Calcul Ştiinţific

Capitolul 3: Matrice rare

Bogdan Dumitrescu

Facultatea de Automatică şi Calculatoare

Universitatea Politehnica Bucureşti

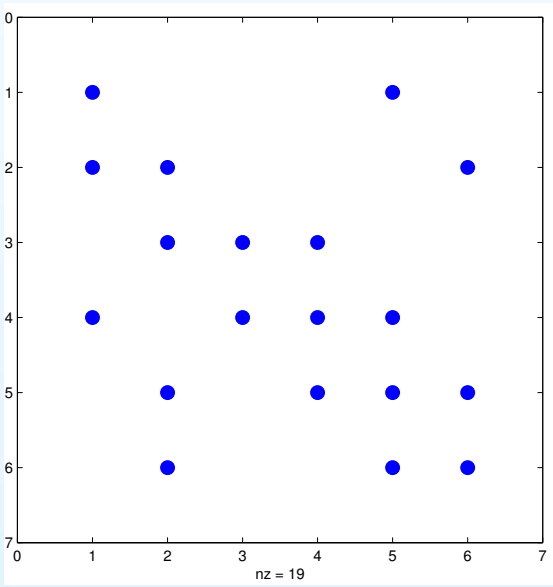
Cuprins

- Noţiuni de bază despre matrice rare
  - Formate de memorare
  - Algoritmi de înmulţire matrice-vector
- Permutări pentru produsul MV: algoritmul Cuthill-McKee
- Rezolvarea sistemelor triunghiulare
- Factorizarea Cholesky
  - umplere, arbore de eliminare, factorizare simbolică
  - factorizare numerică
  - permutare, algoritmi: grad minim, nested dissection
- Algoritmi iterativi pentru rezolvarea sistemelor liniare

Matrice rare

- O matrice rară are majoritatea elementelor egale cu zero
- Lucrăm doar cu matrice pătrate $n \times n$
- Notăm NZ numărul de elemente nenule
- In general: $NZ = O(n)$
- Exemplu (didactic): matrice de dimensiune 6×6

$$A = \begin{bmatrix} 11 & & & & 15 & \\ 21 & 22 & & & & 26 \\ & 32 & 33 & 34 & & \\ 41 & & 43 & 44 & 45 & \\ & 52 & & 54 & 55 & 56 \\ & 62 & & & 65 & 66 \end{bmatrix}$$



Memorarea matricelor rare

- În practică dimensiunile pot fi mult mai mari
- Memorarea a n^2 elemente (ca pentru matrice pline), ar deveni prohibitivă
- De aceea se memorează
  - valorile elementelor nenule
  - poziţiile lor
- Formate de memorare
  - Triplet ("trivial")
  - Comprimat pe linii (compressed row storage)
  - Comprimat pe coloane (compressed column)
  - Diagonale "zimţate" (jagged diagonal)
  - . . . şi altele

Memorare şi algoritmi

- Modul de memorare are efect asupra formei algoritmilor şi deci asupra vitezei de execuţie pe diverse arhitecturi de calcul
- Vom ilustra fiecare format cu algoritmul de înmulţire matrice-vector (util în rezolvarea iterativă a sistemelor liniare)
- Algoritmul standard pentru calculul $y \leftarrow y + Ax$, cu $A \in \mathbb{R}^{n \times n}$, $x, y \in \mathbb{R}^n$

$$
\begin{array}{l}
\texttt{Pentru } i = 1 : n \\
\quad \texttt{Pentru } j = 1 : n \\
\quad\quad y_i \leftarrow y_i + a_{ij} x_j
\end{array}
$$

- Pentru matrice plină sunt $2n^2$ operaţii şi n^2 accese la memorie pentru elementele lui A (o dată la fiecare element)

Formatul triplet

- Pentru fiecare element nenul a_{ij} se memorează poziţia (i, j)
- Se utilizează doi vectori, II şi JJ, de lungime NZ fiecare
- Memorie necesară: NZ reali, $2NZ$ întregi

$$
\begin{array}{lcccccccccccccccccc}
nzval & = & 11 & 15 & 21 & 22 & 26 & 32 & 33 & 34 & 41 & 43 & 44 & 45 & 52 & 54 & 55 & 56 & 62 & 65 & 66 \\
II & = & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 & 4 & 4 & 4 & 4 & 5 & 5 & 5 & 5 & 6 & 6 & 6 \\
JJ & = & 1 & 5 & 1 & 2 & 6 & 2 & 3 & 4 & 1 & 3 & 4 & 5 & 2 & 4 & 5 & 6 & 2 & 5 & 6
\end{array}
$$

- Elementele nenule pot fi aranjate în orice ordine
- Acces dificil la un element oarecare
- Informaţie redundantă

Formatul triplet — produsul MV

- Înmulţire matrice-vector:

$$\texttt{Pentru } k = 1 : NZ$$
$$y(II(k)) \leftarrow y(II(k)) + nzval(k) \cdot x(JJ(k))$$

- $2NZ$ operaţii, $2NZ$ accese la memorie, exact cât e nevoie
- Algoritmul nu depinde de ordinea de memorare în $nzval$
- Accesele la x şi y nu sunt neapărat localizate

Formatul comprimat pe linii (CL)

- Elementele nenule se memorează în ordinea liniilor
- Indicii de coloane se memorează integral
- Indicii primelor elemente ale unei linii se memorează separat

$$
\begin{aligned}
nzval &= 11\ 15\ 21\ 22\ 26\ 32\ 33\ 34\ 41\ 43\ 44\ 45\ 52\ 54\ 55\ 56\ 62\ 65\ 66 \\
colind &= 1\ 5\ 1\ 2\ 6\ 2\ 3\ 4\ 1\ 3\ 4\ 5\ 2\ 4\ 5\ 6\ 2\ 5\ 6 \\
rowptr &= 1\ 3\ 6\ 9\ 13\ 17\ 20
\end{aligned}
$$

- Dacă $nzval(k) = a_{ij}$, atunci $rowptr(i) \leq k < rowptr(i+1)$
- Convenţional, se pune $rowptr(n+1) = NZ + 1$
- Memorie necesară: NZ reali, $NZ + n + 1$ întregi

Formatul CL — produsul MV

- Înmulţire matrice-vector:

$$\begin{array}{l}\texttt{Pentru } i = 1 : n \\ \quad \texttt{Pentru } k = rowptr(i) : rowptr(i+1) - 1 \\ \quad\quad y(i) \leftarrow y(i) + nzval(k) \cdot x(colind(k))\end{array}$$

- $2NZ$ operaţii, $2NZ$ accese la memorie
- Accesele la x nu sunt localizate
- Vectorii care apar in produsul scalar din a doua buclă sunt de regulă scurţi
- Exerciţiu: algoritm pentru $y \leftarrow y + A^T x$!

