LUP dekompozicija

Projektni izvještaj

Opis problema

U praksi je često potrebno riješiti sustav linearnih jednadžbi. Sustav n linearnih jednadžbi s n nepoznanica izgleda ovako:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n, \end{aligned}$$

gdje su $x_1, \ldots, x_n$ nepoznanice, a a_{ij} i b_i, $i, j \in \{1, \ldots, n\}$ poznati realni (ili kompleksni) brojevi. Rješenje sustava su $x_1, x_2, ..., x_n$ koji zadovoljavaju svaku od n jednadžbi.

Iz definicije množenja matrica, lako je vidjeti da gornji sustav možemo zapisati u matričnoj formi, tj. kao matričnu jednadžbu:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

Označimo matricu s elementima a_{ij} s A, a jednostupčane matrice, tj. vektore, s elementima x_i i b_i redom s x i b. Tada gornju jednadžbu možemo zapisati kao:

$$Ax = b$$

Dakle, rješenje sustava linearnih jednadžbi je n-dimenzionalni vektor x koji zadovoljava gornju jednadžbu.

Sada ćemo pokazati da ukoliko je za sustav n linearnih jednadžbi s n nepoznanica matrica A regularna (regularne matrice su potklasa kvadratnih matrica, pa to ima smisla), tada rješenje sustava postoji i ono je jedinstveno.

Pretpostavimo da je kvadratna n-dimenzionalna matrica sustava A regularna, dakle, po definiciji, ima inverznu matricu, tj. postoji kvadratna n-dimenzionalna matrica A^{-1} takva da vrijedi:

$$AA^{-1} = A^{-1}A = I,$$

gdje je $I = I_n$ n-dimenzionalna jedinična matrica.

Sada jednadžbu

$$Ax = b$$

možemo pomnožiti s lijeve strane matricom A^{-1}:

$$A^{-1}Ax = A^{-1}b,$$

pa, iz definicije inverza, dobivamo:

$$Ix = A^{-1}b,$$

tj.

$$x = A^{-1}b.$$

Dakle, rješenje $x = A^{-1}b$ sigurno postoji.

Iz ekvivalencije gornjih jednadžbi i jedinstvenosti inverza matrice možemo zaključiti i da je to jedino rješenje, međutim jedinstvenost možemo dokazati i tako da pretpostavimo suprotno i dođemo do kontradikcije.
Pretpostavimo da postoje dva različita rješenja jednadžbe $Ax = b$, x' i x''. Dakle, vrijedi $Ax' = b$ i $Ax'' = b$, iz čega slijedi $Ax' = Ax''$.
Sada, koristeći regularnost matrice A i asocijativnost matričnog množenja, možemo pisati sljedeće jednakosti:

$$x' = Ix' = (A^{-1}A)x' = A^{-1}(Ax') = A^{-1}(Ax'') = (A^{-1}A)x'' = Ix'' = x''$$

iz čega, zbog tranzitivnosti relacije =, dobivamo da vrijedi $x' = x''$, što je u kontradikciji s pretpostavkom da su x' i x'' različiti. Dakle, zaključujemo da je rješenje sustava $Ax = b$ jedinstveno.

Napomenimo sada još da se sustav linearnih jednadžbi može sastojati od manje jednadžbi nego što one imaju nepoznanica. Za takve sustave kažemo da su nedovoljno određeni i oni najčešće imaju beskonačno mnogo rješenja. Također, sustav linearnih jednadžbi može se sastojati od više jednadžbi nego što one imaju nepoznanica. Za takve sustave kažemo da su oni preodređeni. Budući da je velika vjerojatnost da nisu sve jednadžbe međusobno konzistentne, takvi sustavi najčešće nemaju rješenje. Budući da **algoritme ima smisla promatrati samo za probleme koji imaju jedinstveno rješenje** (jer onda znamo da rješenje postoji, pa ga ima smisla tražiti, ali i znamo točno koje rješenje tražimo), mi ćemo **promatrati samo linearne sustave s n jednadžbi i n nepoznanica za koje je matrica A regularna**.

Neke moguće metode rješavanja

Iz gore pokazanog, očito je da se rješenje suatava n linearnih jednadžbi s n nepoznanica može dobiti računanjem inverza matrice sustava, dakle računanjem matrice A^{-1}, te zatim računanjem produkta $A^{-1}b$. Međutim, standardno računanje inverza matrice u praksi je numerički nestabilno, pa se ta opcija izbjegava.

Sjetimo se da se linearni sustavi mogu efikasno rješavati tzv. metodom Gaussovih eliminacija.

