4 Direkte Verfahren für spezielle Systeme

4.1 Die Cholesky-Zerlegung

Satz 4.1. *Es sei $A = [a_{i,j}] \in \mathbb{R}^{n\times n}$ [$\mathbb{C}^{n\times n}$] symmetrisch [Hermitesch]. Dann sind die folgenden Aussagen äquivalent:*

(a) A *ist positiv definit,*
d.h. $\boldsymbol{x}^\top A\boldsymbol{x} > 0 \; \forall \boldsymbol{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ [$\boldsymbol{x}^H A\boldsymbol{x} > 0 \; \forall \boldsymbol{x} \in \mathbb{C}^n \setminus \{\mathbf{0}\}$].

(b) Alle Eigenwerte von A *sind positiv.*

(c) Alle Hauptuntermatrizen von A *sind positiv definit.*

(d) Es gibt eine orthogonale Matrix $U \in \mathbb{R}^{n\times n}$ *(d.h.* $U^\top U = I_n$*) [eine unitäre Matrix* $U \in \mathbb{C}^{n\times n}$ *(d.h.* $U^H U = I_n$*)] und positive Zahlen* $\lambda_1, \lambda_2, \ldots, \lambda_n$ *mit*

$$U^\top AU = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n) \quad [U^H AU = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)].$$

(e) Für alle regulären Matrizen X *ist* $X^\top AX$ [$X^H AX$] *positiv definit.*

Satz 4.2. *Es sei $A \in \mathbb{R}^{n\times n}$ [$\mathbb{C}^{n\times n}$] symmetrisch [Hermitesch]. A ist genau dann positiv definit, wenn es eine reguläre untere Dreiecksmatrix $L \in \mathbb{R}^{n\times n}$ [$\mathbb{C}^{n\times n}$] gibt mit*

$$A = LL^\top \quad [A = LL^H] \qquad (\textit{Cholesky-Zerlegung}).$$

Bemerkung 4.3. *Statt der in Satz 4.2 beschriebenen Cholesky-Zerlegung wird oft die rationale (oder wurzelfreie) Cholesky-Zerlegung*

$$A = L_1 D L_1^\top \qquad [A = L_1 D L_1^H]$$

verwendet. Hier ist L_1 eine normierte untere $\triangle$-Matrix und D eine Diagonalmatrix mit positiven Hauptdiagonalelementen.
Mit $\Delta = \operatorname{diag}(L)$ gilt $L_1 = L\Delta^{-1}$ und $D = \Delta^2$.

Pseudocode: Cholesky-Zerlegung

for $j = 1 : n$
  $\ell_{j,j} := \left(a_{j,j} - \sum_{k=1}^{j-1} \ell_{j,k}^2 \right)^{1/2}$
  for $i = j + 1 : n$
    $\ell_{i,j} := \left(a_{i,j} - \sum_{k=1}^{j-1} \ell_{i,k} \ell_{j,k} \right) / \ell_{j,j}$

Aufwand: $\frac{1}{3} n^3 + O(n^2)$ flops.

Bemerkung: Der untere $\triangle$-Anteil von A wird kann mit L überschrieben werden. Der (echte) obere $\triangle$-Anteil von A wird im Algorithmus nicht verwendet.

Lösung eines linearen Gleichungssystems $A\boldsymbol{x} = \boldsymbol{b}$, A s.p.d. [H.p.d.] durch Cholesky-Zerlegung:

1: Berechne Cholesky-Zerlegung $A = LL^\top$
2: Löse $L\boldsymbol{y} = \boldsymbol{b}$
3: Löse $L^\top \boldsymbol{x} = \boldsymbol{y}$

Dieses Verfahren ist rückwärtsstabil:

Satz 4.4. *Die in Gleitpunktarithmetik mit Rundungseinheit u durch Cholesky-Zerlegung berechnete Lösung $\tilde{\boldsymbol{x}}$ des linearen Gleichungssystems $A\boldsymbol{x} = \boldsymbol{b}$ mit einer symmetrisch positiv-definiten Matrix A genügt einer Gleichung*

$$(A + \Delta A)\tilde{\boldsymbol{x}} = \boldsymbol{b} \qquad \textit{mit} \quad \|\Delta A\|_\infty \leq 3n^2 u \|A\|_\infty.$$

4.2 Bandmatrizen, Tridiagonalmatrizen

Man nennt $A = [a_{i,j}] \in \mathbb{C}^{n\times n}$ eine Bandmatrix mit unterer Bandbreite b_L und oberer Bandbreite b_R, falls $a_{i,j} = 0$ für $i - j > b_L$ und für $j - i > b_R$ gilt.

Eine Bandmatrix T mit $b_L = b_R = 1$ heißt Tridiagonalmatrix:

$$T = \operatorname{tridiag}(\boldsymbol{b}, \boldsymbol{a}, \boldsymbol{c}) = \begin{bmatrix} a_1 & c_1 & & & \\ b_2 & a_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{n-1} & a_{n-1} & c_{n-1} \\ & & & b_n & a_n \end{bmatrix}.$$

Besitzt die Tridiagonalmatrix T eine LR-Zerlegung, dann hat diese die Form

$$T = \begin{bmatrix} 1 & & & \\ \ell_2 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \ell_n \quad 1 \end{bmatrix} \begin{bmatrix} r_1 & c_1 & & \\ & r_2 & \ddots & \\ & & \ddots & \ddots \\ & & & r_{n-1} \quad c_{n-1} \\ & & & r_n \end{bmatrix}$$

mit $r_1 = a_1$ und $\ell_j = b_j / r_{j-1}$, $r_j = a_j - \ell_j c_{j-1}$ $(j = 2, 3, \ldots, n)$.

Ist Gauß-Elimination ohne Pivotsuche für $T\boldsymbol{x} = \boldsymbol{d}$ durchführbar, so erfordert sie also einen Aufwand von nur $3(n-1)$ Gleitpunktoperationen.

Dies ist aber nur bei speziellen (z.B. diagonaldominanten) T stabil:

Beispiel. Sei $T = \mathrm{tridiag}(2, 1, 3) \in \mathbb{R}^{n\times n}$ und $\boldsymbol{d} \in \mathbb{R}^n$ so gewählt, dass $\boldsymbol{x}^* = [1, 1, \ldots, 1]^\top$ das System $T\boldsymbol{x} = \boldsymbol{d}$ löst. Gauß-Elimination liefert die Lösung $\tilde{\boldsymbol{x}}$.

n	$\|\boldsymbol{x}^* - \tilde{\boldsymbol{x}}\|_2/\|\boldsymbol{x}^*\|_2$ (ohne Spaltenpivotsuche)	$\|\boldsymbol{x}^* - \tilde{\boldsymbol{x}}\|_2/\|\boldsymbol{x}^*\|_2$ (mit Spaltenpivotsuche)
100	$2.1\ldots \cdot 10^{-7}$	$1.3\ldots \cdot 10^{-16}$
200	$2.6\ldots \cdot 10^{+0}$	$9.3\ldots \cdot 10^{-17}$
500	$1.7\ldots \cdot 10^{27}$	$5.9\ldots \cdot 10^{-17}$

Spaltenpivotisierung führt allerdings zu einem sog. fill-in, d.h. auf Positionen, wo die Einträge der Ausgangsmatrix T Null waren, können in der LR-Zerlegung von Null verschiedene Elemente auftreten – was zusätzlichen Speicherplatz erfordert.