Formatul comprimat pe coloane (CC)

- Similar cu CL, dar pe coloane
- Numit şi formatul Harwell-Boeing
- Identic cu CL pentru A^T

$$
\begin{aligned}
nzval &= 11\ 21\ 41\ 22\ 32\ 52\ 62\ 33\ 43\ 34\ 44\ 54\ 15\ 45\ 55\ 65\ 26\ 56\ 66 \\
rowind &= 1\ 2\ 4\ 2\ 3\ 5\ 6\ 3\ 4\ 3\ 4\ 5\ 1\ 4\ 5\ 6\ 2\ 5\ 6 \\
colptr &= 1\ 4\ 8\ 10\ 13\ 17\ 20
\end{aligned}
$$

- Dacă $nzval(k) = a_{ij}$, atunci $colptr(j) \leq k < colptr(j+1)$
- Memorie necesară: NZ reali, $NZ + n + 1$ întregi

Formatul CC — produsul MV

- Înmulţire matrice-vector:

$$\begin{array}{l} \texttt{Pentru } j = 1 : n \\ \quad \texttt{Pentru } k = colptr(j) : colptr(j+1) - 1 \\ \quad\quad y(rowind(k)) \leftarrow y(rowind(k)) + nzval(k) \cdot x(j) \end{array}$$

- $2NZ$ operaţii, $2NZ$ accese la memorie
- Accesele la y nu sunt localizate
- Vectorii care apar in SAXPY din a doua buclă sunt de regulă scurţi

Formatul diagonale zimţate (DZ-ITPACK)

- Forma simplificată: ITPACK
- Idee: memorare într-o matrice plină $n \times L$, unde L este numărul cel mai mare de elemente nenule pe o linie

$$nzval = \begin{bmatrix} 11 & 15 & 0 & 0 \\ 21 & 22 & 26 & 0 \\ 32 & 33 & 34 & 0 \\ 41 & 43 & 44 & 45 \\ 42 & 54 & 55 & 56 \\ 62 & 65 & 66 & 0 \end{bmatrix} \qquad colind = \begin{bmatrix} 1 & 5 & 0 & 0 \\ 1 & 2 & 6 & 0 \\ 2 & 3 & 4 & 0 \\ 1 & 3 & 4 & 5 \\ 2 & 4 & 5 & 6 \\ 2 & 5 & 6 & 0 \end{bmatrix}$$

- Prima coloană din $nzval$ e o "diagonală zimţată", etc.
- Memorie necesară: $nL \geq NZ$ reali, nL întregi
- Eficient când numărul de elemente nenule de pe fiecare linie este cam acelaşi

Formatul DZ-ITPACK — produsul MV

- Înmulţire matrice-vector:

$$\begin{array}{l} \texttt{Pentru } i = 1 : n \\ \quad \texttt{Pentru } j = 1 : L \\ \quad\quad y(i) \leftarrow y(i) + nzval(i,j) \cdot x(colind(i,j)) \end{array}$$

- $2nL \geq 2NZ$ operaţii (apar înmulţiri cu 0)
- Accesele la x nu sunt localizate
- Vectori lungi (N) !
- Eficient pe calculatoare vectoriale

Formatul DZ

- Rafinare a formatului ITPACK
- Liniile matricii sunt permutate în ordinea descrescătoare a numărului de elemente nenule (permutarea trebuie memorată într-un vector suplimentar)
- Matricea redusă memorată într-un vector, ca în formatul CC
- La fel indicii de coloane
- Format avantajos tot pe calculatoare vectoriale

Creşterea vitezei produsului MV

- Am văzut că formatul dictează algoritmul MV şi deci timpul de execuţie
- Alte idei de creştere a vitezei:
  - Format pe blocuri—chiar dacă matricele sunt rare, ele pot conţine blocuri aproape pline
  - Matricea poate fi permutată (explicit sau nu) înainte de înmulţire; se calculează $P^T y \leftarrow P^T y + (P^T A P)(P^T x)$, unde P e o matrice de permutare. Calculul permutării depinde de scopul propus.

Format comprimat pe blocuri

- Presupunem că matricea A este partiţionată în blocuri $n_b \times n_b$

$$A = \left[\begin{array}{cc|cc|cc} 11 & & & & 15 & 16 \\ 21 & 22 & & & & 26 \\ \hline & 32 & 33 & 34 & & \\ 41 & 42 & 43 & 44 & & \\ \hline & & 51 & 52 & 55 & 56 \\ & & & 62 & 65 & 66 \end{array}\right]$$

- Se utilizează unul din formatele prezentate anterior, memorând însă blocuri în loc de elemente
- Blocurile se memorează ca fiind pline, deci inclusiv zerourile
- Format avantajos pe calculatoare cu memorie ierarhică

Format CL pe blocuri

- Format comprimat pe linii (un bloc e memorat pe coloane)

$$
\begin{aligned}
nzval &= 11\ 21\ 0\ 22\ 15\ 0\ 16\ 26\ \ldots\ 55\ 65\ 56\ 66 \\
colind &= 1\ \ 3\ \ 1\ \ 2\ \ \ 2\ \ 3 \\
rowptr &= 1\ \ 3\ \ 5\ \ 7
\end{aligned}
$$

- NZB – numărul de blocuri nenule (6 mai sus)
- Memorie necesară:
  - $NZB \cdot n_b^2 \geq NZ$ reali
  - $NZB + n/n_b + 1$ întregi
- Se memorează mai puţini întregi decât în formatul CL scalar
- Exerciţiu: produsul MV !

