```
>> disp('LDL')
LDL
>> A=rand(10,10);
>> A=A+A';
>> chol(A)
??? Error using ==> chol
Matrix must be positive definite.

>> ldl(A)
ans =
  Columns 1 through 8
    1.0000         0         0         0         0         0         0         0
    1.5542    1.0000         0         0         0         0         0         0
    0.4149    0.1865    0.6358   -1.1233    0.6878    1.0000         0         0
    0.7132    0.2370   -0.0217   -0.5440    0.9127    1.5612    0.5426    0.7242
    0.3824    0.4723    0.5842    0.6617    0.8596    0.2598    1.0000         0
    0.4165    0.4233    0.1142    1.0000         0         0         0         0
    0.8635    0.6194    0.1963   -0.1045    0.1260    0.0658    0.3193   -0.1346
    0.9943    0.2465   -0.2568   -0.1480    0.4423    0.9788   -0.0651    1.0000
    0.1813         0    1.0000         0         0         0         0         0
    1.1971    0.8754   -0.4033         0    1.0000         0         0         0
  Columns 9 through 10
         0         0
         0         0
         0         0
    1.0000         0
         0         0
         0         0
    0.1597    1.0000
         0         0
         0         0
         0         0
>> [L,D]=ldl(A)
L =
  Columns 1 through 8
    1.0000         0         0         0         0         0         0         0
    1.5542    1.0000         0         0         0         0         0         0
    0.4149    0.1865    0.6358   -1.1233    0.6878    1.0000         0         0
    0.7132    0.2370   -0.0217   -0.5440    0.9127    1.5612    0.5426    0.7242
    0.3824    0.4723    0.5842    0.6617    0.8596    0.2598    1.0000         0
    0.4165    0.4233    0.1142    1.0000         0         0         0         0
    0.8635    0.6194    0.1963   -0.1045    0.1260    0.0658    0.3193   -0.1346
    0.9943    0.2465   -0.2568   -0.1480    0.4423    0.9788   -0.0651    1.0000
    0.1813         0    1.0000         0         0         0         0         0
    1.1971    0.8754   -0.4033         0    1.0000         0         0         0
  Columns 9 through 10
```

```
         0         0
         0         0
         0         0
    1.0000         0
         0         0
         0         0
    0.1597    1.0000
         0         0
         0         0
         0         0
D =
  Columns 1 through 8
    1.1357         0         0         0         0         0         0         0
         0   -0.7631    1.4840         0         0         0         0         0
         0    1.4840    0.6928         0         0         0         0         0
         0         0         0   -0.0446    0.9414         0         0         0
         0         0         0    0.9414    0.3683         0         0         0
         0         0         0         0         0    1.0716         0         0
         0         0         0         0         0         0   -1.2222         0
         0         0         0         0         0         0         0   -1.8038
         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0
  Columns 9 through 10
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
   -0.4437         0
         0    0.1434
>> disp('Automatic Differentiation')
Automatic Differentiation
>> D
D =
  Columns 1 through 8
    1.1357         0         0         0         0         0         0         0
         0   -0.7631    1.4840         0         0         0         0         0
         0    1.4840    0.6928         0         0         0         0         0
         0         0         0   -0.0446    0.9414         0         0         0
         0         0         0    0.9414    0.3683         0         0         0
         0         0         0         0         0    1.0716         0         0
         0         0         0         0         0         0   -1.2222         0
         0         0         0         0         0         0         0   -1.8038
```

```
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
  Columns 9 through 10
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
  -0.4437     0
     0  0.1434
>> inv(D)
ans =
  Columns 1 through 8
   0.8805     0     0     0     0     0     0     0
     0  -0.2537  0.5434     0     0     0     0     0
     0   0.5434  0.2794     0     0     0     0     0
     0     0     0  -0.4081  1.0430     0     0     0
     0     0     0   1.0430  0.0494     0     0     0
     0     0     0     0     0  0.9332     0     0
     0     0     0     0     0     0  -0.8182     0
     0     0     0     0     0     0     0  -0.5544
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
  Columns 9 through 10
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
  -2.2540     0
     0  6.9716
>> disp('Sparsity');
Sparsity
>> m=10;
>> lap=2*diag(ones(m,1))-diag(ones(m-1,1),1)-diag(ones(m-1,1),-1);
>> spy(lap)
>> spy(lap)
>> help(sparse)
??? Error using ==> sparse
Not enough input arguments.
```

```
>> help sparse
   SPARSE Create sparse matrix.
   S = SPARSE(X) converts a sparse or full matrix to sparse form by squeezing
   out any zero elements.

   S = sparse(i,j,s,m,n,nzmax) uses vectors i, j, and s to generate an m-by-n
   sparse matrix such that S(i(k),j(k)) = s(k), with space allocated for nzmax
   nonzeros. Vectors i, j, and s are all the same length. Any elements of s
   that are zero are ignored, along with the corresponding values of i and
   j. Any elements of s that have duplicate values of i and j are added
   together.To simplify this six-argument call, you can pass scalars for the
   argument s and one of the arguments i or j.in which case they are expanded
   so that i, j, and s all have the same length.

   The simplifications of the six-argument call are:

   S = SPARSE(i,j,s,m,n) where nzmax = length(s).

   S = SPARSE(i,j,s) where m = max(i) and n = max(j).

   S = SPARSE(m,n) abbreviates SPARSE([],[],[],m,n,0).  This
   generates the ultimate sparse matrix, an m-by-n all zero matrix.

   The argument s and one of the arguments i or j may be scalars,
   in which case they are expanded so that the first three arguments
   all have the same length.

