1.2. Special Matrices

- Complex Matrices
  - $\mathbf{A} \in C^{m\times n}$ implies that a_{ij} are complex numbers

$$a_{ij} = \alpha_{ij} + i\beta_{ij}, \qquad \alpha_{ij}, \beta_{ij} \in \Re,$$
$$i = 1, 2, ..., m, \qquad j = 1, 2, ..., n$$

  - Operations are the same except for:
    * Transposition: if $a_{ij} = \alpha_{ij} + i\beta_{ij}$ then

$$\mathbf{C} = \mathbf{A}^H = \bar{\mathbf{A}}^T, \qquad c_{ij} = \bar{a}_{ji} = \alpha_{ji} - i\beta_{ji}$$

    * Inner product: if $\mathbf{x}, \mathbf{y} \in C^n$ then

$$s = \mathbf{x}^H\mathbf{y} = \sum_{i=1}^{n} \bar{x}_i y_i$$

    * $\mathbf{x}^H\mathbf{x}$ is real

$$\mathbf{x}^H\mathbf{x} = \sum_{i=1}^{n} \bar{x}_i x_i = \sum_{i=1}^{n} |x_i|^2$$

Band Matrices

- **Definition 1**: $\mathbf{A} \in \Re^{m\times n}$ has *lower bandwidth* p if $a_{ij} = 0$ for $i > j + p$ and *upper bandwidth* q if $a_{ij} = 0$ for $j > i + q$.

$$\mathbf{A} = \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 & 0 \\ \times & \times & \times & \times & 0 & 0 \\ \times & \times & \times & \times & \times & 0 \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \end{bmatrix}$$

  - The $\times$ denotes an arbitrary nonzero entry
  - This 8×6 matrix has lower bandwidth 3 and upper bandwidth 1
  - *Example 1.* A 5×7 upper triangular matrix $p = 0$, $q = 6$

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times & \times \end{bmatrix}$$

Some Band Matrices

Matrix	p	q
Diagonal	0	0
Upper Triangular	0	$n-1$
Lower Triangular	$m-1$	0
Tridiagonal	1	1
Upper bidiagonal	0	1
Lower bidiagonal	1	0
Upper Hessenberg	1	$n-1$
Lower Hessenberg	$m-1$	1

- *Example 2.* A 5×6 tridiagonal matrix $p = q = 1$

$$\mathbf{A} = \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 & 0 \\ 0 & \times & \times & \times & 0 & 0 \\ 0 & 0 & \times & \times & \times & 0 \\ 0 & 0 & 0 & \times & \times & \times \end{bmatrix}$$

- *Example 3.* A 4×6 Lower Hessenberg matrix $p = 3$, $q = 1$

$$\mathbf{A} = \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 & 0 \\ \times & \times & \times & \times & 0 & 0 \\ \times & \times & \times & \times & \times & 0 \end{bmatrix}$$

Lower by Upper Triangular Matrix Product

- *Example 4* (cf. Golub and Van Loan (1996), Problem 1.2.3, p. 23).
- $\mathbf{L}$ and $\mathbf{U}$ are $n \times n$ lower and upper triangular matrices
- Give a column saxpy algorithm for computing

$$\mathbf{C} = \mathbf{U}\mathbf{L}$$

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} = [\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3]$$

- Express the columns of $\mathbf{C}$ as a linear combination of the columns of $\mathbf{U}$

$$\mathbf{c}_1 = \begin{bmatrix} u_{11} \\ 0 \\ 0 \end{bmatrix} l_{11} + \begin{bmatrix} u_{12} \\ u_{22} \\ 0 \end{bmatrix} l_{21} + \begin{bmatrix} u_{13} \\ u_{23} \\ u_{33} \end{bmatrix} l_{31}$$

$$\mathbf{c}_2 = \begin{bmatrix} u_{12} \\ u_{22} \\ 0 \end{bmatrix} l_{22} + \begin{bmatrix} u_{13} \\ u_{23} \\ u_{33} \end{bmatrix} l_{32}, \qquad \mathbf{c}_3 = \begin{bmatrix} u_{13} \\ u_{23} \\ u_{33} \end{bmatrix} l_{33}$$