Bei Tridiagonalmatrizen liefert Gauß-Elimination mit Spaltenpivotsuche eine LR-Zerlegung der Form

$$\begin{bmatrix} 1 & & & & \\ \times & 1 & & & \\ & \ddots & \ddots & & \\ & & \times & 1 & \\ & & & \times & 1 \end{bmatrix} \cdot \begin{bmatrix} \times & \times & \times & & \\ & \times & \times & \times & \\ & & \ddots & \ddots & \ddots \\ & & & \times & \times & \times \\ & & & & \times & \times \\ & & & & & \times \end{bmatrix},$$

d.h. der fill-in beschränkt sich auf eine zusätzliche Diagonale in R.

Analoge Eigenschaften besitzt die Gauß-Elimination bei allgemeinen Bandmatrizen mit unterer Bandbreite b_L und oberer Bandbreite b_R:

Der Aufwand zur Berechnung der LR-Zerlegung beträgt (etwa) $(2b_L b_R + 1)n$ Flops. Wird nicht pivotisiert, so entsteht kein fill-in, d.h. L besitzt untere Bandbreite b_L und R besitzt obere Bandbreite b_R. Bei Spaltenpivotsuche verändert sich die Struktur von L nicht, R allerdings besitzt i. Allg. obere Bandbreite $b_L + b_R$.

Bei weniger strukturierten schwach besetzen Matrizen werden Umordnungsstrategien (z.B. reverse Cuthill-McKee) zur Bandbreitenminimierung eingesetzt (vgl. den nächsten Abschnitt).

4.3 Schwach besetzte Matrizen

Schwach besetzte Matrizen sind (üblicherweise sehr große) Matrizen, bei denen viele Einträge den Wert Null besitzen, oder pragmatischer: Matrizen mit so vielen Nulleinträgen, dass es sinnvoll ist, Algorithmen einzusetzen, die weder diese Nulleinträge speichern noch auf ihnen operieren. Wieviel fill-in bei der LR-Zerlegung schwach besetzter Matrizen auftritt, hängt entscheidend von der Nummerierung der Zeilen und Spalten ab: Für

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

liefert Gauß-Elimination (mit Spaltenpivotsuche)

$$PA = \begin{bmatrix} 1.0 & & & & \\ 1.0 & 1.0 & & & \\ 1.0 & & 1.0 & & \\ 1.0 & 1.0 & 1.0 & 1.0 & \\ 1.0 & 1.0 & 1.0 & 0.5 & 1.0 \end{bmatrix} \begin{bmatrix} 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \\ & -1.0 & & -1.0 & -1.0 \\ & & -1.0 & -1.0 & -1.0 \\ & & & 2.0 & 1.0 \\ & & & & 1.5 \end{bmatrix},$$

also eine (nahezu) voll besetzte LR-Zerlegung. (P entspricht der Permutation $(1\,3\,2\,4\,5)$.)

Vertauscht man in A die erste und letzte Zeile sowie die erste und letzte Spalte, so ergibt sich für $B = A([5\,2\,3\,4\,1],[5\,2\,3\,4\,1])$ eine viel schwächer besetzte LR-Zerlegung,

$$B = \begin{bmatrix} 1.0 & & & & \\ & 1.0 & & & \\ & & 1.0 & & \\ & & & 1.0 & \\ 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \end{bmatrix} \begin{bmatrix} 1.0 & & & & 1.0 \\ & 1.0 & & & 1.0 \\ & & 1.0 & & 1.0 \\ & & & 1.0 & 1.0 \\ & & & & -3.0 \end{bmatrix},$$

also keinerlei fill-in.

Wir beschreiben mit der umgekehrten Cuthill-McKee-Nummerierung eine Heuristik, die sich bei symmetrischen Problemen bewährt hat. Ziel ist es, die Zeilen und Spalten von A so umzunummerieren, dass die Bandbreite möglichst klein wird.

Wie benötigen den Begriff des gerichteten Graphen einer Matrix $A \in \mathbb{C}^{n\times n}$ (vorläufig beliebig).

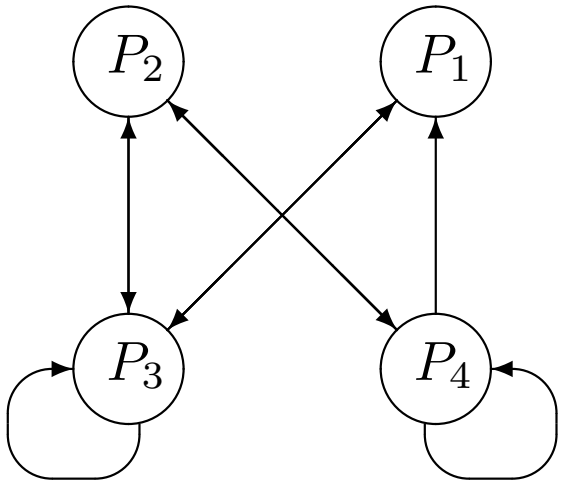
Der Graph $G(A) = (N, V)$ von A besteht aus $n = \dim(A)$ Knoten $P_1, P_2, \ldots, P_n$, die zusammen die Knotenmenge N bilden.

Für jedes Element $a_{i,j} \neq 0$ wird P_i mit P_j durch eine gerichtete Kante verbunden (von P_i nach P_j). Falls $a_{i,i} \neq 0$ ist, so wird P_i mit einer geschlossenen Schleife versehen. Die Gesamtheit aller Kanten bildet die Kantenmenge V. (Bei symmetrischen Matrizen oder allgemeiner bei Matrizen mit symmetrischer Besetzungsstruktur kann man auf die Orientierung der Kanten natürlich verzichten.)

Der Grad d_i eines Knotens P_i ist durch $d_i := |\{j : a_{i,j} \neq 0\}|$ definiert.

Beispiel.