Linearni sustav najprije se svede na trokutasti oblik na sljedeći način. Ukoliko fiksiramo neki uređaj i nepoznanica i jednadžbi, najprije se iz svih jednadžbi osim prve ukloni prva nepoznanica, tako da se, transformacijama jednadžbi sustava koje sustav ostavljaju ekvivalentnim, koeficijenti uz prvu nepoznanicu u tim jednadžbama postave na 0. To se očito postiže tako da se, za svaku jednadžbu osim prve, prva jednadžba pomnoži skalarom koji je jednak

$$\frac{\text{koeficijent uz prvu nepoznanicu u odgovarajućoj jednadžbi}}{\text{koeficijent uz prvu nepoznanicu u prvoj jednadžbi}},$$

ili, jasnije, u skladu s gornjim oznakama, skalarom $\frac{a_{1k}}{a_{11}}$ za k-tu jednadžbu, $k \in \{2, \ldots, n\}$, i zatim oduzme od te iste jednadžbe (sjetimo se da su množenje jednadžbe skalarom i oduzimanje jednadžbi

transformacije koje sustave ostavljaju ekvivalentima, tj. transformirani sustav ima isti skup rješenja kao i početni).
Dalje, analogno se iz svih jednadžbi osim prve i druge ukloni druga nepoznanica odgovarajućim množenjem i oduzimanjem druge jednadžbe, iz svih jednadžbi osim prve tri ukloni se treća nepoznanica, i tako dalje dok sustav ne postane gornjetrokutast.
Budući da su ekvivalentni, rješenje početnog sustava jednako je rješenju dobivenog trokutastog sustava, a rješenje gornjetrokuastog sustava može se jednostavno dobiti tzv. supstitucijom unatrag, koju ćemo detaljno pokazati kasnije.

Može se pokazati da je složenost Gaussovih eliminacija $\in O(n^3)$, gdje je veličina zadaće n dimenzija matrice sustava.

Sada ćemo predstaviti metodu rješavanja linearnih sustava korištenjem LUP dekompozicije i ujedno pokazati **prednost korištenja te metode u odnosu na metodu Gaussovih eliminacija**.

LUP dekompozicija

LUP dekompoziciju kvadratne n-dimenzionalne matrice A čine tri kvadratne n-dimenzionalne matrice L, U i P takve da vrijedi

$$PA = LU$$

i pri čemu je L donjetrokutasta matrica s jedinicama na dijagonali, a U gornjetrokutasta matrica, a P je matrica permutacije, tj. permutirana (u smislu redaka) jedinična matrica I.

Množenje matrice A matricom permutacije s lijeve strane permutira retke matrice A na način na koji su permutirani retci matrice I da bi se dobila matrica P. Ovo je zapravo faktorizacija, tj. zapis, permutirane matrice A kao produkt donjetrokutaste jedinične matrice i gornjetrokutaste matrice.

Budući da nas takve matrice ovdje zanimaju, prilikom opisa same metode pokazat ćemo da **svaka regularna matrica ima LUP dekompoziciju**.

Posljedica poznavanja LUP dekompozicije matrice sustava

Kao drugu stvar važnu za rješavanje linearnih sustava s regularnom matricom, pokazat ćemo kako se linearni sustav može riješiti ako najprije napravimo LUP dekompoziciju matrice sustava, A.

Pretpostavimo da je zadan sustav linearnih jednadžbi s regularnom matricom

$$Ax = b$$

te da je dobivena LUP dekompozicija matrice A:

$$PA = LU.$$

Sada, ako pomožimo jednadžbu

$$Ax = b$$

s lijeve strane matricom P, budući da permutiranje (promjena poretka) jednadžbi sustava kreira sustav ekvivalentan početnom, dobivamo sljedeću ekvivalentnu jednadžbu:

$$PAx = Pb.$$

Sada, koristeći dekompoziciju $PA = LU$, možemo pisati:

$$LUx = Pb.$$

Ukoliko imamo u vidu asocijativnost množenja matrica, možemo pisati

$$L(Ux) = Pb$$

i označiti

$$y = Ux$$

(primijetimo da je Ux jednostupčana matrica, dakle vektor), pa dobivamo sustav

$$Ly = Pb.$$

Dakle, očito, ukoliko imamo LUP dekompoziciju matrice A, da bismo dobili rješenje sustava x, dovoljno je riješiti sustav $Ly = Pb$, i time izračunati vektor y, a zatim riješiti sustav $Ux = y$.
Ovo jesu nova dva sustava linearnih jednadžbi, ali oni su **trokutasti, pa se mogu vrlo jednostavno i efikasno riješiti**, što ćemo pokazati u nastavku.

Sustav $Ly = Pb$ je **donjetrokutast**, a takav se sustav rješava tzv. **suspstitucijom unaprijed**.

