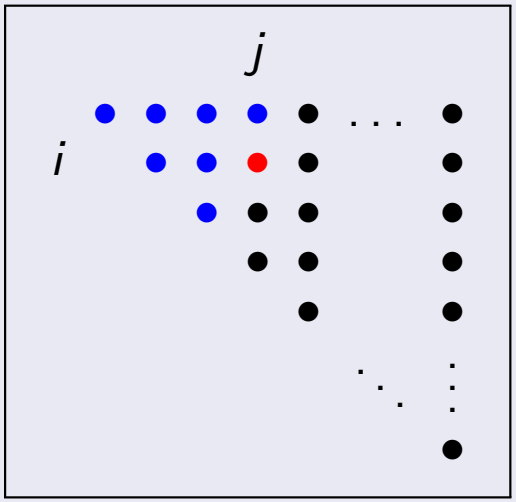
Speicherung und Verarbeitung spezieller Matrixstrukturen

Platzsparende Speicherung – Speicherabbildungsfunktion

- Unterscheide: Indizes i, j von a_{ij} und Speicherposition `A(k)`, wobei $k = k(i,j)$ die Speicherabbildungsfunktion ist.
- Symmetrie: wegen $a_{ij} = a_{ji}$ reicht es, das obere oder untere Dreieck (lückenlos) zu speichern, d. h. $n \cdot (n+1)/2$ Elemente statt n^2.



oberes Dreieck, spaltenweise
$k(i,j) = (j-1) \cdot j/2 + i$

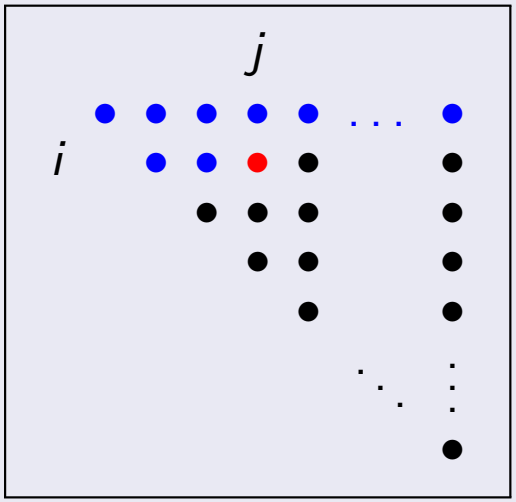
oberes Dreieck, zeilenweise
$k(i,j) = (n+1) \cdot n/2 - (n-i) \cdot (n-i+1)/2 - (n-j)$
$= (2n-i) \cdot (i-1)/2 + j$

- Meist lassen sich Algorithmen so formulieren, dass die Elemente der Matrix in der gespeicherten Reihenfolge verarbeitet werden (sequentieller Zugriff, ohne komplizierte Indexrechnung, cache-optimal)

Speicherung und Verarbeitung spezieller Matrixstrukturen

Platzsparende Speicherung – Speicherabbildungsfunktion

- Unterscheide: Indizes i, j von a_{ij} und Speicherposition `A(k)`, wobei $k = k(i,j)$ die Speicherabbildungsfunktion ist.
- Symmetrie: wegen $a_{ij} = a_{ji}$ reicht es, das obere oder untere Dreieck (lückenlos) zu speichern, d. h. $n \cdot (n+1)/2$ Elemente statt n^2.



oberes Dreieck, spaltenweise
$k(i,j) = (j-1) \cdot j/2 + i$

oberes Dreieck, zeilenweise
$k(i,j) = (n+1) \cdot n/2 - (n-i) \cdot (n-i+1)/2 - (n-j)$
$= (2n-i) \cdot (i-1)/2 + j$

- Meist lassen sich Algorithmen so formulieren, dass die Elemente der Matrix in der gespeicherten Reihenfolge verarbeitet werden (sequentieller Zugriff, ohne komplizierte Indexrechnung, cache-optimal)

Algorithmus für $y = Ax + b$ (1)

A: oberes Dreieck, zeilenweise

```
y=b   ! y_i = b_i ,  i = 1,...,n
k=0
DO i=1,n
  DO j=i,n
    k=k+1   !  Pos. von a_ij
    y(i)=y(i)+A(k)*x(j)
  END DO ! j
END DO ! i
```

A: oberes Dreieck, spaltenweise

```
y=b
k=0
DO j=1,n
  DO i=1,j
    k=k+1   !  Pos. von a_ij
    y(i)=y(i)+A(k)*x(j)
  END DO ! i
END DO ! j
```

Algorithmus für $y = Ax + b$ (2)

A: symmetrisch, zeilenweise

```
y=b   ! y_i = b_i ,  i = 1,...,n
k=0
DO i=1,n
  DO j=i,n
    k=k+1   !  Pos. von a_ij
    y(i)=y(i)+A(k)*x(j)
    IF (j .NE. i) THEN
      y(j)=y(j)+A(k)*x(i)
    END IF
  END DO ! j
END DO ! i
```

A: symmetrisch, spaltenweise

```
y=b
k=0
DO j=1,n
  DO i=1,j
    k=k+1   !  Pos. von a_ij
    y(i)=y(i)+A(k)*x(j)
    IF (i .NE. j) THEN
      y(j)=y(j)+A(k)*x(i)
    END IF
  END DO ! i
END DO ! j
```

IF-Anweisung in der inneren Schleife vermeidbar?