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \quad \Rightarrow \quad G(A):$$

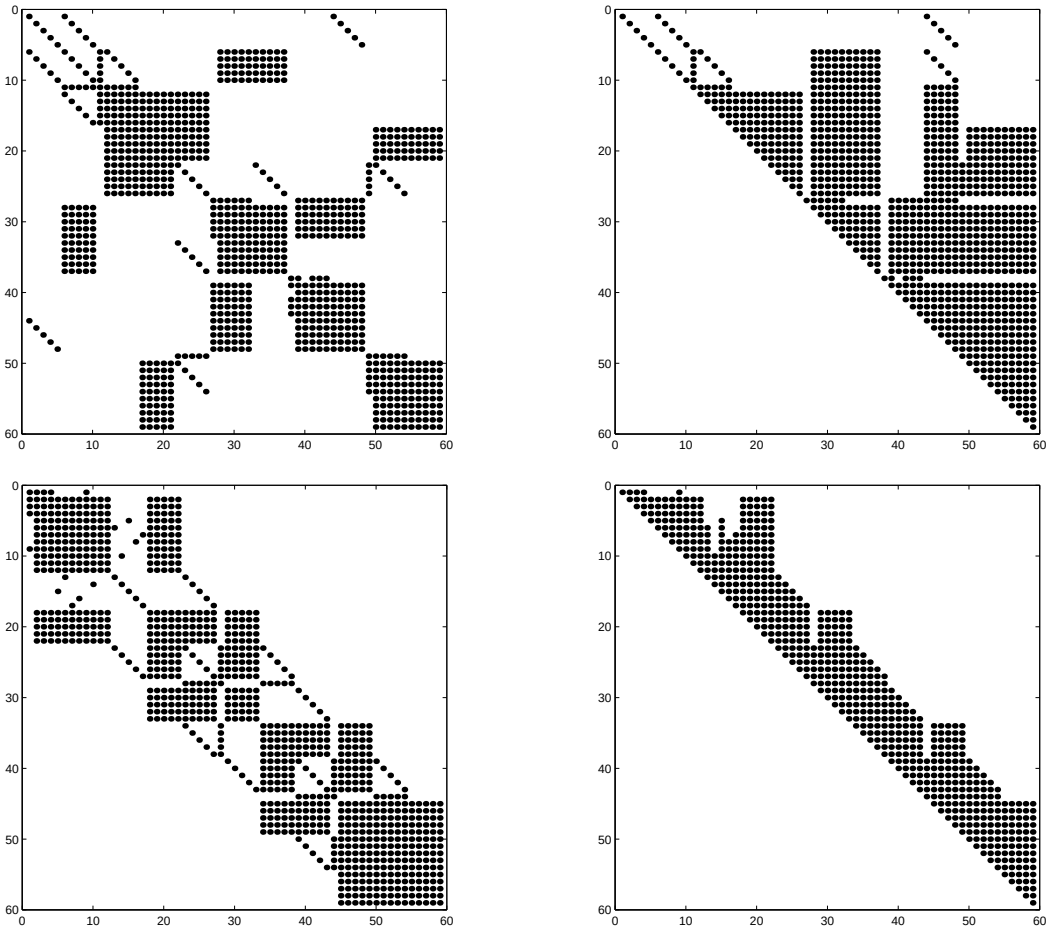


$d_1 = 1, d_2 = 2, d_3 = d_4 = 3.$

Cuthill-McKee-Nummerierung (CMK) für symmetrisches $A \in \mathbb{R}^{n \times n}$:

1: Wähle (beliebigen) Knoten aus $G(A)$ als Knoten P_1
2: **for** $i = 1, 2, \ldots, n-1$
3: Nummeriere die (noch unnummerierten) Nachbarn von P_i nach aufsteigendem Grad
4: Entferne alle Kanten zwischen bereits nummerierten Knoten und bestimme die Grade der noch unnummerierten Knoten bez. dieses neuen Graphen.

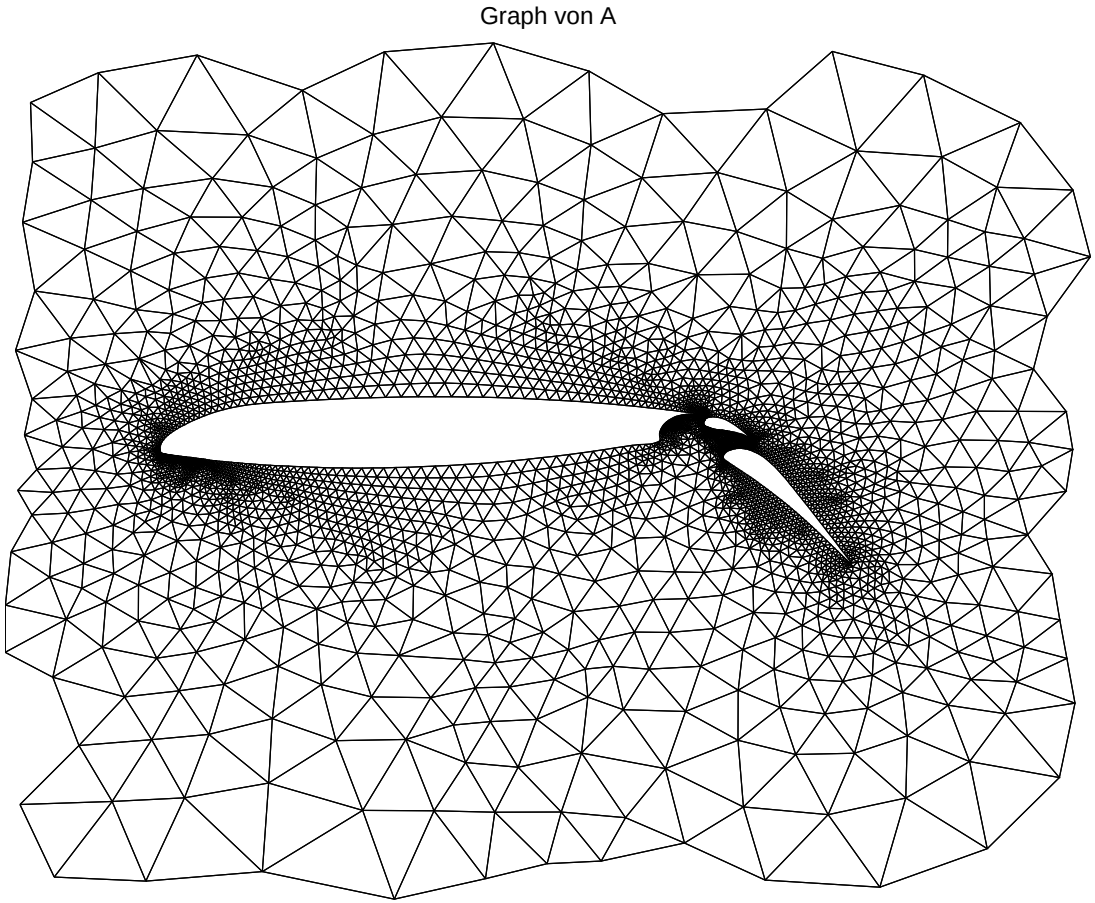
Bemerkungen. Diese Nummerierung ist nur möglich, wenn $G(A)$ zusammenhängend ist, d.h. wenn jeder Knoten von jedem anderen Knoten aus in $G(A)$ erreichbar ist.
Ist $[\pi(1), \pi(2), \ldots, \pi(n)]$ die CMK-Nummerierung, so ist die umgekehrte Cuthill-McKee-Nummerierung durch $[\pi(n), \pi(n-1), \ldots, \pi(1)]$ definert.