Zapišimo $Ly = Pb$ kao:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} b_{P(1)} \\ b_{P(2)} \\ \vdots \\ b_{P(n)} \end{bmatrix}.$$

Što je, po definiciji matričnog množenja, ekvivalentno sljedećem sustavu jednadžbi:

$$\begin{aligned} y_1 &= b_{P(1)} \\ l_{21}y_1 + y_2 &= b_{P(2)} \\ l_{31}y_1 + l_{32}y_2 + y_3 &= b_{P(3)} \\ &\vdots \\ l_{n1}y_1 + l_{n2}y_2 + \cdots + y_n &= b_{P(n)}, \end{aligned}$$

što je ekvivalentno sljedećim jednadžama:

$$\begin{aligned} y_1 &= b_{P(1)} \\ y_2 &= b_{P(2)} - l_{21}y_1 \\ y_3 &= b_{P(3)} - (l_{31}y_1 + l_{32}y_2) \\ &\vdots \\ y_n &= b_{P(n)} - (l_{n1}y_1 + l_{n2}y_2 + \cdots + l_{n,n-1}y_{n-1}). \end{aligned}$$

Dakle, iz ovoga očito zaključujemo da rješenje y ovog sustava možemo dobiti računajući komponente redom odozgo prema dolje, tj. od prve do n-te, pri čemu za izračun svake od njih koristimo sve dotad izračunate komonenete. Očito, opća formula glasi:

$$y_i = b_{P(i)} - \sum_{j=1}^{i-1} l_{ij}y_j, \qquad \forall\, i \in \{1, \ldots, n\}.$$

Ovo se može napisati kao sljedeći kod:

```
    for(int i = 0; i < n; i++)
    {
        sum = 0.0;
        for(int j = 0; j < i-1; j++)
            sum = sum + L[i][j] * x[j];

        x[i] = b[P[i]] - sum;
    }
```

Sustav $Ux = y$ je **gornjetrokutast**, a takav se sustav rješava tzv. **povratnom ili suspstitucijom unatrag**.

Zapišimo $Ux = y$ kao:

$$\begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

Što je, po definiciji matričnog množenja, ekvivalentno sljedećem sustavu jednadžbi:

$$\begin{aligned} u_{11}x_1 + u_{12}x_2 + \cdots + u_{1n}x_n &= y_1 \\ u_{22}x_2 + u_{23}x_3 + \cdots + u_{2n}x_n &= y_2 \\ &\vdots \\ u_{n-2,n-2}x_{n-2} + u_{n-2}x_{n-1} + u_{n-2,n}x_n &= y_{n-2} \\ u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n &= y_{n-1} \\ u_{nn}x_n &= y_n, \end{aligned}$$

što je ekvivalentno sljedećim jednadžama:

$$\begin{aligned} x_n &= y_n/u_{nn} \\ x_{n-1} &= (y_{n-1} - u_{n-1,n}x_n)/u_{n-1,n-1} \\ x_{n-2} &= (y_{n-2} - u_{n-2}x_{n-1} - u_{n-2,n}x_n)/u_{n-2,n-2} \\ &\vdots \\ x_2 &= (y_2 - u_{23}x_3 - \cdots - u_{2n}x_n)/u_{22} \\ x_1 &= (y_1 - u_{12}x_2 - \cdots - u_{1n}x_n)/u_{11}. \end{aligned}$$

Dakle, iz ovoga očito zaključujemo da rješenje x ovog sustava možemo dobiti računajući komponente redom odozdo prema gore, tj. od n-te do prve, pri čemu za izračun svake od njih koristimo sve dotad izračunate komponenete. Očito, opća formula glasi:

$$x_i = \left(y_i - \sum_{j=i+1}^{n} u_{ij}x_j \right) / u_{ii}, \qquad \forall\, i \in \{1, \ldots, n\}.$$

Ovo se može napisati kao sljedeći kod:

```
1    for(int i = n - 1; i >= 0; i--)
2    {
3        sum = 0.0;
4        for(int j = i + 1; j < n; j++)
5            sum = sum + U[i][j] * x[j];
6
7        x[i] = (x[i] - sum) / U[i][i];
8    }
```

Obje gornje petlje mogu se smjestiti u jednu proceduru, nazovimo je LUP_SOLVE, koja će dakle izračunavati rješenje sustava x, ukoliko nam je poznata dekompozicija matrice A, dakle odgovarajuće matrice L, U i P.

Napomena. Uočimo da se, budući da se točno jedna koordinata vektora y koristi za računanje točno jedne koordinate vektora x, u svrhu uštede memorijskog prostora u računalu, vektori x i y mogu izračunavati i pohranjivati u jednom istom n-dimenzionalnom polju.

```
void LUP_SOLVE(double** L, double** U, int* P, double* b, const int n, double* x)
{
    double sum;

    for(int i = 0; i < n; i++)
    {
        sum = 0.0;
        for(int j = 0; j < i-1; j++)
            sum = sum + L[i][j] * x[j];

        x[i] = b[P[i]] - sum;
    }

    for(int i = n - 1; i >= 0; i--)
    {
        sum = 0.0;
        for(int j = i + 1; j < n; j++)
            sum = sum + U[i][j] * x[j];

        x[i] = (x[i] - sum) / U[i][i];
    }
}
```

Sada je, promatranjem ovog koda, lako odrediti složenost ovog algoritma kao broja potrebnih aritmetičkih operacija.
Očito je složenost ovog algoritma

$$T(n) \in O(n^2).$$

Napomena. U praksi se često rješavaju linearni sustavi s istom matricom A za različite vektore b. Iz ovoga zaključujemo da će nas, ako jednom izračunamo LUP dekompoziciju matrice A, promjena desne strane koštati samo $O(n^2)$ operacija. S druge strane, za svaku promjenu vektora b, Gaussove eliminacije moraju se provoditi u cijelosti, a one koštaju $O(n^3)$ operacija.