Permutări şi produsul MV

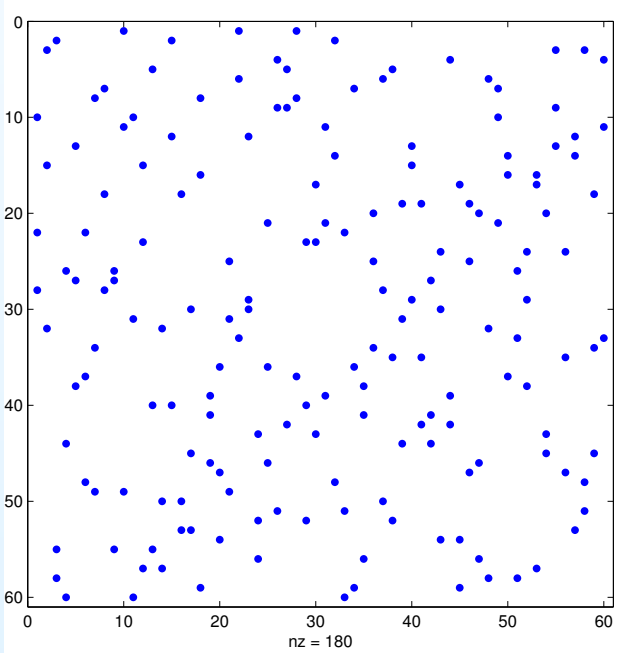
- Matrice de permutare: matricea unitate cu liniile (coloanele) permutate
- $P^T y \leftarrow P^T y + (P^T A P)(P^T x)$: se schimbă doar numerotarea variabilelor
- Scop posibil: creşterea localităţii accesului la x (în formatul CL) sau la y (CC)
- Produsului MV pentru CL

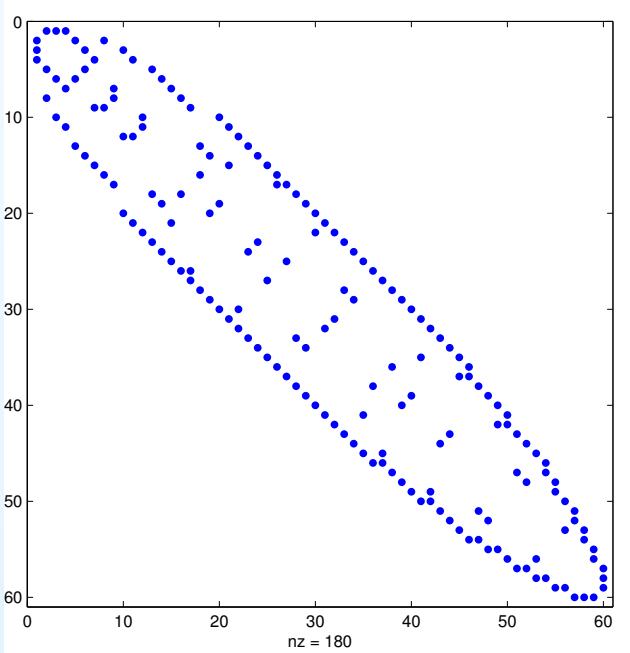
$$\texttt{Pentru } k = rowptr(i) : rowptr(i+1) - 1$$
$$y(i) \leftarrow y(i) + nzval(k) \cdot x(colind(k))$$

- Ideal ar fi ca indicii lui x să fie apropiaţi pentru valori k apropiate, astfel încât valorile x respective sa rămână in memoria rapidă
- Situaţia ideală: matrice bandă—indicii de coloană sunt grupaţi în jurul valorii corespunzătoare diagonalei.

Permutarea ”reverse Cuthill-McKee”

- Se poate deci căuta o permutare P astfel încât $P^T AP$ să aibă elementele cât mai aproape de diagonală
- Problema e dificilă, se utilizează metode euristice
- Reverse Cuthill-McKee: algoritm de reducere a benzii





Graf de conectivitate

- (Ne gândim de acum înainte doar la matrice simetrice)
- Asociem matricei A un graf de conectivitate
- Nodurile sunt numerotate de la 1 la n
- Arcul (i, j) există dacă $a_{ij} \neq 0$
- Gradul unui nod este numărul de arce adiacente: numărul de elemente nenule de pe linia (coloana) respectivă a matricei
- Matricea $P^T AP$ are un graf cu structură identică, dar cu altă numerotare a nodurilor
- Exerciţiu: matrice rară 4×4, graful asociat, efectul unei permutări elementare (care schimbă între ele două linii/coloane)

Algoritmul Cuthill-McKee

- Se porneşte de la graful de conectivitate al unei matrice
- Scop: reordonarea nodurilor într-o listă $\mathcal{L}$ astfel încât matricea asociată are lărgime de bandă $\max_{a_{ij} \neq 0} |i - j|$ cât mai mică
- Primul în $\mathcal{L}$ e nodul cu grad minim (între mai multe noduri cu acelaşi grad, ordinea e aleatoare)
- Următoarele poziţii în $\mathcal{L}$ sunt ocupate de vecinii primului nod, în ordine crescătoare a gradului
- Se continuă punând în $\mathcal{L}$ vecinii celui de-al doilea nod (dacă nu sunt deja în $\mathcal{L}$), etc.
- Ordonarea corespunde unei traversări în lăţime a unui arbore de acoperire al grafului
- "Reverse": în final $\mathcal{L}$ se ordonează invers

Cuthill-McKee: exemplu

- Pentru matricea din stânga, rezultatul este: 5, 3, 2, 1, 4, 6
- Matricea reordonată este cea din dreapta

$$
\begin{array}{c}
1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6
\end{array}
\begin{bmatrix}
\times & \times & & \times & & \times \\
\times & \times & \times & & & \\
 & \times & \times & & \times & \\
\times & & & \times & & \times \\
 & & \times & & \times & \\
\times & & & \times & & \times
\end{bmatrix}
\longrightarrow
\begin{array}{c}
5 \\ 3 \\ 2 \\ 1 \\ 4 \\ 6
\end{array}
\begin{bmatrix}
\times & \times & & & & \\
\times & \times & \times & & & \\
 & \times & \times & \times & & \\
 & & \times & \times & \times & \times \\
 & & & \times & \times & \times \\
 & & & \times & \times & \times
\end{bmatrix}
$$

De ce ”reverse” ?

