```
% LMtestNLP1:  Set up and solve the following nonlinear model using LINDO API's
%              multi-start nonlinear optimizer.
%
%             minimize  f(x,y) =  3*(1-x).^2.*exp(-(x.^2) - (y+1).^2) ...
%                               - 10*(x/5 - x.^3 - y.^5).*exp(-x.^2-y.^2) ...
%                               - 1/3*exp(-(x+1).^2 - y.^2);
%             subject to
%                                x^2 + y   <=  6;
%                                x   + y^2 <=  6;
%
% Usage:  LMtestNLP1

% Remarks:
%  1) The following m-files are called.
%     - LMcbFDE1.m (required callback function to compute function values)
%     - LMcbGE1.m (optional callback function to compute partial derivatives)
%     - LMcbLP.m (optional callback function for multi-start solver)
%  2) Model is loaded via LSloadLPData() and LSloadNLPData() routines. See LINDO-API
%     manual for details.
%
% Copyright (c) 2001-2002
%
% LINDO Systems, Inc.            312.988.7422
% 1415 North Dayton St.          info@lindo.com
% Chicago, IL 60622              http://www.lindo.com
%
% Last update Sep 02, 2003 (MKA)
%

lindo;
clc;
help lmtestnlp1;
pause(0.01);

% Init model size
m  = 2;
n  = 2;
nz = 4;

% Init Bounds and RHS
csense = ['LL'];
b = [ 0  0 ]';
c = [ 0  0 ]';
l = [-3 -3 ]';
u = [+3 +3 ]';

% Init LP matrix
Abegcol = [ 0 2 4]';
Alencol = [ 2 2]';
Arowndx = [ 0 1 0 1]';
Acoef   = [ 0 1 1 0]';

% bounds

% Init NLP matrix mask
Nbegcol = [0 1 2]';
Nlencol = [1 1]';
Nrowndx = [0 1]';
Nobjndx = [0 1]';
Nobjcnt = 2;

% Set verbose
verbose = 1;

if verbose > 0 & 0,
   [x,y] = meshgrid(min(l):6/(40-1):max(u));
   z =  3*(1-x).^2.*exp(-(x.^2) - (y+1).^2) ...
       - 10*(x/5 - x.^3 - y.^5).*exp(-x.^2-y.^2) ...
```

```
       - 1/3*exp(-(x+1).^2 - y.^2);
    surf(x,y,z)
    axis([min(min(x)) max(max(x)) min(min(y)) max(max(y)) ...
          min(min(z)) max(max(z))])
    xlabel('X'), ylabel('Y'), title('A non-convex combination of Gaussian distributions
');
    fprintf('\n Press Enter to start Multi-Start Nonlinear Optimizer... \n\n');
    pause;
end;

% constant objective = 0
oshift = 0;

% Read license key from a license file
[MY_LICENSE_KEY,nErr] = mxlindo('LSloadLicenseString',MY_LICENSE_FILE);

% Create a LINDO environment
[iEnv,nErr]=mxlindo('LScreateEnv',MY_LICENSE_KEY);
if nErr ~= LSERR_NO_ERROR, return; end;

% Declare and create a model
[iModel,nErr]=mxlindo('LScreateModel',iEnv);
if nErr ~= LSERR_NO_ERROR, return; end;

% Set default log function (uncomment if necessary)
%[nErr] = mxlindo('LSsetLogfunc',iModel,'LScbLog','Dummy string');

% Set callback function to compute functional values (see LMcbFDE1.m)
[nErr] = mxlindo('LSsetFuncalc', iModel, 'LMcbFDE1', 'Dummy string');

% Set (optional) callback function to compute derivatives (see LMcbGE1.m)
%[nErr] = mxlindo('LSsetGradcalc', iModel, 'LMcbGE1', 'Dummy string');

% Set callback function to display local solutions (see LMcbLP.m)
if verbose > 0
   fprintf('\n%10s %15s %15s %15s %15s
\n','ITER','PRIMAL_OBJ','DUAL_OBJ','PRIMAL_INF','DUAL_INF');
   [nErr] = mxlindo('LSsetCallback', iModel, 'LMcbLP', 'Dummy string');
end;

% Set NLP print level to 1
[nErr] = mxlindo('LSsetModelIntParameter', iModel, LS_IPARAM_NLP_PRINTLEVEL, verbose);

%[nErr] = mxlindo('LSsetModelIntParameter', iModel, LS_IPARAM_NLP_CONOPT_VER, 2);

%[nErr] = mxlindo('LSsetModelIntParameter', iModel, LS_IPARAM_NLP_PRELEVEL, 126);

% Set NLP solver
[nErr] = mxlindo('LSsetModelIntParameter', iModel, LS_IPARAM_NLP_SOLVER,
LS_NMETHOD_MSW_GRG);


% Load the LP portion of the model
[nErr] = mxlindo('LSloadLPData', iModel, m, n, 1, 0, c, b, csense,...