Računanje LU dekompozicije

Kako bismo bolje razumjeli metodu pronalaska LUP dekompozicije matrice A, najprije čemo opisati metodu nalaženja **LU dekompozicije**, tj. pretpostavit ćemo da matrice P nema, odnosno, preciznije, ona postoji, ali je jednaka n-dimenzionalnoj jediničnoj matrici.

Također, ovo će nam pomoći da shvatimo zašto je važno uvesti matricu permutacije, kao i da uvidimo vezu između LUP dekompozicije matrice i prije spomenute metode Gaussovih elminacija za rješavanje sustava linearnih jednadžbi.

Krećemo od n-dimenzionalne regularne matrice A.

Ukoliko je $n = 1$, matrica A je zapravo samo jedan element, tj. ona je oblika $[a_{11}]$, a može se zapisati kao $[\,1\,][a_{11}]$. Matrica $[\,1\,]$ je trivijalno donjetrokutasta s jedinicama na dijagonali, a matrica $[a_{11}]$ je trvijalno gornjetrokutasta, pa je ovo očito LUP dekompozicija matrice A.

Ukoliko je $n > 1$, matricu A dijelimo na 4 dijela, i označavamo ih, kako slijedi:

$$A = \left[\begin{array}{c|ccc} a_{11} & a_{12} & \cdots & a_{1n} \\ \hline a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{array}\right] = \begin{bmatrix} a_{11} & w^T \\ v & A' \end{bmatrix}.$$

Direktnim množenjem matrica, može se dokazati da vrijedi sljedeća jednakost:

$$A = \begin{bmatrix} a_{11} & w^T \\ v & A' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{bmatrix} \begin{bmatrix} a_{11} & w^T \\ 0 & A' - vw^T/a_{11} \end{bmatrix}.$$

Blok

$$A' - vw^T/a_{11}$$

očito je ($(n-1)$-dimenzionalna)) kvadratna matrica. Ona se u literaturi naziva **Schurov komplment**, ovdje s obzirom na a_{11}.

Budući da je matrica A regularna, Schurov koplement je sigurno također regularna matrica, pa ima svoju LU dekompoziciju.

Stoga, pretpostavimo da vrijedi:

$$A' - vw^T/a_{11} = L'U',$$

gdje je L' donjetrokutasta matrica s jedinicama na dijagonali, a U' gornjetrokutasta matrica.

Sada se, koristeći ovu faktorizaciju, kao i algebarske operacije definirane na matricama, mogu napisati sljedeće jednakosti:

$$\begin{aligned} A &= \begin{bmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{bmatrix} \begin{bmatrix} a_{11} & w^T \\ 0 & A' - vw^T/a_{11} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ v/a_1 1 & I_{n-1} \end{bmatrix} \begin{bmatrix} a_{11} & w^T \\ 0 & L'U' \end{bmatrix} \\ &= \begin{bmatrix} a_{11} & w^T \\ v & vw^T/a_{11} + L'U' \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ v/a_{11} & L' \end{bmatrix} \begin{bmatrix} a_{11} & w^T \\ 0 & U' \end{bmatrix}. \end{aligned}$$

Matrica

$$L = \begin{bmatrix} 1 & 0 \\ v/a_{11} & L' \end{bmatrix}$$

očito je donjetrokutasta matrica s jedinicama na dijagonali, budući da je L' takva matrica.

Nadalje, matrica

$$U = \begin{bmatrix} a_{11} & w^T \\ 0 & U' \end{bmatrix}$$

očito je gornjetrokutasta matrica, budući da je U' takva matrica.

Dobili smo da vrijedi

$$A = LU,$$

pa je to očito LUP dekompozicija matrice A.

Iz ovoga vidimo da je za nalaženje LU dekompozicije n-dimenzionalne regularne matrice A, tj. nalaženje odgovarajućih matrica L i U, potrebno izračunati:

- $(n-1)$-dimenzionalni vektor v/a_{11},
- $(n-1)$-dimenzionalnu kvadratnu matricu $A' - vw^T/a_{11}$,
- i izračunati LU dekompoziciju te $(n-1)$-dimenzionalne kvadratne matrice.