Lower by Upper Triangular Matrix Product

- Use the saxpy matrix multiplication formula (1.7)

$$\mathbf{c}_j = \sum_{k=1}^{n} \mathbf{u}_k l_{kj}$$

- $l_{kj} = 0$ if $k < j$

$$\mathbf{c}_j = \sum_{k=j}^{n} \mathbf{u}_k l_{kj}$$

- $u_{ik} = 0$ if $i > k$

```
function C = suplow(U, L)
% suplow: compute the matrix product C = UL using saxpys
% where L and U are n × n lower and upper
% triangular matrices.
  [n n] = size(U);
  C = zeros(n);
  for j = 1:n
    for k = j:n
      C(1:k,j) = saxpy(L(k,j), U(1:k,k), C(1:k,j));
    end
  end
```

Lower by Upper Triangular Matrix Product

- Operation count:
  - A saxpy for a vector of length k requires k multiplications and k additions
  - The inner k loop has $n-j$ saxpys on vectors of length k and requires

    $$\sum_{k=j}^{n} k$$

    multiplications and additions
    * Recall

      $$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \tag{1}$$

    * The multiplications/additions in the inner loop are

      $$\sum_{k=j}^{n} k = \sum_{k=1}^{n} k - \sum_{k=1}^{j-1} k = \frac{1}{2}[n(n+1) - (j-1)j]$$

  - The multiplications/additions in the outer loop are

    $$\frac{1}{2}\sum_{j=1}^{n}[n(n+1) - (j-1)j]$$

    * Recall

      $$\sum_{i=1}^{n} i^2 = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} \tag{2}$$

Lower by Upper Triangular Matrix Product

- Thus, the total operation count is

$$\frac{1}{2}[n^2(n+1) - \frac{n^3}{3} - \frac{n^2}{2} - \frac{n}{6} + \frac{n(n+1)}{2}]$$

or

$$\frac{1}{2}[\frac{2n^3}{3} + n^2 + \frac{n}{3}] = \frac{(n+1)(2n+1)n}{6}$$

  - For large n, there are approximately $n^3/3$ multiplications and additions

- Multiplication of two $n \times n$ full matrices requires n^3 multiplications and additions

Band-Matrix Storage

- Let $\mathbf{A} \in \Re^{n\times n}$ have lower and upper bandwidth p and q
  - If $p, q \ll n$ then store $\mathbf{A}$ in either a $(p+q+1)\times n$ or a $n \times (p+q+1)$ matrix $\mathbf{A}^{band}$ to save storage
  - *Example 5.* Consider a 8×8 matrix with $p = 3$ and $q = 1$

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 & 0 & 0 & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} & 0 & 0 & 0 & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & 0 & 0 & 0 \\ 0 & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & 0 & 0 \\ 0 & 0 & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} & 0 \\ 0 & 0 & 0 & a_{74} & a_{75} & a_{76} & a_{77} & a_{78} \\ 0 & 0 & 0 & 0 & a_{85} & a_{86} & a_{87} & a_{88} \end{bmatrix}$$

  - *Row Storage*: “slide” all rows to the left or right to align the columns on anti-diagonals

$$\mathbf{A}^{band} = \begin{bmatrix} 0 & 0 & 0 & a_{11} & a_{12} \\ 0 & 0 & a_{21} & a_{22} & a_{23} \\ 0 & a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{52} & a_{53} & a_{54} & a_{55} & a_{56} \\ a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ a_{74} & a_{75} & a_{76} & a_{77} & a_{78} \\ a_{85} & a_{86} & a_{87} & a_{88} & 0 \end{bmatrix}$$

