Notice that by storing A column by column in *A.band*, we obtain a column-oriented saxpy procedure. Indeed, Algorithm 1.2.2 is derived from Algorithm 1.1.4 by recognizing that each saxpy involves a vector with a small number of nonzeros. Integer arithmetic is used to identify the location of these nonzeros. As a result of this careful zero/nonzero analysis, the algorithm involves just $2n(p+q+1)$ flops with the assumption that p and q are much smaller than n.

1.2.6 Working with Diagonal Matrices

Matrices with upper and lower bandwidth zero are *diagonal*. If $D \in \mathbb{R}^{m\times n}$ is diagonal, then we use the notation

$$D = \text{diag}(d_1,\ldots,d_q), \quad q = \min\{m,n\} \quad \Longleftrightarrow \quad d_i = d_{ii}.$$

Shortcut notations when the dimension is clear include diag(d) and diag(d_i). Note that if $D = \text{diag}(d) \in \mathbb{R}^{n\times n}$ and $x \in \mathbb{R}^n$, then $Dx = d.*x$. If $A \in \mathbb{R}^{m\times n}$, then pre-multiplication by $D = \text{diag}(d_1,\ldots,d_m) \in \mathbb{R}^{m\times m}$ scales rows,

$$B = DA \quad \Longleftrightarrow \quad B(i,:) = d_i\cdot A(i,:),\ i = 1{:}m$$

while post-multiplication by $D = \text{diag}(d_1,\ldots,d_n) \in \mathbb{R}^{n\times n}$ scales columns,

$$B = AD \quad \Longleftrightarrow \quad B(:,j) = d_j\cdot A(:,j),\ j = 1{:}n.$$

Both of these special matrix-matrix multiplications require mn flops.

1.2.7 Symmetry

A matrix $A \in \mathbb{R}^{n\times n}$ is *symmetric* if $A^T = A$ and *skew-symmetric* if $A^T = -A$. Likewise, a matrix $A \in \mathbb{C}^{n\times n}$ is *Hermitian* if $A^H = A$ and *skew-Hermitian* if $A^H = -A$. Here are some examples:

$$\text{Symmetric:} \quad \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}, \qquad \text{Hermitian:} \quad \begin{bmatrix} 1 & 2-3i & 4-5i \\ 2+3i & 6 & 7-8i \\ 4+5i & 7+8i & 9 \end{bmatrix},$$

$$\text{Skew-Symmetric:} \begin{bmatrix} 0 & -2 & 3 \\ 2 & 0 & -5 \\ -3 & 5 & 0 \end{bmatrix}, \ \text{Skew-Hermitian:} \begin{bmatrix} i & -2+3i & -4+5i \\ 2+3i & 6i & -7+8i \\ 4+5i & 7+8i & 9i \end{bmatrix}.$$

For such matrices, storage requirements can bc halved by simply storing the lower triangle of elements, e.g.,

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix} \quad \Leftrightarrow \quad A.vec = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}.$$

For general n, we set

$$A.vec((n-j/2)(j-1)+i) \ = \ a_{ij} \quad 1 \le j \le i \le n. \qquad (1.2.2)$$

Here is a column-oriented gaxpy with the matrix A represented in $A.vec$.

Algorithm 1.2.3 (Symmetric Storage Gaxpy) Suppose $A \in \mathbb{R}^{n \times n}$ is symmetric and stored in the $A.vec$ style (1.2.2). If $x, y \in \mathbb{R}^n$, then this algorithm overwrites y with $y + Ax$.

```
for j = 1:n
    for i = 1:j - 1
        y(i) = y(i) + A.vec((i - 1)n - i(i - 1)/2 + j)x(j)
    end
    for i = j:n
        y(i) = y(i) + A.vec((j - 1)n - j(j - 1)/2 + i)x(j)
    end
end
```

This algorithm requires the same $2n^2$ flops that an ordinary gaxpy requires.

1.2.8 Permutation Matrices and the Identity

We denote the n-by-n *identity matrix* by I_n, e.g.,

$$I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

We use the notation e_i to designate the ith column of I_n. If the rows of I_n are reordered, then the resulting matrix is said to be a *permutation matrix*, e.g.,

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}. \tag{1.2.3}$$

The representation of an n-by-n permutation matrix requires just an n-vector of integers whose components specify where the 1's occur. For example, if $v \in \mathbb{R}^n$ has the property that v_i specifies the column where the "1" occurs in row i, then $y = Px$ implies that $y_i = x_{v_i}$, $i = 1{:}n$. In the example above, the underlying v-vector is $v = [\,2\,4\,3\,1\,]$.

1.2.9 Specifying Integer Vectors and Submatrices

For permutation matrix work and block matrix manipulation (§1.3) it is convenient to have a method for specifying structured integer vectors of subscripts. The MATLAB colon notation is again the proper vehicle and a few examples suffice to show how it works. If $n = 8$, then

$$\begin{array}{lcl} v = 1{:}2{:}n & \Longrightarrow & v = [\,1\,3\,5\,7\,], \\ v = n{:}{-1}{:}1 & \Longrightarrow & v = [\,8\,7\,6\,5\,4\,3\,2\,1\,], \\ v = [\,(1{:}2{:}n)\ (2{:}2{:}n)\,] & \Longrightarrow & v = [\,1\,3\,5\,7\,2\,4\,6\,8\,]. \end{array}$$