Elemente a_{11}, v, w^T i A' imamo iz same matrice A.

Dakle, očito za riješiti zadaću nalaženja LU dekompozicije matrice veličine n, gdje je n dimnenzija kvadratne matrice, treba obaviti neke operacije i riješiti istu takvu zadaću, ali veličine $n-1$, dakle ovaj prikazani **algoritam je rekurzivan**.

Složenost algoritma

Označimo vremensku složenost gornjeg algoritma za nalaženje LU dekompozicije n-dimenzionalne regularne matrice s $T(n)$, dakle veličina zadaće je dimenzija matrice. Složenost ćemo mjeriti kao broj potrebnih aritemtičkih operacija.

Kako za rješavanje zadaće treba izračunati $(n-1)$-dimenzionalni vektor v/a_{11}, gdje su $(n-1)$-dimenzionalni vektor v i broj a_{11} poznati, znači da treba obaviti $n-1$ operacija dijeljenja brojeva.

Za računanje $(n-1)$-dimenzionalne kvadratne matrice $A' - vw^T/a_{11}$, treba najprije izračunati produkt $(n-1)$-dimenzionalnih vektora v/a_{11} i w^T, tj. produkt $(n-1)$-dimenzionalne jednostupčane matrice v/a_{11} i $(n-1)$-dimenzionalne jednoretčane matrice w^T, znači da treba napraviti i $(n-1)^2$ operacija množenja brojeva. Zatim, treba od $(n-1)$-dimenzionalne kvadratne matrice A' oduzeti $(n-1)$-dimenzionalnu kvadratnu matricu vw^T/a_{11}, dakle treba obaviti i $(n-1)^2$ operacija oduzimanja.

Konačno, treba obaviti nalaženje LU dekompozicije sada poznate $(n-1)$-dimenzionalne matrice $A' - vw^T/a_{11}$, a složenost te zadaće, dakle broj potrebnih računskih operacija, je $T(n-1)$.

Dakle, možemo zaključiti da je složenost $T(n)$ dana sljedećom **rekurzivnom jednadžbom**:

$$\begin{aligned} T(n) &= (n-1) + (n-1)^2 + (n-1)^2 + T(n-1) \\ &= T(n-1) + n - 1 + 2(n-1)^2 \\ &= T(n-1) + n - 1 + 2(n^2 - 2n + 1) \\ &= T(n-1) + n - 1 + 2n^2 - 4n + 2 \\ &= T(n-1) + 2n^2 - 3n + 1 \end{aligned}$$

Možemo iskoristiti i gore pokazanu činjenicu da za $n = 1$ nije potrebno raditi ništa kako bi se dobila LU dekompozicija matrice, pa iz toga dobivamo početni uvjet:

$$T(1) = 0.$$

Jednadžba

$$T(n) = T(n-1) + 2n^2 - 3n + 1,$$

koju možemo pisati i kao

$$t_n = t_{n-1} + 2n^2 - 3n + 1,$$

tj.

$$t_n - t_{n-1} = 2n^2 - 3n + 1$$

očito je **linearna nehomogena rekurzivna jednadžba prvog reda**, i to takva da je desna strana, tj. "nehomogeni dio" oblika

$$g(n) = b^n p_\alpha(n),$$

gdje je $b = 1$, $n = 1$ i $\alpha = 2$.

Sjetimo se s predavanja iz OAA da se takva jednadžba može prevesti u linearnu homogenu jednadžbu reda $k + \alpha + 1$, gdje je k red polazne jednadžbe.

Ta homogena jednadžba ima karakterističnu jednadžbu

$$(a_k x^k + a_{k-1}x^{k-1} + \cdots + a_0)(x-b)^{\alpha+1} = 0,$$

gdje je

$$a_k x^k + a_{k-1}x^{k-1} + \cdots + a_0 = 0$$

karakteristična jednadžba polazne rekurzije.

Karakteristična jednadžba naše polazne rekurzije je:

$$x - 1 = 0,$$

pa zbog $b = 1$ i $\alpha = 2$ dobivamo karakterističnu jednadžbu:

$$(x-1)(x-1)^3 = 0,$$

tj.

$$(x-1)^4 = 0.$$

Dakle, opće rješenje ove jednadžbe je oblika

$$t_n = dn^3 + cn^2 + bn + a,$$

ili

$$T(n) = dn^3 + cn^2 + bn + a.$$

Sada možemo pisati:

$$\begin{aligned} T(n-1) &= d(n-1)^3 + c(n-1)^2 + b(n-1) + a \\ &= d(n^3 - 3n^2 + 3n - 1) + c(n^2 - 2n + 1) + bn - b + a \\ &= dn^3 - 3dn^2 + 3dn - d + cn^2 - 2cn + c + bn - b + a \\ &= dn^3 + (c - 3d)n^2 + (3d - 2c + b)n + c - d - b + a. \end{aligned}$$

