Mátrix inverz számítása C programozási nyelven

Pálóczy Makszim

Inverz Mátrix

Az inverz mátrix (A^{-1}) egy olyan négyzetes mátrix, amely megszorozva az eredeti mátrixszal (A) az egységmátrixot (E) adja.
Csak akkor létezik, ha a mátrix determinánsa nem nulla, különben szinguláris.

Mátrix inverzének lépései

1. Kibővített mátrix létrehozása: Készítsünk egy mátrixot [A | E], ahol A a kiinduló mátrix, E az egységmátrix.
2. Pivot kiválasztása: Minden oszlopban keressük meg a legnagyobb abszolút értékű elemet (pivot) a stabilitás érdekében.
3. Sorcsere: Ha a pivot nem az aktuális sorban van, cseréljük fel a pivot sort az aktuális sorral.
4. Pivot sor normalizálása: Osszuk el a pivot sort a pivot értékével, hogy a főátlóban 1 legyen.
5. Eliminációó: Minden más sorból vonjuk ki a pivot sor megfelelő többszörösét, hogy az adott oszlopban mindenhol 0 legyen kivéve a pivotot.
6. Inverz kimásolása: A Gauss–Jordan végén a kibővített mátrix jobb oldala már az inverz mátrix, ezt másoljuk át az eredmény mátrixba.
7. Eredmény ellenőrzése: Ha a pivot érték túl kicsi bármely lépésnél → a mátrix szinguláris, nincs inverze.

C program: Memóriafoglalás

- Dinamikusan foglalunk memóriát egy n × m méretű mátrixnak.
- Ez azért fontos, mert a mátrix mérete futásidőben változhat.
- Ha bármelyik sor foglalása sikertelen, visszafoglaljuk a már lefoglalt memóriát (free) és visszatérünk NULL-lal.
- A freeMatrix felszabadítja a mátrix összes sorát és magát a mátrixot is.

```c
/* Memória segédfüggvények */
double **allocMatrix(int n, int m) { // n sor, m oszlop
    double **mat = malloc(n * sizeof(double*));
    if (!mat) return NULL;
    for (int i = 0; i < n; i++) {
        mat[i] = malloc(m * sizeof(double));
        if (!mat[i]) {
            for (int k = 0; k < i; k++) free(mat[k]);
            free(mat);
            return NULL;
        }
    }
    return mat;
}

void freeMatrix(double **mat, int n) {
    if (!mat) return;
    for (int i = 0; i < n; i++) free(mat[i]);
    free(mat);
}
```

C program: Kibővített mátrix [A | I] létrehozása

- A bal oldal az eredeti mátrix, a jobb oldal az egységmátrix.
- Ez a kiindulópont a Gauss–Jordan módszerhez, mert a cél: a bal oldalból az egységmátrixot csináljuk, miközben a jobb oldal lesz az inverz.

```c
int invert(double **A, double **inv, int n) {
    // 1. Létrehozzuk a kibővített mátrixot: [ A | I ]
    // Mérete: n sor, 2*n oszlop
    double **aug = allocMatrix(n, 2 * n);
    if (!aug) return 0;

    // Feltöltés
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            aug[i][j] = A[i][j];         // Bal oldal: A mátrix
            aug[i][j + n] = (i == j) ? 1.0 : 0.0; // Jobb oldal: Egységmátrix
        }
    }
```

C program: Pivot kiválasztás

- Minden oszlopban a legnagyobb abszolút értékű elemet választjuk pivotnak.
- Ha a pivot értéke túl kicsi (közel nulla), a mátrix szinguláris → nincs inverze.

```c
// 2. Gauss-Jordan Elimináció
for (int col = 0; col < n; col++) {

    // --- Pivot keresés (részleges pivotálás) ---
    int pivot_row = col;
    double max_val = fabs(aug[col][col]);

    for (int r = col + 1; r < n; r++) {
        if (fabs(aug[r][col]) > max_val) {
            max_val = fabs(aug[r][col]);
            pivot_row = r;
        }
    }

    // Ha a főelem túl kicsi, a mátrix szinguláris
    if (max_val < 1e-12) {
        freeMatrix(aug, n);
        return 0;
    }
```

C program: Sorcsere (ha szükséges)

- Ha a legnagyobb pivot nem az aktuális sor → sorokat felcseréljük.
- A csere pointerekkel történik → gyors és memóriahatékony.

```c
// --- Sorcsere (Pointerek cseréje) ---
if (pivot_row != col) {
    double *temp = aug[col];
    aug[col] = aug[pivot_row];
    aug[pivot_row] = temp;
}
```

C program: Pivot sor normalizálása

- A pivot sor minden elemét elosztjuk a pivot értékével → a főátlóban 1 lesz.
- Ez szükséges ahhoz, hogy a többi sorból könnyen nullákat csinálhassunk az adott oszlopban.

```
// --- Normalizálás (Főátló legyen 1) ---
double pivot = aug[col][col];

// Végigmegyünk az EGÉSZ soron (bal és jobb térfélen is egyszerre)
// Optimalizálás: kezdhetjük 'col'-tól, mert előtte már nullák vannak
for (int j = col; j < 2 * n; j++) {
    aug[col][j] /= pivot;
}
```

C program: Eliminációó

- Minden sorból kivonjuk a pivot sor megfelelő többszörösét → az aktuális oszlopban mindenhol 0 lesz kivéve a pivotot.
- Ez az egész bal oldali mátrixot egységmátrixszá alakítja.
- Eközben a jobb oldalon kialakul az inverz mátrix.

```c
// --- Elimináció (Többi sor nullázása) ---
for (int i = 0; i < n; i++) {
    if (i == col) continue; // A pivot sort kihagyjuk

    double factor = aug[i][col];

    // Kivonjuk a pivot sor megfelelő többszörösét
    // Itt is mehetünk 'col'-tól 2*n-ig
    for (int j = col; j < 2 * n; j++) {
        aug[i][j] -= factor * aug[col][j];
    }
}
```

C program: Inverz mátrix kimásolása

- A kibővített mátrix jobb oldala most már az inverz mátrix.
- Átmásoljuk a inv mátrixba → felhasználható további számításokhoz.
- Felszabadítjuk a kibővített mátrix memóriáját.

```
// 3. Eredmény kimásolása a jobb oldalról
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        inv[i][j] = aug[i][j + n];
    }
}

freeMatrix(aug, n);
return 1;
```

C program: Felhasználói beolvasás és kiírás

```c
int main(void) {
    int n;

    // Output buffer kikapcsolása (konzol hiba elkerülése végett)
    setvbuf(stdout, NULL, _IONBF, 0);

    printf("Add meg a matrix meretet (n): ");
    if (scanf("%d", &n) != 1 || n <= 0) {
        printf("Hibas meret.\n");
        return 1;
    }

    double **A = allocMatrix(n, n);
    double **inv = allocMatrix(n, n);

    if (!A || !inv) {
        printf("Memoria hiba.\n");
        return 1;
    }
    printf("Add meg az A matrix elemeit (%dx%d):\n", n, n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%lf", &A[i][j]);
        }
    }

    if (invert(A, inv, n)) {
        printf("\nAz A matrix inverze:\n");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                printf("%10.5f ", inv[i][j]);
            }
            printf("\n");
        }
    } else {
        printf("HIBA: A matrix nem invertalhato (szingularis).\n");
    }

    freeMatrix(A, n);
    freeMatrix(inv, n);

    return 0;
}
```