   For example, this dissects and then reassembles a sparse matrix:

         [i,j,s] = find(S);
         [m,n] = size(S);
         S = sparse(i,j,s,m,n);

   So does this, if the last row and column have nonzero entries:

         [i,j,s] = find(S);
         S = sparse(i,j,s);

   All of MATLAB's built-in arithmetic, logical and indexing operations
   can be applied to sparse matrices, or to mixtures of sparse and
   full matrices.  Operations on sparse matrices return sparse matrices
   and operations on full matrices return full matrices.  In most cases,
   operations on mixtures of sparse and full matrices return full
   matrices.  The exceptions include situations where the result of
   a mixed operation is structurally sparse, eg.  A .* S is at least
```

```
    as sparse as S .  Some operations, such as S >= 0, generate
    "Big Sparse", or "BS", matrices -- matrices with sparse storage
    organization but few zero elements.

    See also issparse, spalloc, spones, speye, spconvert, full, find, sparfun.

    Overloaded methods:
      intval/sparse
      gradient/sparse
      hessian/sparse
      taylor/sparse
      slope/sparse

    Reference page in Help browser
      doc sparse

>> [i,j,s]=find(A);
>> S=sparse(i,j,s);
>> w=rand(m,1);
>> tic; S\w; toc
Elapsed time is 0.001918 seconds.
>> tic; A\w; toc
Elapsed time is 0.000346 seconds.
>> tic; S\w; toc
Elapsed time is 0.000105 seconds.
>> m=1000;
>> lap=2*diag(ones(m,1))-diag(ones(m-1,1),1)-diag(ones(m-1,1),-1);
>> [i,j,s]=find(lap);
>> S=sparse(i,j,s);
>> w=rand(m,1);
>> tic; S\w; toc
Elapsed time is 0.000099 seconds.
>> tic; lap\w; toc
Elapsed time is 0.080999 seconds.
>> m=10000;
>> lap=2*diag(ones(m,1))-diag(ones(m-1,1),1)-diag(ones(m-1,1),-1);
>> [i,j,s]=find(lap);
>> S=sparse(i,j,s);
>> w=rand(m,1);
>> tic; S\w; toc
Elapsed time is 0.000715 seconds.
>> tic; lap\w; toc
Elapsed time is 40.217010 seconds.
>> pwd
ans =
/Users/anitescu/Documents/UNIXLIKE/ANITESCU/ATWORK/TEACHING/2011/
```

```
CODE/NEWTON
>> ls
fenton.m                newtonLikeIteration.m
fenton_wrap.m                   newtonLikeMethod.m

>> [xout,iteratesGradNorms]=newtonLikeMethod(@fenton_wrap,[3 4]',1,1e-12,200)
xout =
   NaN
   NaN
iteratesGradNorms =
   1.0e+18 *
    0.0000
    0.0000
    0.0000
```

```
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0001
    0.0096
    0.0048
    0.0023
    0.0011
    0.0004
    0.0009
    0.0070
    0.0019
    0.0098
    0.1739
    0.0804
    0.0231
    0.1001
    1.4939
    0.6673
      NaN
>> [xout,iteratesGradNorms]=newtonLikeMethod(@fenton_wrap,[3 2]',1,1e-12,200)
xout =
    1.7435
    2.0297
iteratesGradNorms =
    0.5514
   35.8698
   13.9590
    5.4318
    2.0896
```

```
    0.7623
    0.2364
    0.0496
    0.0043
    0.0000
    0.0000
    0.0000
>> format long
>> [xout,iteratesGradNorms]=newtonLikeMethod(@fenton_wrap,[3 2]',1,1e-12,200)
xout =
   1.743452086941417
   2.029694710000688
iteratesGradNorms =
   0.551447440605826
  35.869783141008888
  13.959019599207084
   5.431810148871664
   2.089564861428548
   0.762299983874110
   0.236390708796542
   0.049644124388102
   0.004346715638112
   0.000048585039581
   0.000000006600513
   0.000000000000000
>> [xout,iteratesGradNorms]=newtonLikeMethod(@fenton_wrap,[3 4]',1,1e-12,200)
xout =
   NaN
   NaN
iteratesGradNorms =
   1.0e+18 *
   0.000000000000000
   0.000000000000000
   0.000000000000000
   0.000000000000000
   0.000000000000000
   0.000000000000000
   0.000000000000000
   0.000000000000000
   0.000000000000000
   0.000000000000000
   0.000000000000000
   0.000000000000039
   0.000000000000019
   0.000000000000009
   0.000000000000004
```

0.000000000000001
0.000000000000008
0.000000000000258
0.000000000000125
0.000000000000055
0.000000000000005
0.000000000000580
0.000000001455113
0.000000000727539
0.000000000363735
0.000000000181799
0.000000000090763
0.000000000045106
0.000000000021987
0.000000000009742
0.000000000001146
0.000000000068137
0.000000047423993
0.000000023708027
0.000000011846069
0.000000005907101
0.000000002921318
0.000000001393209
0.000000000535294
0.000000000390841
0.000000000648065
0.000000008450557
0.000000170903974
0.000000080478323
0.000000027835669
0.000000047096135
0.000000220512353
0.000000076507615
0.000001039760686
0.000022549416943
0.000010700820832
0.000003954906922
0.000004145388553
0.000005438712481
0.000097497942953
0.009576456574615
0.004770589234759
0.002349464470413
0.001098505034618
0.000354806829224
0.000889140675198

```
   0.007022698448425
   0.001943083375274
   0.009772154931589
   0.173900636132894
   0.080449224286501
   0.023091284205416
   0.100065699075068
   1.493897536540798
   0.667264703289445
             NaN
>>
```