Iz zadane rekurzivne jednadžbe dobivamo:

$$dn^3 + cn^2 + bn + a = dn^3 + (c-3d)n^2 + (3d - 2c + b)n + c - d - b + a + 2n^2 - 3n + 1,$$

što je ekvivalentno s:

$$\begin{aligned} &d = d \\ &c = c - 3d + 2 \quad \Leftrightarrow \quad 3d = 2 \quad \Leftrightarrow \quad d = \frac{2}{3} \\ &b = 3d - 2c + b - 3 \quad \Leftrightarrow \quad 2c = 3d - 3 \quad \Rightarrow \quad 2c = 3 \cdot \frac{2}{3} - 3 \quad \Leftrightarrow \quad c = -\frac{1}{2} \\ &a = c - d - b + a + 1 \quad \Leftrightarrow \quad b = c - d + 1 \quad \Rightarrow \quad b = -\frac{1}{2} - \frac{2}{3} + 1 = -\frac{1}{6} \end{aligned}$$

Iz početnog uvjeta $T(1) = 0$, dobivamo:

$$\begin{aligned} d + c + b + a &= 0 \\ a &= -b - c - d \\ a &= -\frac{2}{3} + \frac{1}{2} + \frac{1}{6} \\ a &= 0. \end{aligned}$$

Dakle, dobili smo da je rješenje

$$\boxed{T(n) = \frac{2}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n},$$

pa za složenost gornjeg algoritma vrijedi

$$T(n) \in O(n^3).$$

Implementacija - rekurzivna funkcija

Gore opisani rekurzivni algoritam implementirala sam direktno, pomoću rekurzivne funkcije:

```
void LU_DECOMPOSITION_REK(double** A, double** L, double** U, const int n, const int k)
{
    if(k < n)
        U[k][k] = A[k][k];   ///pivotni element ide na dijagonalu matrice U

    for(int i = k + 1; i < n; i++)
    {
        L[i][k] = A[i][k] / A[k][k];  ///racunamo k-ti donji podstupac matrice L

        U[k][i] = A[k][i];   /// racunamo k-ti desni podredak matrice U
    }

    for(int i = k + 1; i < n; i++)  /// racunamo Schurov komplement s obz na a_kk
        for(int j = k + 1; j < n; j++)
        {
            A[i][j] = A[i][j] - L[i][k] * U[k][j];
        }

    if(k < n - 1)
        LU_DECOMPOSITION_REK(A, L, U, n, k + 1);
}
```

Poziv iz glavnog programa izgleda ovako ($k = 0$):

LU_DECOMPOSITION_REK(A, L, U, n, 0);

Napomena 1. Zbog specifičnog oblika matrica L i U, očito je jasno da njihove netrivijalne elemente u računalu možemo pohraniti u jednu punu matricu, umjesto u dvije trokutaste. Ovo doprinosi štednji memorijskog prostora, odnosno smanjenju prostorne složenosti.

Također, ako nije bilo jasno odmah iz opisa ovog rekurzivnog algoritma, iz gornjeg je programa očito da u k-tom koraku novogenerirani elementi matrice L, novogeneriran elementi matrice U i Schurov komplement za taj korak, čine **particiju** $(n-k)$-dimenzionalne matrice na kojoj se radi, kao i da se u narednom koraku, od početne matrice A koriste samo elementi Schurovog komplementa iz tog, sad prethodnog, koraka. Iz ovoga možemo zaključiti da značajne elemente matricea L i U možemo odmah prilikom generiranja pohraniti upravo na odgovarajuća mjesta u matirici A od koje smo krenuli.

Napomena 2. Ako nije bilo jasno odmah iz opisa ovog rekurzivnog algoritma, iz gornjeg je programa očito da donjetrokutasta matrica L koje se dobije po završetku elementa u donjem trokutu sadrži upravo odgovarajuće multiplikatore iz metode Gaussovih eliminacija. Nadalje, matrica U očito je jednaka gornjetrokutastoj matrici suatava ekvivalentom početnom koji se dobije po završetku izvođenja Gaussovih eliminacija.

Implementacija - tzv. tail - call optimizacija

```
void LU_DECOMPOSITION(double** A, double** L, double** U, const int n)
{
    for(int k = 0; k < n; k++)
    {
        U[k][k] = A[k][k];

        for(int i = k + 1; i < n; i++)
        {
            L[i][k] = A[i][k] / A[k][k];

```

```
            U[k][i] = A[k][i];
        }

        for(int i = k + 1; i < n; i++)
            for(int j = k + 1; j < n; j++)
            {
                A[i][j] = A[i][j] - L[i][k] * U[k][j];
            }
    }
}
```

Složenost - 2. način

U jednoj iteraciji gornje vanjske for-petlje, dakle za fiksan k, očito se izvrši:

- $n-k$ operacija dijeljenja,
- $(n-k)(n-k) = (n-k)^2$ operacija množenja,
- $(n-k)(n-k) = (n-k)^2$ operacija oduzimanja,

a to troje se radi za svaki k od $k = 1$ do $k = n-1$, pa za složenost ovog algoritma možemo zaključiti:

$$T(n) = \sum_{k=1}^{n-1} \left[(n-k) + 2(n-k)^2\right]$$

Ako zamijenimo

$$n - k \to k,$$

korištenjem poznatih formula za konačne sume, dobivamo:

$$\begin{aligned}
T(n) &= \sum_{k=1}^{n-1} \left(k + 2k^2\right) \\
&= \sum_{k=1}^{n-1} k + 2\sum_{k=1}^{n-1} k^2 \\
&= \frac{(n-1)(n-2)}{2} + 2 \cdot \frac{(n-1)n(2n-1)}{6} \\
&= \frac{3n(n-1) + 2(n-1)n(2n-1)}{6} \\
&= \frac{4n^3 - 3n^2 - n}{6} \\
&= \frac{2}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n.
\end{aligned}$$

Dakle,

$$\boxed{T(n) = \frac{2}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n},$$

kao i maloprije.

Računanje LUP dekompozicije

Kao što smo vidjeli maloprije, za nalaženje LU dekompozicije kvadratne matrice potrebno je provoditi dijeljenje gornjim lijevim elementom matrice. Međutim, zbog aritmetike računala i grešaka zaokruživanja pri spremanju brojeva, ukoliko je element kojim se dijeli jednak 0, ili jako mali po apsolutnoj vrijednosti, može doći do katastrofalnog kraćenja i algoritam je numerički nestabilan.

Zbog toga, ideja je modificirati prethodni algoritam, tako da se prije LU faktorizacije regularne kvadratne matrice na gornje lijevo mjesto dovede element te matrice koji je najveći po apsolutnoj vrijednoti u tom stupcu. Budući da je matrica regularna, taj će element sigurno biti različit od 0, a i bit će najveći mogući po apsolutnoj vrijednosti u stupcu.

Ovo se postiže tako da se zamijene odgovarajući retci matrice, a znamo da takva transformacija ostavlja sustav ekvivalentim početnom.

Sjetimo se još da je zamjena redaka matrice ekvivalentna množenju matrice s lijeva odgovarajućom matricom permutacije.

Opišimo sada modificirani algoritam. Njime ćemo dobiti LU faktorizaciju permutirane matrice A, dakle matrice PA, gdje je P matrica permutacije. dakle, dobit ćemo LUP dekompoziciju matrice A:

$$PA = LU.$$

Najprije pronalazimo najveći eleement po apsolutnoj vrijednosti u prvom stupcu, te regularnu kvadratnu matricu A pomnožimo s lijeva odgovarajućom matricom permutacije, označimo je s Q. Sada postupamo kao i ranije:

$$QA = \left[\begin{array}{c|ccc} a_{k1} & a_{k2} & \cdots & a_{kn} \\ \hline a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{array}\right] = \begin{bmatrix} a_{k1} & w^T \\ v & A' \end{bmatrix},$$

i vrijedi

$$QA = \begin{bmatrix} a_{k1} & w^T \\ v & A' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ v/a_{k1} & I_{n-1} \end{bmatrix} \begin{bmatrix} a_{11} & w^T \\ 0 & A' - vw^T/a_{k1} \end{bmatrix}.$$

Budući da je matrica A regularna, Schurov komplement je sigurno također regularna matrica, pa ima svoju LUP dekompoziciju.

Stoga, pretpostavimo da vrijedi:

$$P'(A' - vw^T/a_{11}) = L'U',$$

gdje je L' donjetrokutasta matrica s jedinicama na dijagonali, U' gornjetrokutasta matrica, aP' matrica permutacije.

Definirajmo još i:

$$P = \begin{bmatrix} 1 & 0 \\ 0 & P' \end{bmatrix} Q.$$

Sada se, koristeći ovu faktorizacije, kao i algebarske operacije definirane na matricama, mogu napisati sljedeće jednakosti:

$$
\begin{aligned}
PA &= \begin{bmatrix} 1 & 0 \\ 0 & P' \end{bmatrix} QA \\
&= \begin{bmatrix} 1 & 0 \\ 0 & P' \end{bmatrix} \begin{bmatrix} 1 & 0 \\ v/a_{k1} & I_{n-1} \end{bmatrix} \begin{bmatrix} a_{k1} & w^T \\ 0 & A' - vw^T/a_{k1} \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ P'v/a_{k1} & P' \end{bmatrix} \begin{bmatrix} a_{k1} & w^T \\ 0 & A' - vw^T/a_{k1} \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ P'v/a_{k1} & I_{n-1} \end{bmatrix} \begin{bmatrix} a_{k1} & w^T \\ 0 & P'(A' - vw^T/a_{k1}) \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ P'v/a_{k1} & P' \end{bmatrix} \begin{bmatrix} a_{k1} & w^T \\ 0 & L'U' \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ P'v/a_{k1} & L' \end{bmatrix} \begin{bmatrix} a_{k1} & w^T \\ 0 & U' \end{bmatrix}.
\end{aligned}
$$