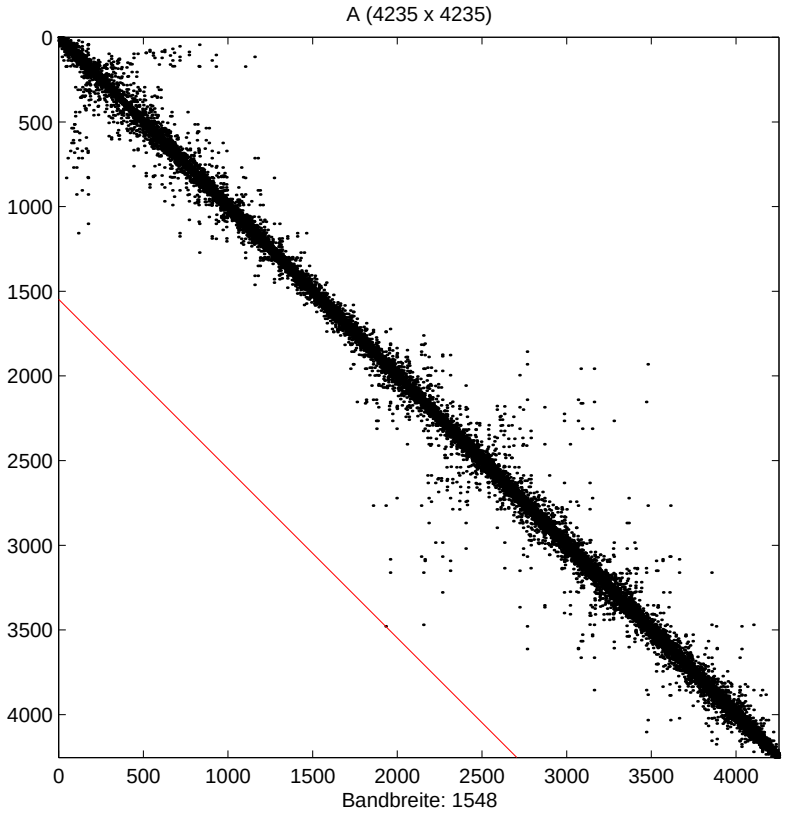

0
10
20
30
40
50
60

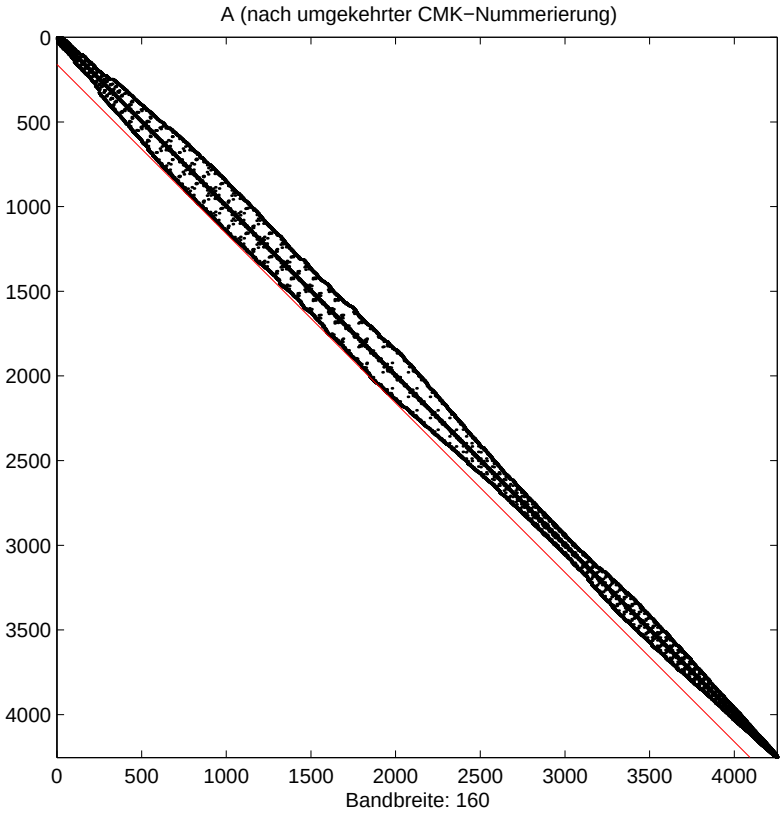
Legende. Links oben wird die Struktur der symmetrischen positiv definiten Matrix $A = CC^\top$ (auf die Bedeutung dieser Matrix wird später eingegangen) gezeigt, wobei die nichtsymmetrische Matrix $C \in \mathbb{R}^{59\times 59}$ aus der Modellierung eines chemischen Betriebs stammt (zu Details siehe `http://math.nist.gov:80/MatrixMarket/data/Harwell-Boeing/chemimp/impcolb.html`).
Rechts oben ist die Besetztheitsstruktur des Cholesky-Faktors $L_A^\top$ von A geplottet.
Sei P die Permutationsmatrix, die zur umgekehrten Cuthill-McKee-Nummerierung gehört, dann ist $B = PAP^\top$ (links unten) und $L_B^\top$ der zugehörige Cholesky-Faktor.

	A	L_A	B	L_B
# entries (i,j) $\neq 0$ $(i \leq j)$	503 (28%)	1073 (61%)	503 (28%)	657 (37%)

Weiteres Beispiel: Finite-Element Modell eines Tragflächenprofils.




A (4235 x 4235)
0
500
1000
1500
2000
2500
3000
3500
4000
Bandbreite: 1548


A (nach umgekehrter CMK−Nummerierung)
0
500
1000
1500
2000
2500
3000
3500
4000
Bandbreite: 160

4.4 Vandermondesche Matrizen

Eine Matrix der Bauart

$$V = V(x_0, x_1, x_2, \ldots, x_n) = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ x_0 & x_1 & x_2 & & x_n \\ x_0^2 & x_1^2 & x_2^2 & & x_n^2 \\ \vdots & & & \ddots & \vdots \\ x_0^n & x_1^n & x_2^n & \cdots & x_n^n \end{bmatrix} \in \mathbb{C}^{(n+1)\times(n+1)}$$

heißt Vandermondesche Matrix. Es gilt

$$\det\left(V(x_0, x_1, \ldots, x_n)\right) = \prod_{i>j}(x_i - x_j).$$

Insbesondere ist V genau dann regulär, wenn die x_j paarweise verschieden sind.

