# Appendix G

# MATLAB programs

## G.1 isedm()

```
%Is real D a Euclidean Distance Matrix. -Jon Dattorro
%
%[Dclosest,X,isisnot,r] = isedm(D,tolerance,verbose,dimension,V)
%
%Returns: closest EDM in Schoenberg sense (default output),
%         a generating list X,
%         string 'is' or 'isnot' EDM,
%         actual affine dimension r of EDM output.
%Input: matrix D,
%       optional absolute numerical tolerance for EDM determination,
%       optional verbosity 'on' or 'off',
%       optional desired affine dim of generating list X output,
%       optional choice of 'Vn' auxiliary matrix (default) or 'V'.

function [Dclosest,X,isisnot,r] = isedm(D,tolerance_in,verbose,dim,V);

isisnot = 'is';
N = length(D);
```

```
if nargin < 2 | isempty(tolerance_in)
   tolerance_in = eps;
end
tolerance = max(tolerance_in, eps*N*norm(D));
if nargin < 3 | isempty(verbose)
   verbose = 'on';
end
if nargin < 5 | isempty(V)
   use = 'Vn';
else
   use = 'V';
end

%is empty
if N < 1
   if strcmp(verbose,'on'), disp('Input D is empty.'), end
   X = [ ];
   Dclosest = [ ];
   isisnot = 'isnot';
   r = [ ];
   return
end
%is square
if size(D,1) ~= size(D,2)
   if strcmp(verbose,'on'), disp('An EDM must be square.'), end
   X = [ ];
   Dclosest = [ ];
   isisnot = 'isnot';
   r = [ ];
   return
end
%is real
if ~isreal(D)
   if strcmp(verbose,'on'), disp('Because an EDM is real,'), end
   isisnot = 'isnot';
   D = real(D);
end
```

```
%is nonnegative
if sum(sum(chop(D,tolerance) < 0))
   isisnot = 'isnot';
   if strcmp(verbose,'on'), disp('Because an EDM is nonnegative,'),end
end
%is symmetric
if sum(sum(abs(chop((D - D')/2,tolerance)) > 0))
   isisnot = 'isnot';
   if strcmp(verbose,'on'), disp('Because an EDM is symmetric,'), end
   D = (D + D')/2;  %only required condition
end
%has zero diagonal
if sum(abs(diag(chop(D,tolerance))) > 0)
   isisnot = 'isnot';
   if strcmp(verbose,'on')
      disp('Because an EDM has zero main diagonal,')
   end
end
%is EDM
if strcmp(use,'Vn')
   VDV = -Vn(N)'*D*Vn(N);
else
   VDV = -Vm(N)'*D*Vm(N);
end
[Evecs Evals] = signeig(VDV);
if ~isempty(find(chop(diag(Evals),...
            max(tolerance_in,eps*N*normest(VDV))) < 0))
   isisnot = 'isnot';
   if strcmp(verbose,'on'), disp('Because -VDV < 0,'), end
end
if strcmp(verbose,'on')
   if strcmp(isisnot,'isnot')
      disp('matrix input is not EDM.')
   elseif tolerance_in == eps
      disp('Matrix input is EDM to machine precision.')
   else
      disp('Matrix input is EDM to specified tolerance.')
   end
```

```
end

%find generating list
r = max(find(chop(diag(Evals),...
        max(tolerance_in,eps*N*normest(VDV))) > 0));
if isempty(r)
   r = 0;
end
if nargin < 4 | isempty(dim)
   dim = r;
else
   dim = round(dim);
end
t = r;
r = min(r,dim);
if r == 0
   X = zeros(1,N);
else
   if strcmp(use,'Vn')
      X = [zeros(r,1) diag(sqrt(diag(Evals(1:r,1:r))))*Evecs(:,1:r)'];
   else
      X = [diag(sqrt(diag(Evals(1:r,1:r))))*Evecs(:,1:r)']/sqrt(2);
   end
end
if strcmp(isisnot,'isnot') | dim < t
   Dclosest = Dx(X);
else
   Dclosest = D;
end
```

### G.1.1 Subroutines for isedm()

#### G.1.1.1 chop()

```
%zeroing entries below specified absolute tolerance threshold
%-Jon Dattorro
function Y = chop(A,tolerance)

R = real(A);
I = imag(A);

if nargin == 1
   tolerance = max(size(A))*norm(A)*eps;
end
idR = find(abs(R) < tolerance);
idI = find(abs(I) < tolerance);

R(idR) = 0;
I(idI) = 0;

Y = R + i*I;
```