Dakle, vidimo da je za

$$
P = \begin{bmatrix} 1 & 0 \\ 0 & P' \end{bmatrix} Q, \qquad L = \begin{bmatrix} 1 & 0 \\ v/a_{k1} & L' \end{bmatrix}, \qquad U = \begin{bmatrix} a_{k1} & w^T \\ 0 & U' \end{bmatrix}
$$

dekompozicija

$$
PA = LU
$$

LUP dekompozicija matrice A. Iz ovoga vidimo da je za nalaženje LUP dekompozicije n-dimenzionalne regularne matrice A, tj. nalaženje odgovarajućih matrica L i U, potrebno izračunati:

- $(n-1)$-dimenzionalni vektor v/a_{11},
- $(n-1)$-dimenzionalnu kvadratnu matricu $A' - vw^T/a_{11}$,
- izračunati LUP dekompoziciju te $(n-1)$-dimenzionalne kvadratne matrice
- pomnožiti tu matricu slijeva matricom permutacije P'
- pomnožiti slijeva vektor v/a_{11} matricom permutacije P'

Budući da je množenje matricom permuatcije slijeva zapravo zamjena redaka matrice, u računalu se ne implementira kao množenje matrica, već kao zamjene odgovarajućih elemenata dvaju redaka matrice. Stoga, možemo zaključiti da ova modifikacija algoritma za nalaženje LU dekompozicije matrice ne utječe bitno na složenost algoritma, te da će složenost algoritma za nalaženje LUP dekompozicije n-dimenzionalne kvadratne matrice A također biti

$$
T(n) \in (n^3).
$$

Implementacija - rekurzivno

```
void LUP_DECOMPOSITION_REK(double** A, double** L, double** U, int* P, const int n, cons
{
    if(k < n - 1)
    {
        double max = 0.0;
        int max_ind;
        for(int i = k; i < n; i++)
        {
```

```
            if(abs(A[i][k]) > max)
            {
                max = abs(A[i][k]);
                max_ind = i;
            }
        }

        if(max == 0)
        {
            cout << "Matrica sustava je singularna!" << endl;
            exit(-1);
        }

        if(max_ind != k)
        {
            int temp = P[k];
            P[k] = P[max_ind];
            P[max_ind] = temp;

            double pom;
            for(int j = k; j < n; j++)
            {
                pom = A[k][j];
                A[k][j] = A[max_ind][j];
                A[max_ind][j] = pom;
            }

            for(int j = 0; j < k; j++)
            {
                pom = L[k][j];
                L[k][j] = L[max_ind][j];
                L[max_ind][j] = pom;
            }
        }
    }

    if(k < n)
        U[k][k] = A[k][k];

    for(int i = k + 1; i < n; i++)
    {
        L[i][k] = A[i][k] / A[k][k];

        U[k][i] = A[k][i];
    }

    for(int i = k + 1; i < n; i++)
        for(int j = k + 1; j < n; j++)
        {
            A[i][j] = A[i][j] - L[i][k] * U[k][j];
        }

    if(k < n - 1)
        LUP_DECOMPOSITION_REK(A, L, U, P, n, k + 1);
}
```

Implementacija - tail-call optimizacija

```
void LUP_DECOMPOSITION(double** A, double** L, double** U, int* P, const int n)
{
    for(int k = 0; k < n; k++)
    {
        double max = 0;
```

```
        int max_ind;
        for(int i = k; i < n; i++)
        {
            if(abs(A[i][k]) > max)
            {
                max = abs(A[i][k]);
                max_ind = i;
            }
        }

        if(max == 0)
        {
            cout << "Matrica sustava je singularna!" << endl;
            exit(-1);
        }

        if(max_ind != k)
        {
            int temp = P[k];
            P[k] = P[max_ind];
            P[max_ind] = temp;

            double pom;
            for(int j = k; j < n; j++)
            {
                pom = A[k][j];
                A[k][j] = A[max_ind][j];
                A[max_ind][j] = pom;
            }

            for(int j = 0; j < k; j++)
            {
                pom = L[k][j];
                L[k][j] = L[max_ind][j];
                L[max_ind][j] = pom;
            }
        }

        U[k][k] = A[k][k];

        for(int i = k + 1; i < n; i++)
        {
            L[i][k] = A[i][k] / A[k][k];

            U[k][i] = A[k][i];
        }

        for(int i = k + 1; i < n; i++)
            for(int j = k + 1; j < n; j++)
            {
                A[i][j] = A[i][j] - L[i][k] * U[k][j];
            }
    }
}
```