V ist zwar voll besetzt, besitzt aber eine sehr spezielle Struktur. Man erwartet, dass diese ausgenutzt werden kann zur Konstruktion von Algorithmen, die LGSe mit Koeffizientenmatrix V bzw. LGSe der Form

$$V^\top \boldsymbol{a} = \boldsymbol{f} \qquad \left(V \in \mathbb{R}^{(n+1)\times(n+1)}, \boldsymbol{f} \in \mathbb{R}^{n+1}\right) \tag{LGS}$$

mit deutlich weniger als $O(n^3)$ Flops lösen.

Wir betrachten die folgende polynomiale Interpolationsaufgabe:

Zu gegebenen (paarweise verschiedenen) Knoten $x_0, x_1, \ldots, x_n \in \mathbb{R}$ und gegebenen Funktionswerten $f_0, f_1, \ldots, f_n \in \mathbb{R}$ soll ein Interpolationspolynom

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \in \mathscr{P}_n$$

(mit reellen Koeffizienten $a_0, a_1, \ldots, a_n$ = $n+1$ Freiheitsgrade) vom Grad n konstruiert werden, das die $n+1$ Interpolationsbedingungen

$$p(x_i) = f_i \qquad (i = 0, 1, \ldots, n) \tag{IP}$$

erfüllt.

Satz 4.5. *Die Probleme* (LGS) *und* (IP) *sind äquivalent: Der Vektor*

$$\boldsymbol{a} = [a_0, a_1, \ldots, a_n]^\top \in \mathbb{R}^{n+1} \quad \textit{löst das LGS} \quad V^\top \boldsymbol{a} = \boldsymbol{f}$$

genau dann, wenn das Polynom

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \in \mathscr{P}_n$$

die Interpolationsbedingungen (IP) *erfüllt.*

Bemerkung 4.6. *Damit ist unsere Interpolationsaufgabe genau dann eindeutig lösbar, wenn die Knoten* $\{x_i\}_{i=0}^n$ *paarweise verschieden sind, d.h. es gibt in diesem Fall genau ein Polynom* $p \in \mathscr{P}_n$*, das den Bedingungen* (IP) *genügt.*

Wir lösen $V^\top \boldsymbol{a} = \boldsymbol{f}$ dadurch, dass wir die Koeffizienten des Interpolationspolynoms p in Newton-Form berechnen. Letztere verwendet die dividierten Differenzen der Wertepaare $\{(x_i, f_i)\}_{i=0}^n$, welche wie folgt definiert sind:

Sind $i_0, i_1, \ldots, i_k \in \{0, 1, \ldots, n\}$ paarweise verschieden, so setzen wir

$$f_{i_0,i_1,\ldots,i_k} := \frac{f_{i_1,i_2,\ldots,i_k} - f_{i_0,i_1,\ldots,i_{k-1}}}{x_{i_k} - x_{i_0}} \qquad (k \geq 1).$$

Satz 4.7 (Newton, 1669)**.** *Bezüglich der dividierten Differenzen lässt sich das Interpolationspolynom p in Newton-Form*

$$\begin{aligned} p(x) = f_0 &+ f_{0,1}(x - x_0) \\ &+ f_{0,1,2}(x - x_0)(x - x_1) \\ &+ \cdots \\ &+ f_{0,1,\ldots,n}(x - x_0)(x - x_1)\cdots(x - x_{n-1}) \end{aligned}$$

darstellen.

Die dividierten Differenzen der Wertepaare $\{(x_i, f_i)\}_{i=0}^{n}$ kann man sukzessive durch spaltenweises Auffüllen des Newton-Tableaus generieren. Für $n = 4$ erhält man beispielsweise

x_i	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$
x_0	f_0				
		$f_{0,1}$			
x_1	f_1		$f_{0,1,2}$		
		$f_{1,2}$		$f_{0,1,2,3}$	
x_2	f_2		$f_{1,2,3}$		$f_{0,1,2,3,4}$
		$f_{2,3}$		$f_{1,2,3,4}$	
x_3	f_3		$f_{2,3,4}$		
		$f_{3,4}$			
x_4	f_4				

Folgender Algorithmus berechnet die dividierten Differenzen:

Gegeben: Wertepaare $\{x_i, f_i\}_{i=0}^n$, x_i paarweise verschieden

for $k = 0 : n-1$
  for $j = n : -1 : k+1$
    $f_j := (f_j - f_{j-1})/(x_j - x_{j-k-1})$

Die im weiteren Verlauf nicht mehr benötigten Differenzen werden im obigen Algorithmus überschrieben. Dadurch wird kein zusätzlicher Speicherplatz benötigt. Am Ende stehen die dividierten Differenzen $f_0, f_{0,1}, f_{0,1,\ldots,n}$ in den Variablen $f_0, f_1, \ldots, f_n$.

Aufwand: $\frac{3}{2}(n^2 + n)$ Flops.

Um aus der Newton-Darstellung

$$p(x) = \sum_{k=0}^{n} f_{0,1,\ldots,k} \prod_{j=0}^{k-1} (x - x_j)$$

die Koeffizienten in $p(x) = a_0 + a_1 x + \cdots + a_n x^n$ zu ermitteln, beachte man, dass die Rekursion

$$\begin{aligned} p_n(x) &:= f_{0,1,\ldots,n}, \\ p_k(x) &:= f_{0,1,\ldots,k} + (x - x_k) p_{k+1}(x) \qquad (k = n-1, n-2, \ldots, 0) \end{aligned}$$

mit $p_0(x) = p(x)$ endet.

Durch Koeffizientenvergleich in dieser Rekursion erhält man folgenden Algorithmus.

Gegeben: Dividierte Differenzen der Newton-Darstellung in den Variablen f_0,...,f_n, zugehörige x-Werte $x_0, \ldots, x_n$ paarweise verschieden.

for $k = n-1 : 1 : 0$
  for $j = k : n-1$
    $f_j := f_j - f_{j+1} x_k$

Am Ende stehen die Koeffizienten $a_0, \ldots, a_n$ in den Variablen $f_0, \ldots, f_n$.
Aufwand: $n^2 + n$ Flops.

Fazit: Da das Hintereinanderausführen der beiden vorangehenden Algorithmen die Koeffizienten $\{a_i\}_{i=0}^n$ des Interpolationspolynoms der Wertepaare $\{(x_i, f_i)\}_{i=0}^n$ in $\frac{5}{2}n^2 + O(n)$ Flops liefert, ist damit auch die äquivalente Aufgabe, das LGS $V^\top \boldsymbol{a} = \boldsymbol{f}$ zu lösen, mit derselben Komplexität gelöst.