    * $\mathbf{A}^{band} \in \Re^{n\times p+q+1}$ with

$$a_{ij} = a^{band}_{i,j-i+p+1}$$

    * Can also store $\mathbf{A}$ by columns

Band Matrix-Vector Multiplication

- *Example 6.* Multiply a band matrix $\mathbf{A}$ by a vector $\mathbf{x}$

$$y_i = \sum_{j=\max(1,i-p)}^{\min(n,i+q)} a_{ij} x_j = \sum_{j=\max(1,i-p)}^{\min(n,i+q)} a^{band}_{i,j-i+p+1} x_j$$

function $\mathbf{y}$ = bmatvec(p, q, $\mathbf{A}^{band}$, $\mathbf{x}$)
% bmatvec: compute the matrix-vector product $\mathbf{y} = \mathbf{A}\mathbf{x}$
% where $\mathbf{A}^{band}$ is an $n \times n$ matrix stored in banded
% form by rows and $\mathbf{x}$ is an n-vector. The scalars p and q
% are the lower and upper bandwidths of $\mathbf{A}$, respectively.

% start and stop indicate the column index of the first and
% last nonzero entries in row i of $\mathbf{A}^{band}$.
% jstart and jstop indicate similar column indices for $\mathbf{A}$.

```
[n n] = size(A);
y(1:n) = 0;
for i = 1:n
   jstart = max(1, i-p);
   start = jstart - i + p + 1;
   jstop = min(n, i+q);
   stop = jstop - i + p + 1;
   y(i) = dot(A^band(i,start:stop)', x(jstart:jstop));
end
```

Symmetric Matrics

- **Definition 2**: $\mathbf{A} \in \Re^{n \times n}$ is *symmetric* if $\mathbf{A}^T = \mathbf{A}$

$$\mathbf{A} = \begin{bmatrix} 13 & 9 & -1 \\ 9 & 4 & 7 \\ -1 & 7 & -8 \end{bmatrix}$$

  - Only half of the matrix need be stored
  - Store the lower or upper triangular part of $\mathbf{A}$ as a vector

$$\mathbf{a}^{sym} = [13, 9, -1, 4, 7, -8]$$

  - Store the upper triangular part of $\mathbf{A}$ by rows

$$a_{ij} = a^{sym}_{(i-1)n - i(i-1)/2 + j} \tag{3a}$$

  - Arrange storage so that inner loops access contiguous data
    - *FORTRAN* stores matrices by columns
    - *C* stores matrices by rows
  - Symmetric matrix by vector multiplication

$$y_i = \sum_{j=1}^{n} a_{ij} x_j = \sum_{j=1}^{i-1} a_{ji} x_j + \sum_{j=i}^{n} a_{ij} x_j \tag{3b}$$

- Can also store $\mathbf{A}$ by diagonals

Block Matrices

- Partition the rows and columns of $\mathbf{A} \in \Re^{m \times n}$ into submatrices such that

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \cdots & \mathbf{A}_{1q} \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \cdots & \mathbf{A}_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{p1} & \mathbf{A}_{p2} & \cdots & \mathbf{A}_{pq} \end{bmatrix}$$

  – $\mathbf{A}_{ij} \in R^{m_i \times n_j}$, $i = 1 : m$, $j = 1 : n$

  – $m_1 + m_2 + ... + m_p = m$, $n_1 + n_2 + ... + n_q = n$

  – All operations on matrices with scalar components apply to those with matrix components

    * Matrix operations on the blocks are required

  – Let $\mathbf{B} \in \Re^{k \times l}$

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} & \cdots & \mathbf{B}_{1s} \\ \mathbf{B}_{21} & \mathbf{B}_{22} & \cdots & \mathbf{B}_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{B}_{r1} & \mathbf{B}_{r2} & \cdots & \mathbf{B}_{rs} \end{bmatrix}$$

    * $\mathbf{B}_{ij} \in \Re^{k_i \times l_j}$, $i = 1 : k$, $j = 1 : l$
    * $k_1 + k_2 + ... + k_r = k$, $l_1 + l_2 + ... + l_s = l$