#### G.1.1.2 Vn()

```
function y = Vn(N)

y = [-ones(1,N-1);
      eye(N-1)]/sqrt(2);
```

#### G.1.1.3 Vm()

```
%returns EDM V matrix
function V = Vm(n)

V = [eye(n)-ones(n,n)/n];
```

#### G.1.1.4 signeig()

```
%Sorts signed real part of eigenvalues
%and applies sort to values and vectors.
%[Q, lam] = signeig(A)
%-Jon Dattorro

function [Q, lam] = signeig(A);

[q l] = eig(A);

lam = diag(l);
[junk id] = sort(real(lam));
id = id(length(id):-1:1);
lam = diag(lam(id));
Q = q(:,id);

if nargout < 2
   Q = diag(lam);
end
```

#### G.1.1.5 Dx()

```
%Make EDM from point list
function D = Dx(X)

[n,N] = size(X);
one = ones(N,1);

del = diag(X'*X);
D = del*one' + one*del' - 2*X'*X;
```

## G.2 conic independence, conici()

The recommended subroutine lp() (§G.2.1) is a linear program solver from MATLAB's *Optimization Toolbox* v2.0 (R11). Later releases of MATLAB replace lp() with linprog() that we find quite inferior to lp() on a wide range of problems.

Given an arbitrary set of directions, this c.i. subroutine removes the conically dependent members. Yet a conically independent set returned is not necessarily unique. In that case, if desired, the set returned may be altered by reordering the set input.

```
% Test for c.i. of arbitrary directions in rows or columns of X.
% -Jon Dattorro

function [Xci, indep_str, how_many_depend] = conici(X,rowORcol,tol);

if nargin < 3
   tol=max(size(X))*eps*norm(X);
end
if nargin < 2 | strcmp(rowORcol,'col')
   rowORcol = 'col';
   Xin = X;
elseif strcmp(rowORcol,'row')
   Xin = X';
else
   disp('Invalid rowORcol input.')
   return
end
[n, N] = size(Xin);

indep_str = 'conically independent';
how_many_depend = 0;
if rank(Xin) == N
   Xci = X;
   return
end
```

```
count = 1;
new_N = N;
%remove zero rows or columns
for i=1:N
   if chop(Xin(:,count),tol)==0
      how_many_depend = how_many_depend + 1;
      indep_str = 'conically Dependent';
      Xin(:,count) = [ ];
      new_N = new_N - 1;
   else
      count = count + 1;
   end
end
%remove conic dependencies
count = 1;
newer_N = new_N;
for i=1:new_N
   if newer_N > 1
      A = [Xin(:,1:count-1) Xin(:,count+1:newer_N); -eye(newer_N-1)];
      b = [Xin(:,count); zeros(newer_N-1,1)];
      [a, lambda, how] = lp(zeros(newer_N-1,1),A,b,[ ],[ ],[ ],n,-1);
      if ~strcmp(how,'infeasible')
         how_many_depend = how_many_depend + 1;
         indep_str = 'conically Dependent';
         Xin(:,count) = [ ];
         newer_N = newer_N - 1;
      else
         count = count + 1;
      end
   end
end
if strcmp(rowORcol,'col')
   Xci = Xin;
else
   Xci = Xin';
end
```

### G.2.1 lp()

```
LP     Linear programming.
X=LP(f,A,b) solves the linear programming problem:

             min f'x    subject to:   Ax <= b
              x

X=LP(f,A,b,VLB,VUB) defines a set of lower and upper
bounds on the design variables, X, so that the solution is
always in the range VLB <= X <= VUB.

X=LP(f,A,b,VLB,VUB,X0) sets the initial starting point to X0.

X=LP(f,A,b,VLB,VUB,X0,N) indicates that the first N constraints
defined by A and b are equality constraints.

X=LP(f,A,b,VLB,VUB,X0,N,DISPLAY) controls the level of warning
messages displayed.  Warning messages can be turned off with
DISPLAY = -1.

[X,LAMBDA]=LP(f,A,b) returns the set of Lagrangian multipliers,
LAMBDA, at the solution.

[X,LAMBDA,HOW] = LP(f,A,b) also returns a string how that
indicates error conditions at the final iteration.

LP produces warning messages when the solution is either
unbounded or infeasible.
```