Mit den $(n+1) \times (n+1)$-Matrizen

$$D_k := \text{diag}(\underbrace{1, \ldots, 1}_{k+1}, x_{k+1} - x_0, \ldots, x_n - x_{n-k+1}),$$

$$L_k(\alpha) := \left[\begin{array}{c|c} I_k & O \\ \hline O & T_{n+1-k} \end{array}\right],$$

$k = 0, 1, \ldots, n-1$, wobei

$$T_{n+1-k} = \text{tridiag}(-\alpha, 1, 0) \in \mathbb{R}^{(n+1-k),\times(n+1-k)}, \quad \alpha \in \mathbb{R},$$

definieren wir eine normierte untere Δ-Matrix L sowie eine obere Δ-Matrix R gemäß

$$L := L_{n-1}(x_{n-1}) L_{n-2}(x_{n-2}) \ldots L_0(x_0),$$
$$R := L_0(1)^\top D_0^{-1} L_1(1)^\top D_1^{-1} \ldots L_{n-1}(1)^\top D_{n-1}^{-1}.$$

Satz 4.8. *Für* $V = V(x_0, x_1, \ldots, x_n)$ *ist*

$$V^{-1} = RL.$$

Mit anderen Worten: Der Algorithmus zur Lösung von $V^\top \boldsymbol{a} = \boldsymbol{f}$ *berechnet implizit eine RL-Zerlegung von* V^{-1}*.*

Dies kann zur Lösung von LGSen der Bauart $V\boldsymbol{z} = \boldsymbol{b}$ verwendet werden. Der folgende Algorithmus berechnet $\boldsymbol{z} = V^{-1}\boldsymbol{b}$ in $5n^2/2$ Flops. Eingabedaten sind $x_0, x_1, \ldots, x_n$ (paarweise verschieden), so dass $V = V(x_0, x_1, \ldots, x_n)$, und $\boldsymbol{b} = [b_0, b_1, \ldots, b_n]^\top$.
Die rechte Seite $\boldsymbol{b}$ wird mit der Lösung $\boldsymbol{z}$ überschrieben.

```
for k=0:n-1
    for j=n:-1:k+1
        b(j)=b(j)-x(k)*b(j-1)
    end
end
for k=n-1:-1:0
    for j=k+1:n
        b(j)=b(j)/(x(j)-x(j-k-1)
    end
    for j=k:n-1
        b(j)=b(j)-b(j+1)
    end
end
```

4.5 Toeplitz-Matrizen

Eine Matrix T heißt Toeplitz-Matrix, wenn sie die Form

$$T = \begin{bmatrix} t_0 & t_1 & t_2 & \cdots & \cdots & t_{n-1} \\ t_{-1} & t_0 & t_1 & & & \vdots \\ t_{-2} & t_{-1} & t_0 & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & t_0 & t_1 \\ t_{1-n} & \cdots & \cdots & \cdots & t_{-1} & t_0 \end{bmatrix},$$

also konstante Einträge auf jeder Diagonalen besitzt:

$$T = [t_{i-j}]_{1 \le i,j \le n} \in \mathbb{C}^{n \times n}.$$

Jede Toeplitz-Matrix ist persymmetrisch, d.h. symmetrisch zur SW-NO-Diagonalen.

Formal:

$$TE = ET^\top \quad \text{mit} \quad E := \begin{bmatrix} & & & 1 \\ & & \iddots & \\ & \iddots & & \\ 1 & & & \end{bmatrix}.$$

Beachte: Ist T invertierbar und persymmetrisch, so ist auch T^{-1} persymmetrisch (besitzt aber i.A. keine Toeplitz-Struktur), denn $E^2 = I$.

Wir interessieren uns hier ausschließlich für symmetrisch positiv definite Toeplitz-Matrizen. O.B.d.A. können wir dann von $t_0 = 1$ ausgehen.
Im Folgenden seien also $1 = t_0, t_1, \ldots, t_{n-1}$ so gewählt, dass die symmetrische Toeplitz-Matrix $T = [t_{|i-j|}]$ positiv definit ist.

Zunächst lösen wir positiv definite Toeplitz-Systeme mit spezieller rechter Seite (sog. Yule-Walker-Gleichungen):

$$\begin{bmatrix} 1 & t_1 & \cdots & t_{n-1} \\ t_1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & t_1 \\ t_{n-1} & \cdots & t_1 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix} = - \begin{bmatrix} t_1 \\ \vdots \\ t_{n-1} \\ t_n \end{bmatrix},$$

oder kurz, $T_n \boldsymbol{y} = -\boldsymbol{t}_n.$

Hierzu nehmen wir an, wir hätten das k-te Yule-Walker-System

$$T_k \boldsymbol{y} = -\boldsymbol{t}_k, \quad \boldsymbol{t}_k = [t_1, \ldots, t_k]^\top,$$

bereits gelöst und zeigen, dass das $(k+1)$-te System dann in $O(k)$ Schritten gelöst werden kann.

Dazu zerlegen wir das $(k+1)$-te System wie folgt:

$$\begin{bmatrix} T_k & E_k \boldsymbol{t}_k \\ \boldsymbol{t}_k^\top E_k & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{z} \\ \alpha \end{bmatrix} = - \begin{bmatrix} \boldsymbol{t}_k \\ t_{k+1} \end{bmatrix}, \quad \text{d.h.} \quad \begin{aligned} T_k \boldsymbol{z} + \alpha E_k \boldsymbol{t}_k &= -\boldsymbol{t}_k. \\ \boldsymbol{t}_k^\top E_k \boldsymbol{z} + \alpha &= -t_{k+1}. \end{aligned}$$

Wir erhalten

$$\begin{aligned} \boldsymbol{z} &= T_k^{-1}(-\boldsymbol{t}_k - \alpha E_k \boldsymbol{t}_k) = \boldsymbol{y} - \alpha T_k^{-1} E_k \boldsymbol{t}_k, \\ \alpha &= -t_{k+1} - \boldsymbol{t}_k^\top E_k \boldsymbol{z}. \end{aligned}$$

Wegen der Persymmetrie von T_k^{-1} gilt $T_k^{-1} E_k = E_k T_k^{-1}$ und damit

$$\begin{aligned} \boldsymbol{z} &= \boldsymbol{y} - \alpha E_k T_k^{-1} \boldsymbol{t}_k = \boldsymbol{y} + \alpha E_k \boldsymbol{y} \\ \alpha &= -t_{k+1} - \boldsymbol{t}_k^\top E_k (\boldsymbol{y} + \alpha E_k \boldsymbol{y}) = -(t_{k+1} + \boldsymbol{t}_k^\top E_k \boldsymbol{y} + \alpha \boldsymbol{t}_k^\top \boldsymbol{y}) \end{aligned}$$

und schließlich

$$\alpha = -\frac{t_{k+1} + \boldsymbol{t}_k^\top E_k \boldsymbol{y}}{1 + \boldsymbol{t}_k^\top \boldsymbol{y}}.$$

Wegen

$$\begin{bmatrix} I & E_k \boldsymbol{y} \\ O & 1 \end{bmatrix}^\top \begin{bmatrix} T_k & E_k \boldsymbol{t}_k \\ \boldsymbol{t}_k^\top E_k & 1 \end{bmatrix} \begin{bmatrix} I & E_k \boldsymbol{y} \\ O & 1 \end{bmatrix} = \begin{bmatrix} T_k & O \\ O & 1 + \boldsymbol{t}_k^\top \boldsymbol{y} \end{bmatrix}$$

und da T_{k+1} positiv definit, ist $1 + \boldsymbol{t}_k^\top \boldsymbol{y} > 0$.

