekkel
  - „szinguláris értékek" – az AA^* (és A^*A) sajátértékei
  - V oszlopai a hozzájuk tartozó (jobboldali) szinguláris vektorok
- Kiszámítás:
  - Bidiagonális alakra hozzuk A-t, majd iteratív algoritmus...
  - Julia: `svd(a)` [`LinearAlgebra`]
  - GSL: `gsl_linalg_SV_decomp(A, V, S, work)` [`A` ← `U`]
  - Eigen: `a.jacobiSvd()` / `a.bdcSvd()`
- Felhasználás:
  - LSQ illesztés, pszeudoinverz, főkomponens-analízis, tömörítés...
  - Eckart–Young-tétel: $U\hat{\Sigma}V^*$ a legjobb k-adrangú közelítés, ahol $\hat{\Sigma} = \Sigma$, de $\hat{\Sigma}_{ii} = 0$ minden $i > k$-ra