- Inversarea ordinii nu modifică lăţimea de bandă
- În schimb micşorează (nu creşte, de fapt) profilul
- Notăm f_i indicele celui mai mic j pentru care $a_{ij} \neq 0$
- Notăm $d_i = i - f_i$ (cât de departe este cel mai depărtat element de diagonală, pe linia i)
- Profilul este $\sum_{i=1}^{n} d_i$
- Un profil mai mic înseamnă o mai bună localitate a acceselor la x în produsul MV, format CL

Algoritmul GPS

- Alţi algoritmi de reducere a lărgimii benzii utilizează euristici sau optimizare
- Un exemplu: algoritmul Gibbs-Poole-Stockmeyer (GPS)
- Se găseşte întâi un pseudo-diametru, adică două noduri aflate la distanţă aproape maximă unul de altul
- (Distanţa dintre două noduri este lungimea drumului cel mai scurt între ele)
- Se construieşte lista $\mathcal{L}$ ca în Cuthill-McKee, dar pornind de la ambele capete
- Rezultatele nu sunt mult mai bune, dar algoritmul e mai rapid

Rezolvarea sistemelor triunghiulare

- Fie A o matrice inferior triunghiulară: $a_{ij} = 0, \forall i < j$
- Algoritm pentru rezolvarea $Ax = b$, pe linii

$$
\begin{aligned}
&x \leftarrow b \\
&\texttt{Pentru } i = 1 : n \\
&\quad \texttt{Pentru } j = 1 : i - 1 \\
&\quad\quad x_i \leftarrow x_i - a_{ij} x_j \\
&\quad x_i \leftarrow x_i / a_{ii}
\end{aligned}
$$

- Algoritm pentru rezolvarea $Ax = b$, pe coloane

$$
\begin{aligned}
&x \leftarrow b \\
&\texttt{Pentru } j = 1 : n \\
&\quad x_j \leftarrow x_j / a_{jj} \\
&\quad \texttt{Pentru } i = j + 1 : n \\
&\quad\quad x_i \leftarrow x_i - a_{ij} x_j
\end{aligned}
$$

- Fiecare element al matricei A e folosit exact o dată

Rezolvare sisteme triunghiulare în format CL

- Matricea A este rară, memorată în format CL
- Presupunem că toate elementele diagonale sunt nenule
- Matricea fiind memorată pe linii, folosim algoritmul pe linii
- Dacă $a_{ij} = 0$, nu se efectuează nici o operaţie, deci e suficient să parcurgem elementele nenule
- Număr de operaţii: aprox. $2NZ$

$$
\begin{aligned}
&x \leftarrow b \\
&\texttt{Pentru } i = 1 : n \\
&\qquad \texttt{Pentru } k = rowptr(i) : rowptr(i+1) - 2 \\
&\qquad\qquad x(i) \leftarrow x(i) - nzval(k) \cdot x(colind(k)) \\
&\qquad x(i) \leftarrow x(i) \;/\; nzval(rowptr(i+1) - 1)
\end{aligned}
$$

- (Elementul diagonal este ultimul pe linia lui !)

Rezolvare sisteme triunghiulare în format CC

- Matricea A este acum memorată în format CC
- Transformăm algoritmul pe coloane:

$$
\begin{array}{l}
x \leftarrow b \\
\texttt{Pentru } j = 1 : n \\
\quad x(j) \leftarrow x(j) \ / \ nzval(colptr(j)) \\
\quad \texttt{Pentru } k = colptr(j) + 1 : colptr(j+1) - 1 \\
\quad\quad x(rowind(k)) \leftarrow x(rowind(k)) - nzval(k) \cdot x(j)
\end{array}
$$

- (Elementul diagonal este primul pe coloana lui !)
- Exerciţiu: cum se transformă algoritmul în cazul unui format CC pe blocuri ?
- Exerciţiu: algoritm pentru rezolvarea unui sistem superior triunghiular (matrice memorată CC sau CL)

Factorizarea Cholesky

- Fie A o matrice simetrică şi pozitiv definită
- Factorizarea Cholesky: $A = LL^T$, cu L inferior triunghiulară
- La nivel de element (ne interesează doar triunghiul inferior):

$$a_{ij} = \sum_{k=1}^{j} \ell_{ik}\ell_{jk}, \quad i \geq j$$

- De aici rezultă că se poate calcula

$$\ell_{jj} = \left(a_{jj} - \sum_{k=1}^{j-1} \ell_{jk}^2\right)^{1/2},$$
$$\ell_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} \ell_{ik}\ell_{jk}\right)/\ell_{jj}, \quad i > j$$

- ℓ_{jj} depinde doar de elementele de pe linia j din L
- ℓ_{ij} depinde de elementele de pe liniile i şi j din L

Algoritm pe coloane

- Factorul L se poate calcula pe loc în A (din care e memorat doar triunghiul inferior)
- Algoritm de calcul

```
Pentru j = 1 : n
    Pentru k = 1 : j − 1                    % calcul element diagonal
        a_jj ← a_jj − a_jk^2
    a_jj ← sqrt(a_jj)
    Pentru i = j + 1 : n                    % restul elementelor de pe coloana j
        Pentru k = 1 : j − 1
            a_ij ← a_ij − a_ik a_jk
        a_ij ← a_ij / a_jj
```