Damit ist der k-te Schritt des Algorithmus von DURBIN vollendet, den wir nun folgendermaßen zusammenfassen können:

1: $y_1 = -t_1$
2: **for** $k = 1, 2, \ldots, n-1$
3: $\quad \beta_k = 1 + \boldsymbol{t}_k^\top \boldsymbol{y}_k$
4: $\quad \alpha_k = -(t_{k+1} + \boldsymbol{t}_k^\top E_k \boldsymbol{y}_k)/\beta_k$
5: $\quad \boldsymbol{y}_{k+1} = \begin{bmatrix} \boldsymbol{y}_k + \alpha_k E_k \boldsymbol{y}_k \\ \alpha_k \end{bmatrix}$

Diese Variante benötigt $3n^2 + O(n)$ Flops. Durch Ausnutzung einiger der obigen Beziehungen ist es jedoch möglich, den Rechenaufwand noch etwas zu reduzieren. Man beachte nämlich

$$\begin{aligned}\beta_k &= 1 + \boldsymbol{t}_k^\top \boldsymbol{y}_k = 1 + [\boldsymbol{t}_{k-1}^\top \, t_k] \begin{bmatrix} \boldsymbol{y}_{k-1} + \alpha_{k-1} E_{k-1} \boldsymbol{y}_{k-1} \\ \alpha_{k-1} \end{bmatrix} \\ &= \underbrace{1 + \boldsymbol{t}_{k-1}^\top \boldsymbol{y}_{k-1}}_{\beta_{k-1}} + \alpha_{k-1} \underbrace{(t_k + \boldsymbol{t}_{k-1}^\top E_{k-1} \boldsymbol{y}_{k-1})}_{-\alpha_{k-1}\beta_{k-1}} \\ &= (1 - \alpha_{k-1}^2)\beta_{k-1}.\end{aligned}$$

Durch Einbeziehung dieser zusätzlichen Rekursion gelangt man zur endgültigen Fassung des . . .

Algorithmus zur Lösung eines Yule-Walker-Systems [J. DURBIN (1960)].
Gegeben: $1 = t_0, t_1, \ldots, t_{n-1}, t_n$, so dass $T_n = [t_{|i-j|}]$ spd ist.
Gesucht: $\boldsymbol{y} = -T_n^{-1}\boldsymbol{t}_n$.

```
1:      y(1) = -t(1); beta = 1; alpha = -t(1);
2:      for k=1:n-1,
3:          beta = (1-alpha*alpha)*beta;
4:          alpha = -(t(k+1) + t(k:-1:1)'*y(1:k)) / beta;
5:          y(1:k) = y(1:k) + alpha*y(k:-1:1);
6:          y(k+1) = alpha;
7:      end
```

Aufwand: $2n^2 + O(n)$ flops.

Wir betrachten nun symmetrisch positiv definite Toeplitz-Systeme mit beliebiger rechter Seite:

$$\begin{bmatrix} 1 & t_1 & \cdots & t_{n-1} \\ t_1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & t_1 \\ t_{n-1} & \cdots & t_1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix},$$

oder kurz $T_n \boldsymbol{x} = \boldsymbol{b}$. Wir gehen dabei wieder von der Lösung des Systems

$$T_k \boldsymbol{x} = \boldsymbol{b}_k, \quad \boldsymbol{b}_k = [b_1, \ldots, b_k]^\top \tag{4.1}$$

aus und bestimmen daraus die Lösung des nächstgrößeren Systems

$$\begin{bmatrix} T_k & E_k \boldsymbol{t}_k \\ \boldsymbol{t}_k^\top E_k & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{v} \\ \mu \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_k \\ b_{k+1} \end{bmatrix}, \qquad \boldsymbol{t}_k = [t_1, \ldots, t_k]^\top. \tag{4.2}$$

Wir nehmen ferner an, die Lösung $\boldsymbol{y}$ des k-ten Yule-Walker-Systems $T_k \boldsymbol{y} = -\boldsymbol{t}_k$ sei verfügbar. Wegen $T_k \boldsymbol{v} + \mu E_k \boldsymbol{t}_k = \boldsymbol{b}_k$ folgt

$$\boldsymbol{v} = T_k^{-1}(\boldsymbol{b}_k - \mu E_k \boldsymbol{t}_k) = \boldsymbol{x} - \mu \underbrace{T_k^{-1} E_k}_{= E_k T_k^{-1}} \boldsymbol{t}_k = \boldsymbol{x} + \mu E_k \boldsymbol{y},$$

und somit

$$\mu = b_{k+1} - \boldsymbol{t}_k^\top E_k \boldsymbol{v} = b_{k+1} - \boldsymbol{t}_k^\top E_k \boldsymbol{x} - \mu \boldsymbol{t}_k^\top \boldsymbol{y}$$

d.h. $$\mu = \frac{b_{k+1} - \boldsymbol{t}_k^\top E_k \boldsymbol{x}}{1 + \boldsymbol{t}_k^\top \boldsymbol{y}}.$$

Damit ist der Schritt von (4.1) nach (4.2) in $6k + 1$ Operationen vollzogen. Bei diesem Algorithmus zur Lösung eines Toeplitz-Systems mit beliebiger rechter Seite werden also gleichzeitig das System $T_n \boldsymbol{x} = \boldsymbol{b}$ und die Yule-Walker-Gleichung $T_n \boldsymbol{y} = -\boldsymbol{t}_n$ schrittweise gelöst.

Algorithmus zur Lösung eines spd Toeplitz-Systems [N. LEVINSON (1947)].
Gegeben: $1 = t_0, t_1, \ldots, t_{n-1}$, so dass $T_n = [t_{|i-j|}]$ spd ist, $\boldsymbol{b} \in \mathbb{R}^n$.
Gesucht: $\boldsymbol{x} = T_n^{-1}\boldsymbol{b}$.

```
 1:       y(1) = -t(1); x(1) = b(1); beta = 1; alpha = -t(1);
 2:        for k=1:n-1,
 3:            beta = (1-alpha*alpha)*beta;
 4:            mu = (b(k+1) - t(1:k)'*x(k:-1:1)) / beta;
 5:            x(1:k) = x(1:k) + mu*y(k:-1:1);
 6:            x(k+1) = mu;
 7:            if k < n-1,
 8:               alpha = -(t(k+1) + t(1:k)'*y(k:-1:1)) / beta;
 9:               y(1:k) = y(1:k) + alpha*y(k:-1:1);
10:               y(k+1) = alpha;
11:            end
12:        end
```

Aufwand: $4n^2 + O(n)$ flops.