## G.3 Map of the USA

### G.3.1 EDM, mapusa()

```
%Find map of USA using only distance information.
% -Jon Dattorro
%Reconstruction from EDM.
clear all;
close all;

load usalo; %From Matlab Mapping Toolbox
http://www-ccs.ucsd.edu/matlab/toolbox/map/usalo.html


%To speed-up execution (decimate map data), make
%'factor' bigger positive integer.
factor = 1;
Mg = 2*factor; %Relative decimation factors
Ms = factor;
Mu = 2*factor;

gtlakelat  = decimate(gtlakelat,Mg);
gtlakelon  = decimate(gtlakelon,Mg);
statelat   = decimate(statelat,Ms);
statelon   = decimate(statelon,Ms);
uslat      = decimate(uslat,Mu);
uslon      = decimate(uslon,Mu);

lat = [gtlakelat; statelat; uslat]*pi/180;
lon = [gtlakelon; statelon; uslon]*pi/180;
phi = pi/2 - lat;
theta = lon;
x = sin(phi).*cos(theta);
y = sin(phi).*sin(theta);
z = cos(phi);

%plot original data
plot3(x,y,z), axis equal, axis off
```

```
lengthNaN = length(lat);
id = find(isfinite(x));
X = [x(id)'; y(id)'; z(id)'];
N = length(X(1,:))

% Make the distance matrix
clear gtlakelat gtlakelon statelat statelon
clear factor x y z phi theta conus
clear uslat uslon Mg Ms Mu lat lon
D = diag(X'*X)*ones(1,N) + ones(N,1)*diag(X'*X)' - 2*X'*X;

%destroy input data
clear X

Vn = [-ones(1,N-1); speye(N-1)];
VDV = (-Vn'*D*Vn)/2;

clear D Vn
pack

[evec evals flag] = eigs(VDV, speye(size(VDV)), 10, 'LR');
if flag, disp('convergence problem'), return, end;
evals = real(diag(evals));

index = find(abs(evals) > eps*normest(VDV)*N);
n = sum(evals(index) > 0);
Xs = [zeros(n,1)  diag(sqrt(evals(index)))*evec(:,index)'];

warning off; Xsplot=zeros(3,lengthNaN)*(0/0); warning on;
Xsplot(:,id) = Xs;
figure(2)

%plot map found via EDM.
plot3(Xsplot(1,:), Xsplot(2,:), Xsplot(3,:))
axis equal, axis off
```

#### G.3.1.1 USA map input-data decimation, decimate()

```
function xd = decimate(x,m)
roll = 0;
rock = 1;
for i=1:length(x)
   if isnan(x(i))
      roll = 0;
      xd(rock) = x(i);
      rock=rock+1;
   else
      if ~mod(roll,m)
         xd(rock) = x(i);
         rock=rock+1;
      end
      roll=roll+1;
   end
end
xd = xd';
```

