function A = blockdiags(B,d,m,n)

% BLOCKDIAGS : Create sparse block diagonal matrices.

%

% A = blockdiags(B,d,m,n).

%

% Blockdiags, which generalizes the function "spdiags",

% produces a sparse matrix with specified block diagonals.

%

% A is an m\*k-by-n\*k matrix, or an m-by-n matrix of k-by-k blocks.

% The nonzero blocks of A are located on p block diagonals.

% B is a min(m,n)\*k-by-p\*k matrix whose k-by-k block columns

% are the block diagonals of A.

% (Alternatively, B is k-by-p\*k, and then A is block Toeplitz.)

% d is a vector of p integers in the range -m+1 : n-1,

% specifying which block diagonals in A are to be nonzero.

% The values of p and k are determined from the dimensions of B and d.

%

% For k=1 this is exactly the same as A = spdiags(B,d,m,n); see spdiags

% for examples of use. For k>1 this is conceptually the same as spdiags,

% but k-by-k blocks replace matrix elements everywhere.

%

% For example, the following code sets A to the n^2-by-n^2 matrix of

% the Laplacian on an n-by-n square grid; the matrix is block tridiagonal,

% and the nonzero blocks themselves are tridiagonal or the identity.

%

% a = blockdiags ([-1 4 -1], -1:1, n, n);

% I = speye (n, n);

% A = blockdiags ([-I a -I], -1:1, n, n);

%

% John Gilbert, Xerox PARC, 17 April 1991.

% Copyright (c) 1990-1996 by Xerox Corporation. All rights reserved.

% HELP COPYRIGHT for complete copyright and licensing notice.

if nargin ~= 4

error ('Usage: A = blockdiags(B,d,m,n)');

end

k = length(d);

[nrB,ncB] = size(B);

p = ncB/k;

% Check for reasonable input.

if any(size(m)>1) | any(size(n)>1)

error ('blockdiags(B,d,m,n): m or n not scalar');

end

if min(size(d)~=1)

error ('blockdiags(B,d,m,n): d not a vector');

end

if any(rem(size(B),p))

error ('blockdiags(B,d,m,n): block size does not divide size of B');

end

% Process A in compact form.

[i,j,a] = find(B);

nzB = length(a);

% Duplicate rows of B if A is to be block Toeplitz.

if nrB == p

nzA = m\*nzB;

range = [0:nzA-1]';

r = 1 + rem(range,nzB);

a = a(r);

i = i(r);

j = j(r);

a = a(:);

i = i(:) + p \* floor(range/nzB);

j = j(:);

end

% Rows of B are rows of A. Shift columns appropriately.

dd = d(1+floor((j-1)/p));

dd = dd(:);

j = 1 + rem(j-1,p) + p \* (dd + floor((i-1)/p));

% Clip columns that shifted off the matrix.

f = find(j>=1 & j<=n\*p);

a = a(f);

i = i(f);

j = j(f);

% Put A back into sparse form.

A = sparse (i,j,a,m\*p,n\*p);