Block Matrix
Addition and Multiplication

- *Addition*: a partition is *conformable* for addition if
  - $m = k$, $n = l$, $p = r$, $q = s$
  - $m_i = k_i$, $n_j = l_j$, $i = 1 : m$, $j = 1 : l$
  - Then $\mathbf{C} = \mathbf{A} + \mathbf{B}$ with $\mathbf{C}_{ij} = \mathbf{A}_{ij} + \mathbf{B_{ij}}$, $i = 1 : p$, $j = 1 : q$
- *Multiplication*: a partition is *conformable* for multiplication if
  - $n = k$, $q = r$, $n_j = k_j$, $j = 1 : q$
  - Then

$$\mathbf{C} = \mathbf{AB} = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} & \cdots & \mathbf{C}_{1s} \\ \mathbf{C}_{21} & \mathbf{C}_{22} & \cdots & \mathbf{C}_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{C}_{p1} & \mathbf{C}_{p2} & \cdots & \mathbf{C}_{ps} \end{bmatrix} \tag{4a}$$

with

$$\mathbf{C}_{ij} = \sum_{t=1}^{q} \mathbf{A}_{it}\mathbf{B}_{tj}, \qquad i = 1 : p, \qquad j = 1 : s \tag{4b}$$

Sparse Matrices

- *Definition*
  - Sparse matrices have a "significant" number of zeros
  - A matrix is considered to be sparse if the zeros are neither to be stored nor operated upon
- *Motivation*
  - Sparse matrices with arbitrary zero structures arise in finite element and network problems

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 2 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 6 & 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 & 0 \\ 0 & 0 & 0 & 0 & 12 \end{bmatrix}$$

- *Storage*
  - Store $\mathbf{A}$ as a vector $\mathbf{a}$ by rows or columns
  - An integer vector $\mathbf{ja}$ stores the column indices of $\mathbf{A}$
    - when $\mathbf{A}$ is stored by rows
  - A second vector $\mathbf{ia}$ stores the indices in $\mathbf{ja}$ of the first element of each row of $\mathbf{A}$
    - The last element of $\mathbf{ia}$, $ia_n + 1$, usually signals termination

Sparse Matrices

- For example

$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \end{bmatrix} \qquad \mathbf{ja} = \begin{bmatrix} 1 \\ 4 \\ 1 \\ 2 \\ 4 \\ 1 \\ 3 \\ 4 \\ 5 \\ 3 \\ 4 \\ 5 \end{bmatrix} \qquad \mathbf{ia} = \begin{bmatrix} 1 \\ 3 \\ 6 \\ 10 \\ 12 \\ 13 \end{bmatrix}$$

- The dimension of **a** and **ja** is the number of nonzeros in **A**. That of **ia** is $n + 1$
  - $ia_n + 1$ is set to the beginning index of a ficticious row $n + 1$
- The storage scheme is called the *compressed sparse row (CSR)* format
  - CSC is the *compressed sparse column* format
- Linked structures can be used instead of arrays

Sparse Matrices

- *Example 7.* Multiply a sparse matrix **A** by a vector **x**

```
function y = smatvec(a, ja, ia, x)
% smatvec: compute the matrix-vector product y = Ax
% where a is the n-by-n matrix A stored in CSR
% format, ja is a vector of pointers to the nonzero elements
% of A, ia is a vector pointing to the first nonzero
% element in each row of A.

   n = length(ia) - 1;
   for i = 1:n

% start and stop locate the beginning
% and end of row i in a and ja.

   start = ia(i);
   stop = ia(i+1) - 1;
   y(i) = 0;
   for k = start:stop
      y(i) = y(i) + a(k)*x(ja(k));
   end
end
```

- cf. Y. Saad (1996), *Iterative Methods for Sparse Linear Systems,* PWS, Boston, Section 3.4.