Algorithmus für $y = Ax + b$ (3)

A: symmetrisch, zeilenweise

```
y=b   ! y_i = b_i ,  i = 1,...,n
k=0
DO i=1,n
  k=k+1   !  Pos. von a_ii
  y(i)=y(i)+A(k)*x(i)
  DO j=i+1,n
    k=k+1   !  Pos. von a_ij
    y(i)=y(i)+A(k)*x(j)
    y(j)=y(j)+A(k)*x(i)
  END DO ! j
END DO ! i
```

A: symmetrisch, spaltenweise

```
y=b
k=0
DO j=1,n
  DO i=1,j-1
    k=k+1   !  Pos. von a_ij
    y(i)=y(i)+A(k)*x(j)
    y(j)=y(j)+A(k)*x(i)
  END DO ! i
  k=k+1   !  Pos. von a_jj
  y(j)=y(j)+A(k)*x(j)
END DO ! j
```

Sparse-Matrix-Techniken – Bandmatrizen, Profilspeicherung

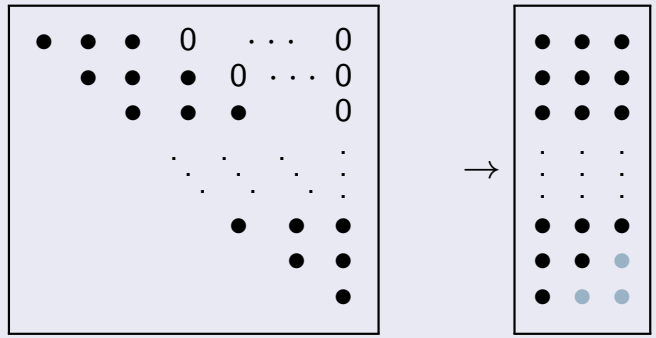
Ziel: sehr viele Nullen, die in einer (schwachbesetzten) Matrix vorkommen, werden nicht gespeichert, dann auch

keine unnötigen Zugriffe auf diese Elemente, Einsparung von nutzlosen Rechenoperationen.

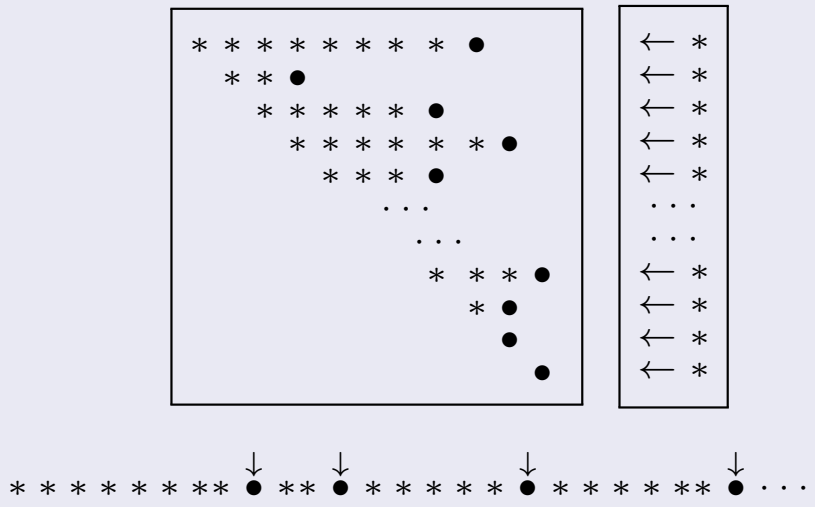
Eine $n \times n$-Matrix mit Bandweite m kann als rechteckige $n \times m$-Matrix gespeichert und verarbeitet werden.