- Algoritmul se poate scrie mai compact (dar cu acelaşi număr de operaţii), prin unificarea unor bucle

Algoritm compact

- Algoritmul compact:

$$
\begin{array}{l}
\texttt{Pentru } j = 1 : n \\
\quad \texttt{Pentru } k = 1 : j-1 \\
\quad\quad \texttt{Pentru } i = j : n \qquad \% \; cmod(j,k) \\
\quad\quad\quad a_{ij} \leftarrow a_{ij} - a_{ik} a_{jk} \\
\quad a_{jj} \leftarrow \sqrt{a_{jj}} \\
\quad \texttt{Pentru } i = j+1 : n \qquad \% \; cdiv(j) \\
\quad\quad a_{ij} \leftarrow a_{ij} / a_{jj}
\end{array}
$$

- Număr de operaţii: aprox. $n^3/3$

Cholesky pentru matrice rare

- Dacă A este rară, L rezultă de obicei rară
- Număr de operaţii mai mic, în acest caz
- Caz favorabil extrem:

$$A = \begin{bmatrix} \times & & & \times \\ & \times & & \times \\ & & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \Longrightarrow L = \begin{bmatrix} \times & & & \\ & \times & & \\ & & \times & \\ \times & \times & \times & \times \end{bmatrix}$$

- Caz nefavorabil extrem:

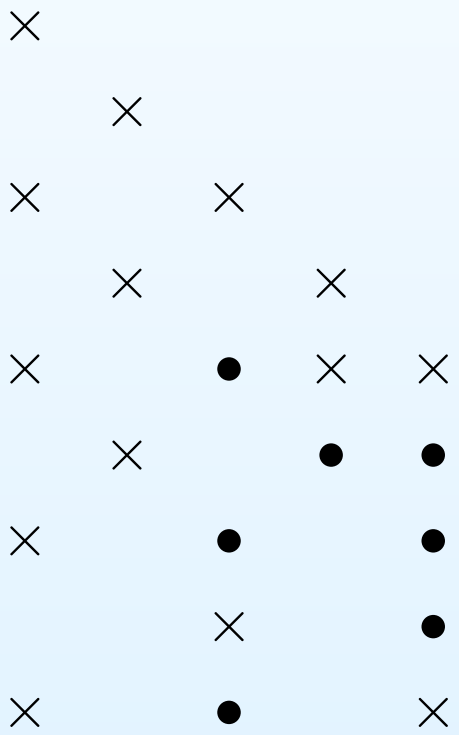
$$A = \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & & \\ \times & & \times & \\ \times & & & \times \end{bmatrix} \Longrightarrow L = \begin{bmatrix} \times & & & \\ \times & \times & & \\ \times & \times & \times & \\ \times & \times & \times & \times \end{bmatrix}$$

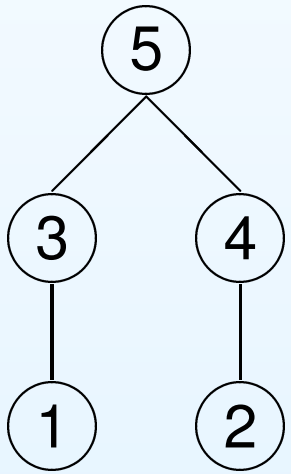
Ce structură are L ?

- L moşteneşte elementele nenule ale lui A
- În plus, în L apar elemente nenule noi (fill-in, umplere)
- Unele elemente ale lui L ar putea rezulta nule în urma calculelor, dar nu ne interesează decât zerourile structurale
- $j = 1$: prima coloană a lui L e identică cu cea a lui A
- $j = 2$: $\ell_{i2} = (a_{i2} - \ell_{i1}\ell_{21})/\ell_{22}$
- Rezultă că $\ell_{i2} \neq 0$ dacă $\ell_{21} \neq 0$ ŞI $\ell_{i1} \neq 0$
- Deci, dacă $\ell_{21} \neq 0$, coloana 2 se "umple" cu toate elementele coloanei 1 (pe lângă cele moştenite de la A)
- În general, două elemente nenule pe coloana k, anume ℓ_{ik}, ℓ_{jk} $(i > j)$ produc un element nenul nou ℓ_{ij}
- Umplerea poate fi caracterizată minimal de un arbore (numit arbore de eliminare)

Arbore de eliminare

- Notăm $p(j) = \min_{\ell_{ij} \neq 0,\ i>j} i$
- În arborele de eliminare, $p(j)$ este părintele nodului j
- Legendă: $\times$ – nenule în A, $\bullet$ – umplere





Comentarii

- Coloana $p(j)$ moşteneşte întreaga structură a coloanei j (vezi de exemplu în pagina anterioară $p(1) = 3$)
- Pentru a determina structura unei coloane, e suficient sa ne uităm la nodurile "fiu" ale unui nod (atenţie însă, structura lui L se calculează dinamic, arborele trebuie parcurs de la frunze spre rădăcină)
- În exemplul anterior:
  - coloana 1 transmite structura sa coloanei 3, deoarece $a_{31} \neq 0$
  - coloana 3 transmite structura sa coloanei 5, deoarece ℓ_{53} a devenit nenul
  - indirect, coloana 1 transmite structura sa coloanei 5, dar transferul se face prin intermediul coloanei 3, deci nu mai trebuie studiat explicit

Graful asociat lui L

- O coloană k din L transmite structura sa tuturor coloanelor i pentru care $\ell_{ik} \neq 0$
- În graful asociat lui L, aceşti indici i formează o clică (conexiune fiecare cu fiecare)
- Arborele de eliminare este un arbore de acoperire a acestui graf
- Exerciţiu: construiţi graful asociat matricei din exemplul anterior, completând până la 9 coloane (matricea are 9 linii)