   nz, Abegcol, Alencol, Acoef, Arowndx, l, u);

if nErr >  0,  return;  end;

% Load the NLP portion of the model
[nErr] = mxlindo('LSloadNLPData', iModel, Nbegcol, Nlencol,...
   [], Nrowndx, Nobjcnt,Nobjndx,[]);

if nErr >  0,  return;  end;

% Display model dimension
[n, nErr] = mxlindo('LSgetInfo',iModel,LS_IINFO_NUM_VARS);
[m, nErr] = mxlindo('LSgetInfo',iModel,LS_IINFO_NUM_CONS);
[LPNz, nErr] = mxlindo('LSgetInfo',iModel,LS_IINFO_NUM_NONZ);
```

```
[QCNz, nErr] = mxlindo('LSgetInfo',iModel,LS_IINFO_NUM_QC_NONZ);
[NLPNz, nErr] = mxlindo('LSgetInfo',iModel,LS_IINFO_NUM_NLP_NONZ);
[NLPobjNz, nErr] = mxlindo('LSgetInfo',iModel,LS_IINFO_NUM_NLPOBJ_NONZ);



% Optimize model
[solStatus,nErr] = mxlindo('LSoptimize', iModel, 0);

%[solStatus,nErr] = mxlindo('LSgetModelIntParameter',iModel,LS_IPARAM_STATUS);

if (solStatus == LS_STATUS_OPTIMAL | solStatus == LS_STATUS_LOCAL_OPTIMAL | solStatus ==
LS_STATUS_FEASIBLE)
   if verbose > 0
      fprintf('\n An (local) optimal solution is found ...  \n\n');
   end;


   % Get solution
   [x,nErr]=mxlindo('LSgetPrimalSolution',iModel);
   [obj,nErr]=mxlindo('LSgetInfo',iModel,LS_DINFO_POBJ);

   % Report
   fprintf(' f(X,Y) = %11.5f\n',obj);
   fprintf(' X      = %11.5f\n',x(1));
   fprintf(' Y      = %11.5f\n',x(2));
else
   if verbose > 0
      fprintf('\n Optimizer failed....  \n\n');
   end;
end;

% Test NLP data access routines
[aiBegcol,aiColcnt,adRowcoef,aiRowndx,nObjcnt,aiObjndx,...
      adObjcoef,adContype,nErr] = mxLINDO('LSgetNLPData',iModel);

[nNcnt,aiColndx,adColcoef,nErr] = mxLINDO('LSgetNLPConstraintDatai',iModel,0);
[nNcnt,aiColndx,adColcoef,nErr] = mxLINDO('LSgetNLPConstraintDatai',iModel,1);

[nNcnt,aiRowndx,adRowcoef,nErr] = mxLINDO('LSgetNLPVariableDataj',iModel,0);
[nNcnt,aiRowndx,adRowcoef,nErr] = mxLINDO('LSgetNLPVariableDataj',iModel,1);

[nNcnt,aiColndx,adObjcoef,nErr] = mxLINDO('LSgetNLPObjectiveData',iModel);

% Exit LINDO API
[nErr]=mxlindo('LSdeleteEnv',iEnv);
```