Schließlich bestimmen wir die Inverse einer spd Toeplitz-Matrix T_n, die wir wie folgt partitionieren:

$$T_n^{-1} = \begin{bmatrix} A & E\boldsymbol{t} \\ \boldsymbol{t}^\top E & 1 \end{bmatrix}^{-1} = \begin{bmatrix} B & \boldsymbol{v} \\ \boldsymbol{v}^\top & \gamma \end{bmatrix}, \qquad \begin{matrix} A = T_{n-1}, \\ E = E_{n-1}, \end{matrix} \quad \boldsymbol{t} = \boldsymbol{t}_{n-1}.$$

Aus

$$\begin{bmatrix} A & E\boldsymbol{t} \\ \boldsymbol{t}^\top E & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{v} \\ \gamma \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ 1 \end{bmatrix}$$

folgt zunächst

$$A\boldsymbol{v} = -\gamma E\boldsymbol{t}, \qquad \gamma = 1 - \boldsymbol{t}^\top E\boldsymbol{v}.$$

Löst $\boldsymbol{y}$ die $(n-1)$-te Yule-Walker-Gleichung $A\boldsymbol{y} = -\boldsymbol{t}$, so gilt

$$\begin{aligned} \boldsymbol{v} &= -\gamma A^{-1} E\boldsymbol{t} = -\gamma E A^{-1}\boldsymbol{t} = \gamma E\boldsymbol{y}, \\ \gamma &= 1 - \gamma \boldsymbol{t}^\top \boldsymbol{y}, \quad \text{d.h. } \gamma = 1/(1 + \boldsymbol{t}^\top \boldsymbol{y}). \end{aligned}$$

Damit sind die letzte Zeile und Spalte von T_n^{-1} bestimmt, und es verbleibt die Bestimmung einer Rekursion für die Matrix B. Hierzu beachte man, dass wegen $AB + E\boldsymbol{t}\boldsymbol{v}^\top = I_{n-1}$ sowie $A^{-1}E\boldsymbol{t} = EA^{-1}\boldsymbol{t} = -E\boldsymbol{y} = -\boldsymbol{v}/\gamma$ folgt

$$B = A^{-1} - A^{-1}E\boldsymbol{t}\boldsymbol{v}^\top = A^{-1} + \boldsymbol{v}\boldsymbol{v}^\top/\gamma.$$

Da A^{-1} persymmetrisch, d.h. $[A^{-1}]_{ij} = [A^{-1}]_{n-j,n-i}$, erhalten wir die Beziehung

$$\begin{aligned} b_{ij} &= [A^{-1}]_{ij} + \frac{v_i v_j}{\gamma} \\ &= [A^{-1}]_{n-j,n-i} + \frac{v_i v_j}{\gamma} \\ &= b_{n-j,n-i} - \frac{v_{n-j} v_{n-i}}{\gamma} + \frac{v_i v_j}{\gamma} \\ &= b_{n-j,n-i} + \frac{1}{\gamma}(v_i v_j - v_{n-j} v_{n-i}). \end{aligned} \tag{4.3}$$

Dies zeigt, dass B zwar nicht persymmetrisch ist, aber wir können einen Eintrag b_{ij} durch Spiegelung an der NO-SW-Diagonalen bestimmen. Unter zusätzlicher Ausnutzung der Persymmetrie von T_n^{-1} können wir deshalb B vom Rand nach innen berechnen. In der folgenden schematischen Darstellung des Algorithmus anhand eines 6×6 Beispieles stehen die Symbole u und b für unbekannte bzw. bereits bestimmte Matrixeinträge von T_n^{-1}. Wir beginnen mit der berechneten letzten Zeile und Spalte und nutzen dann abwechselnd die Persymmetrie von T_n^{-1} und Formel (4.3).

$$\begin{bmatrix} u & u & u & u & u & b \\ u & u & u & u & u & b \\ u & u & u & u & u & b \\ u & u & u & u & u & b \\ u & u & u & u & u & b \\ b & b & b & b & b & b \end{bmatrix} \rightarrow \begin{bmatrix} b & b & b & b & b & b \\ b & u & u & u & u & b \\ b & u & u & u & u & b \\ b & u & u & u & u & b \\ b & u & u & u & u & b \\ b & b & b & b & b & b \end{bmatrix} \rightarrow \begin{bmatrix} b & b & b & b & b & b \\ b & u & u & u & b & b \\ b & u & u & u & b & b \\ b & u & u & u & b & b \\ b & b & b & b & b & b \\ b & b & b & b & b & b \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} b & b & b & b & b & b \\ b & b & b & b & b & b \\ b & b & u & u & b & b \\ b & b & u & u & b & b \\ b & b & b & b & b & b \\ b & b & b & b & b & b \end{bmatrix} \rightarrow \begin{bmatrix} b & b & b & b & b & b \\ b & b & b & b & b & b \\ b & b & u & b & b & b \\ b & b & b & b & b & b \\ b & b & b & b & b & b \\ b & b & b & b & b & b \end{bmatrix} \rightarrow \begin{bmatrix} b & b & b & b & b & b \\ b & b & b & b & b & b \\ b & b & b & b & b & b \\ b & b & b & b & b & b \\ b & b & b & b & b & b \\ b & b & b & b & b & b \end{bmatrix}.$$

Den Algorithmus von TRENCH erhält man, wenn man noch berücksichtigt, dass T_n^{-1} als sowohl symmetrische als auch persymmetrische Matrix durch ihren „oberen Keil“ bestimmt ist, d.h. im Fall $n = 6$ durch den Anteil

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \\ & & \times & \times & & \end{bmatrix}.$$

Algorithmus [W. F. TRENCH (1964)]:

```
Loese T_{n-1} y = -t(1:n)' mit dem Algorithmus von Durbin
gamma = 1/(1 + t(1:n-1)'*y(1:n-1);
v(1:n-1) = gamma*y(n-1:-1:1);
B(1,1) = gamma;
B(1,2:n) = v(n-1:-1:1)';
for i=2:floor((n-1)/2)+1,
    for j=1:n-i+1,
        B(i,j) = B(i-1,j-1)+ ...
                 (v(n+1-j)*v(n+1-i)-v(i-1)*v(j-1)) / gamma;
    end
end
```

Aufwand: $\frac{13}{4}n^2 + O(n)$ (!) Flops.
Wie erwähnt werden nur $b_{i,j}$ mit $i \leq j, i + j \leq n + 1$ berechnet.