Factorizare simbolică

- Factorizarea Cholesky se face în doi paşi
  - Factorizare simbolică: se determină structura lui L şi se rezervă memoria corespunzătoare
  - Factorizare numerică: se calculează efectiv L
- Notăm $cstruct(X, j)$ mulţimea poziţiilor elementelor nenule din coloana j a matricei X
- Algoritm de factorizare simbolică

  `Pentru` $j = 1 : n$
  $cstruct(L, j) = cstruct(A, j)$
  `Pentru` $j = 1 : n$
  calculează $p(j)$
  $cstruct(L, p(j)) = cstruct(L, p(j)) \cup cstruct(L, j)$

Factorizare simbolică pe coloane

- Algoritmul precedent are defectul că se operează de mai multe ori cu aceeaşi coloană
- În cazul unei memorări pe coloane, e de preferat să se determine necesarul de memorie în ordinea coloanelor
- Algoritmul devine

Pentru $j = 1 : n$
    $cstruct(L, j) = cstruct(A, j)$
    Pentru $k = 1 : j - 1$
        Dacă $p(k) = j$ atunci
            $cstruct(L, j) = cstruct(L, j) \cup cstruct(L, k)$
    calculează $p(j)$

- Pentru eficienţă, vectorul p se poate organiza sub forma unei liste, având în vedere că un element e folosit o singură dată

Factorizare numerică—algoritm fan-in

- Notăm $lstruct(L, i)$ mulţimea indicilor coloanelor cu elemente nenule pe linia i din L
- Presupunem că factorizarea simbolică a fost efectuată
- Algoritmul compact pe coloane devine

```
Pentru j = 1 : n
    Pentru k ∈ lstruct(L, j) \ {k}
        Pentru i ∈ cstruct(L, k), i ≥ j      % cmod(j, k)
            a_ij ← a_ij − a_ik a_jk
    a_jj ← √a_jj
    Pentru i ∈ cstruct(L, j) \ {j}           % cdiv(j)
        a_ij ← a_ij / a_jj
```

- Exerciţiu: scrieţi algoritmul cu L în format CC. Dificultăţi:
  - parcurgerea $lstruct(L, j)$
  - găsirea elementelor din $cstruct(L, k)$ în $cstruct(L, j)$

Comentarii

- În algoritmul anterior, toate operaţiile pe coloana j se fac în ultimul moment, după calculul coloanelor anterioare
- Coloana j e "beneficiarul" calculelor, de aici numele "fan-in" dat algoritmului
- Numărul de operaţii este minim, în sensul că operaţiile din $cmod$ se fac doar când ambii factori sunt nenuli

Algoritm fan-out

- Se poate adopta strategia opusă: de îndată ce o coloană este calculată, ea este utilizată pentru actualizarea coloanelor următoare
- Rezultă algoritmul "fan-out":

Pentru $k = 1 : n$
  $a_{kk} \leftarrow \sqrt{a_{kk}}$
  Pentru $i \in cstruct(L, k) \setminus \{k\}$ % $cdiv(k)$
    $a_{ik} \leftarrow a_{ik}/a_{kk}$
  Pentru $j \in cstruct(L, k) \setminus \{k\}$
    Pentru $i \in cstruct(L, k), i \geq j$ % $cmod(j, k)$
      $a_{ij} \leftarrow a_{ij} - a_{ik}a_{jk}$

- Avantaj: se accesează doar coloane, nu e necesară căutare pe linii
- Număr minim de operaţii

Algoritmi la nivel de bloc

- Algoritmii de mai sus funcţionează şi dacă se lucrează la nivel de bloc
- Factorizarea simbolică e identică !
- Pentru factorizarea numerică ţinem seama că $A = LL^T$ implică

$$A_{ij} = \sum_{k=1}^{j} L_{ik} L_{jk}^T, \quad i \geq j$$

- Modificări în factorizarea numerică:
  - $\sqrt{a_{kk}}$ se înlocuieşte cu o factorizare Cholesky a blocului $A_{kk} = L_{kk} L_{kk}^T$
  - $a_{ik} \leftarrow a_{ik}/a_{kk}$ se înlocuieşte cu $A_{ik} \leftarrow X$, unde X este soluţia sistemului superior triunghiular $X A_{kk}^T = A_{ik}$
  - $a_{ij} \leftarrow a_{ij} - a_{ik} a_{jk}$ devine $A_{ij} \leftarrow A_{ij} - A_{ik} A_{kj}^T$

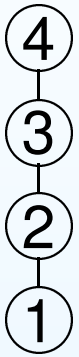
Permutare

- O permutare a matricei A poate modifica umplerea
- Dacă P este o matrice de permutare, calculăm factorizarea Cholesky a matricei $P^T AP$
- De obicei calculăm factorizarea Cholesky pentru rezolvarea sistemului $Ax = b$
- Permutarea revine la a rezolva $(P^T AP)(P^T x) = P^T b$:
  - Se factorizează $P^T AP = LL^T$
  - Se rezolvă sistemul inf. triunghiular $Ly = P^T b$
  - Se rezolvă sistemul sup. triunghiular $L^T z = y$
  - Soluţia este: $x = Pz$
- Micşorarea umplerii reduce numărul de operaţii în factorizarea Cholesky şi rezolvarea de sisteme triunghiulare, compensând permutările

Exemplu

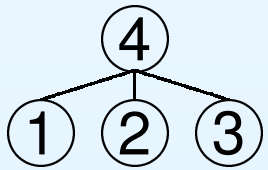
- Matrice săgeată cu factor L plin

$$A = \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & & \\ \times & & \times & \\ \times & & & \times \end{bmatrix}$$



- Matrice săgeată fără umplere

$$A = \begin{bmatrix} \times & & & \times \\ & \times & & \times \\ & & \times & \times \\ \times & \times & \times & \times \end{bmatrix}$$