Zeilenprofil: pro Zeile alle Elemente bis zum letzten Nichtnullelement, zusätzlich ein Indexvektor, der pro Zeile die Position des letzten gespeicherten Elements angibt.
(Spaltenprofil analog)

Bandmatrix:



Zeilenprofil:



Algorithmus für $y = Ax + b$ (4)

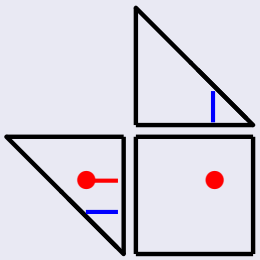
A: symmetrisch, oberes Dreieck als Zeilenprofil gespeichert (A,L)

```
y=b                              ! y_i = b_i ,  i = 1,...,n
JE=0                             ! Initialisierung
DO i=1,n                         ! fuer alle Zeilen
  JA=JE+1                        ! Position von a_ii
  JE=L(i)                        ! Position Zeilenende
  y(i)=y(i)+A(JA)*x(i)           ! Diagonalelement
  j=i                            ! Spaltenindex
  DO k=JA+1,JE                   ! Position von a_ij
    j=j+1                        ! naechster Spaltenindex
    y(i)=y(i)+A(k)*x(j)          ! a_ij x_j
    y(j)=y(j)+A(k)*x(i)          ! a_ji x_i
  END DO
END DO
```

Motivation zur Verwendung der Profilspeicherung

Profilspeicherung eignet sich auch für Faktorisierungsalgorithmen weil das Fill-In nur innerhalb des Profils auftritt (kein zusätzlicher Speicher).
Beispiel Cholesky-Zerlegung:

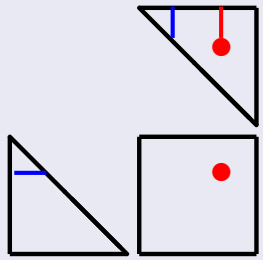
oberes Dreieck als Zeilenprofil
von rechts unten nach links oben
$A = RR^\top = L^\top L$



$$r_{ij} r_{jj} = a_{ij} - \sum_{k=j+1}^{n} r_{ik} r_{jk}$$

in Zeile i:
$a_{ij} = 0 \; \forall j > j_0 \Rightarrow r_{ij} = 0 \; \forall j > j_0$

oberes Dreieck als Spaltenprofil
von links oben nach rechts unten
$A = R^\top R = LL^\top$



$$r_{ii} r_{ij} = a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj}$$

in Spalte j:
$a_{ij} = 0 \; \forall i < i_0 \Rightarrow r_{ij} = 0 \; \forall i < i_0$

Sparse-Matrix-Techniken – Liste von Nichtnullelementen

- Ausschließliche Speicherung der Nichtnullelemente (NNE) erfordert zusätzlich eine Information über beide zugehörigen Indizes.
- Ungeordnete Speicherung ist möglich mit 3 sequentiellen Listen:
  `A(k)` – Wert eines Matrixelementes a_{ij}
  `indz(k)` – zugehöriger Zeilenindex i
  `inds(k)` – zugehöriger Spaltenindex j
- evtl. doppelte Verkettung durch „Zeiger“ auf Nachfolger in Zeile/Spalte
  `knxtz(k)` – Position nächstes NNE in Zeile i
  `knxts(k)` – Position nächstes NNE in Spalte j
- Bei zeilenweiser Anordnung ist eine Kombinationsform möglich (wie bei Profilspeicherung):
  `L(i)` – Position des letzten NNE aus Zeile i in der Liste A
  `A(k)` – NNE der Zeile i für $L_{i-1} < k \leq L_i$
  `is(k)` – zugehöriger Spaltenindex j

Algorithmus für $y = Ax + b$ (5)

A: symmetrisch, oberes Dreieck als Liste der NNE gespeichert (A,L,INDS)

```
y=b                          ! y_i = b_i ,  i = 1,...,n
JE=0                         ! Initialisierung
DO i=1,n                     ! fuer alle Zeilen
  JA=JE+1                    ! Position von a_ii
  JE=L(i)                    ! Position Zeilenende
  y(i)=y(i)+A(JA)*x(i)       ! Diagonalelement
  j=i                        ! Spaltenindex
  DO k=JA+1,JE               ! Position von a_ij
    j=INDS(k)                ! Spaltenindex j
    y(i)=y(i)+A(k)*x(j)      ! a_ij x_j
    y(j)=y(j)+A(k)*x(i)      ! a_ji x_i
  END DO
END DO
```

Algorithmus für $y = Ax + b$ (6)

A: beliebig, als Liste der NNE gespeichert (A,INDZ,INDS)

```
y=b                              ! y_i = b_i ,  i = 1,...,n
DO k=1,nne                       ! fuer alle Nichtnullelemente
    i=INDZ(k)                    ! Zeilenindex i
    j=INDS(k)                    ! Spaltenindex j
    y(i)=y(i)+A(k)*x(j)          ! a_ij x_j
END DO
```

Symmetrische Version funktioniert analog

```
    IF (i .NE. j) y(j)=y(j)+A(k)*x(i)   ! a_ji x_i
```