- A doua matrice se obţine din prima prin permutarea inversă

Ordonare pe baza gradului minim

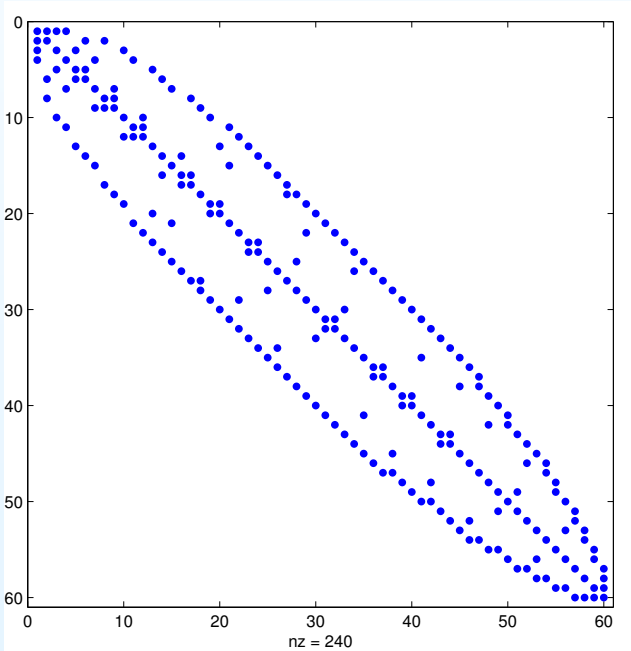
- Calculul permutării optime este dificil $\Rightarrow$ folosim euristici
- Reamintire: o coloană j din L transmite structura sa tuturor coloanelor din $cstruct(L, j)$. În concluzie, nodurile din $cstruct(L, j)$ formează o clică în graful asociat
- Cu cât această clică este mai mică, cu atât umplerea este mai mică
- Idee: la un moment dat, se aduce pe prima poziţie liberă nodul cu grad minim (coloana cu $|cstruct(L, j)|$ minim)
- Algoritmul e "lacom": nu se evaluează umplerea în adâncime, ci doar la pasul curent

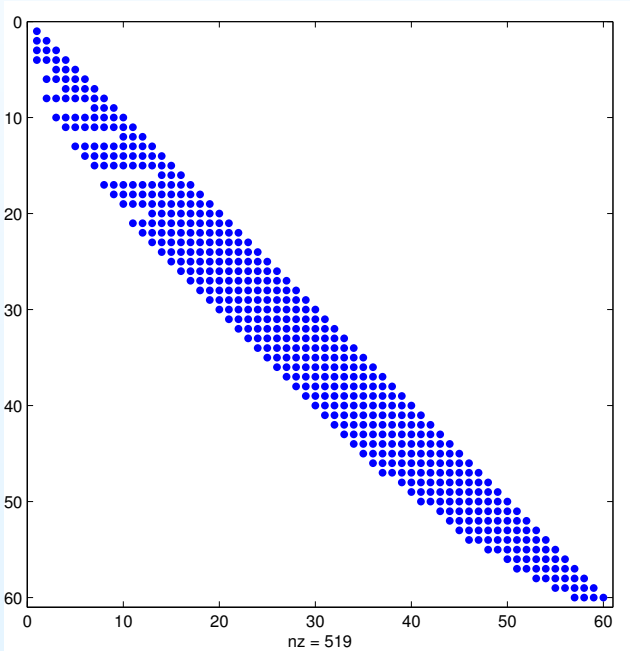
Algoritmul ”grad minim”

- Pas 1: se aduce în prima poziţie nodul cu grad minim din graful asociat lui A
- Se adaugă la graf arcele necesare pentru a forma o clică între nodurile legate de nodul cu grad minim
- Se actualizează gradele nodurilor
- Se elimină nodul din prima poziţie
- Pas 2: se continuă similar pe graful rămas
- Diverse variante ale algoritmului folosesc diverse metode de alegere între noduri cu acelaşi grad, aproximează gradul în loc să-l calculeze exact, etc.
- Funcţii Matlab: amd, symamd

Exemplu: matrice originală

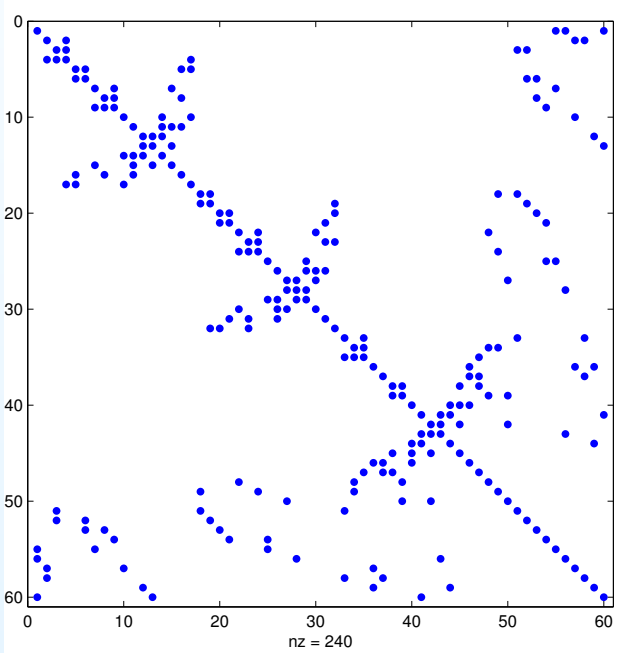
- Pentru matricea din stânga, cu 150 de elemente nenule în triunghiul inferior, factorul Cholesky din dreapta are 519 elemente nenule

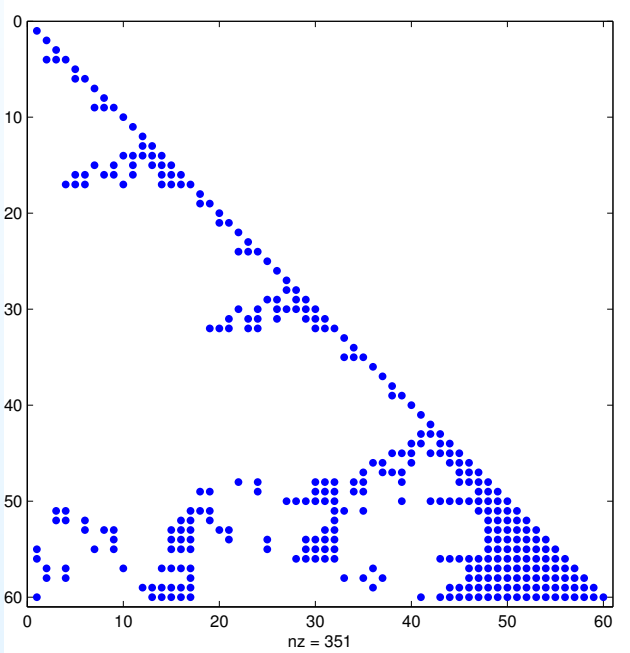




Exemplu: permutare AMD

- După permutarea cu funcţie symamd (stânga), factorul Cholesky (dreapta) are 351 elemente nenule





Nested dissection

- Idee: o permutare care aduce A la forma $\begin{bmatrix} B & 0 & * \\ 0 & C & * \\ * & * & S \end{bmatrix}$
- Factorul Cholesky va avea forma $\begin{bmatrix} L_B & 0 & 0 \\ 0 & L_C & 0 \\ * & * & \tilde{L} \end{bmatrix}$
- Interpretare: nodurile sunt grupate în trei mulţimi, astfel încât graful asociat să nu conţină arce între noduri din prima şi a doua
- B şi C se factorizează separat. Fiecare e permutat conform aceluiaşi algoritm
- Dificultate: găsirea rapidă a unui separator S de dimensiune cât mai mică

Utilitate

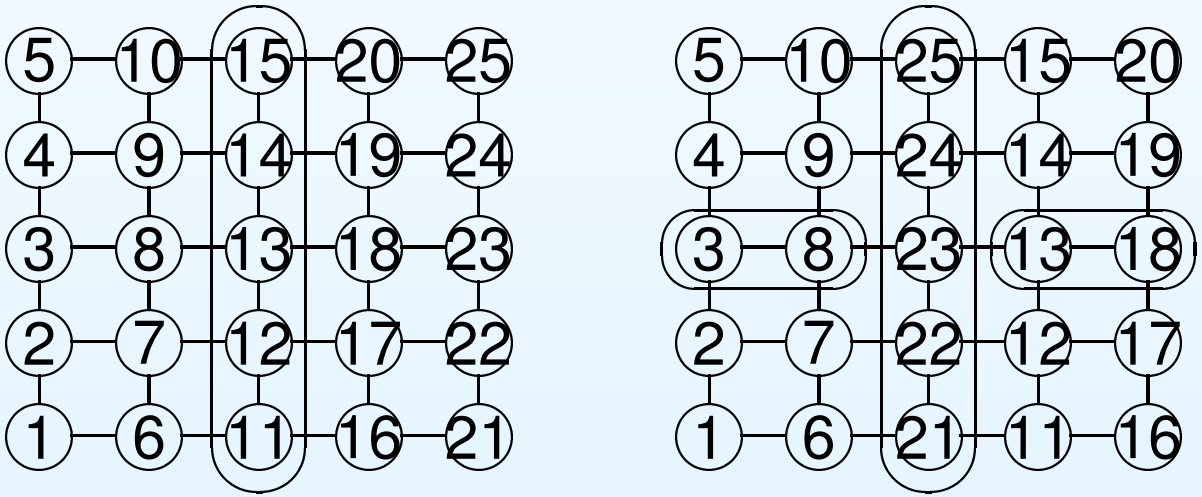
- Permutarea e utilă pentru factorizarea pe calculatoare paralele
- Exemplu, după două bisecţii

$$\begin{bmatrix} B_1 & 0 & * & 0 & 0 & 0 & * \\ 0 & C_1 & * & 0 & 0 & 0 & * \\ * & * & S_1 & 0 & 0 & 0 & * \\ 0 & 0 & 0 & B_2 & 0 & * & * \\ 0 & 0 & 0 & 0 & C_2 & * & * \\ 0 & 0 & 0 & * & * & S_1 & * \\ * & * & * & * & * & * & S \end{bmatrix}$$

- Blocurile B_1, C_1, B_2, C_2 pot fi factorizate în paralel. Bloc liniile 3 şi 6, idem
- Dacă blocurile au dimensiuni apropiate, eficienţă bună

Exemplu standard – grila

- Graful asociat matricei A are formă regulată pentru diverse probleme de discretizare
- Exemplu: grilă ordonată natural (stânga)
- Un separator evident este coloana centrală. Renumerotare (dreapta)



- Fiecare grup de coloane poate fi separat de linia centrală. Exerciţiu: renumerotaţi !

Rezolvarea sistemelor liniare

- Metode pentru rezolvarea sistemului $Ax = b$
  - directe, bazate pe factorizări (Cholesky, LU, QR)
  - iterative, în care fiecare iteraţie constă în produs matrice-vector şi/sau rezolvare de sisteme triunghiulare
- Factorizările LU şi QR utilizează aceleaşi etape ca Cholesky (permutare, factorizare simbolică), factorizare numerică), dar sunt mai complicate
- Metodele iterative construiesc aproximaţii din ce în ce mai bune ale soluţiei
- Pentru sisteme mari, metodele iterative pot fi eficiente, atât ca viteză, cât şi ca memorie (se memorează doar A)

Metode iterative

- Tipuri de metode iterative:
  - metode de relaxare (Jacobi, Gauss-Seidel, supra-relaxare)
  - metode de proiecţie (steepest descent, reziduu maxim)
  - metode pe subspaţii Krylov (Arnoldi, GMRES, Lanczos, gradient conjugat)
- Cursul va conţine, în măsura timpului disponibil, între 0 şi 3 din metodele Jacobi, Gauss-Seidel, proiecţie