### G.3.2 EDM using ordinal data, omapusa()

```
%Find map of USA using ORDINAL distance information.
% -Jon Dattorro
clear all;
close all;

load usalo; %From Matlab Mapping Toolbox
http://www-ccs.ucsd.edu/matlab/toolbox/map/usalo.html

factor = 1;
Mg = 2*factor; %Relative decimation factors
Ms = factor;
Mu = 2*factor;

gtlakelat  = decimate(gtlakelat,Mg);
gtlakelon  = decimate(gtlakelon,Mg);
statelat   = decimate(statelat,Ms);
statelon   = decimate(statelon,Ms);
```

```
uslat      = decimate(uslat,Mu);
uslon      = decimate(uslon,Mu);

lat = [gtlakelat; statelat; uslat]*pi/180;
lon = [gtlakelon; statelon; uslon]*pi/180;
phi = pi/2 - lat;
theta = lon;
x = sin(phi).*cos(theta);
y = sin(phi).*sin(theta);
z = cos(phi);

%plot original data
plot3(x,y,z), axis equal, axis off

lengthNaN = length(lat);
id = find(isfinite(x));
X = [x(id)'; y(id)'; z(id)'];
N = length(X(1,:))

% Make the distance matrix
clear gtlakelat gtlakelon statelat
clear statelon state stateborder greatlakes
clear factor x y z phi theta conus
clear uslat uslon Mg Ms Mu lat lon
D = diag(X'*X)*ones(1,N) + ones(N,1)*diag(X'*X)' - 2*X'*X;

%ORDINAL MDS - vectorize D
count = 1;
M = (N*(N-1))/2;
f = zeros(M,1);
for j=2:N
   for i=1:j-1
         f(count) = D(i,j);
         count = count + 1;
   end
end
%sorted is f(idx)
[sorted idx] = sort(f);
```

```
clear D sorted X
f(idx)=((1:M).^2)/M^2;

%Create ordinal data matrix
O = zeros(N,N);
count = 1;
for j=2:N
   for i=1:j-1
         O(i,j) = f(count);
         O(j,i) = f(count);
         count = count+1;
   end
end

clear f idx

Vn = sparse([-ones(1,N-1); eye(N-1)]);
VOV = (-Vn'*O*Vn)/2;

clear O Vn
pack

[evec evals flag] = eigs(VOV, speye(size(VOV)), 10, 'LR');
if flag, disp('convergence problem'), return, end;
evals = real(diag(evals));

Xs = [zeros(3,1)  diag(sqrt(evals(1:3)))*evec(:,1:3)'];

warning off; Xsplot=zeros(3,lengthNaN)*(0/0); warning on;
Xsplot(:,id) = Xs;
figure(2)

%plot map found via Ordinal MDS.
plot3(Xsplot(1,:), Xsplot(2,:), Xsplot(3,:))
axis equal, axis off
```

## G.4 Rank reduction subroutine, RRf()

```
%Rank Reduction function -Jon Dattorro
%Inputs are:
%  Xstar matrix,
%  affine equality constraint matrix A whose rows are in svec format.
%
% Tolerance scheme needs revision...

function X = RRf(Xstar,A);
rand('seed',23);
m = size(A,1);
n = size(Xstar,1);
if size(Xstar,1)~=size(Xstar,2)
   disp('Rank Reduction subroutine: Xstar not square')
   pause
end
toler = norm(eig(Xstar))*size(Xstar,1)*1e-9;
if sum(chop(eig(Xstar),toler)<0) ~= 0
   disp('Rank Reduction subroutine: Xstar not PSD')
   pause
end
X = Xstar;
for i=1:n
   [v,d]=signeig(X);
   d(find(d<0))=0;
   rho = rank(d);
   for l=1:rho
      R(:,l,i)=sqrt(d(l,l))*v(:,l);
   end
   %find Zi
   svectRAR=zeros(m,rho*(rho+1)/2);
   cumu=0;
   for j=1:m
      temp = R(:,1:rho,i)'*svectinv(A(j,:))*R(:,1:rho,i);
      svectRAR(j,:) = svect(temp)';
      cumu = cumu + abs(temp);
   end
```

```
   %try to find sparsity pattern for Z_i
   tolerance = norm(X,'fro')*size(X,1)*1e-9;
   Ztem = zeros(rho,rho);
   pattern = find(chop(cumu,tolerance)==0);
   if isempty(pattern) %if no sparsity, do random projection
      ranp = svect(2*(rand(rho,rho)-0.5));
      Z(1:rho,1:rho,i)...
       =svectinv((eye(rho*(rho+1)/2)-pinv(svectRAR)*svectRAR)*ranp);
   else
      disp('sparsity pattern found')
      Ztem(pattern)=1;
      Z(1:rho,1:rho,i) = Ztem;
   end
   phiZ = 1;
   toler = norm(eig(Z(1:rho,1:rho,i)))*rho*1e-9;
   if sum(chop(eig(Z(1:rho,1:rho,i)),toler)<0) ~= 0
      phiZ = -1;
   end
   B(:,:,i) = -phiZ*R(:,1:rho,i)*Z(1:rho,1:rho,i)*R(:,1:rho,i)';
   %calculate t_i^*
   t(i) = max(phiZ*eig(Z(1:rho,1:rho,i)))^-1;
   tolerance = norm(X,'fro')*size(X,1)*1e-6;
   if chop(Z(1:rho,1:rho,i),tolerance)==zeros(rho,rho)
      break
   else
      X = X + t(i)*B(:,:,i);
   end
end
```

### G.4.1 svect()

```
%Map from symmetric matrix to vector
% -Jon Dattorro

function y = svect(Y,N)

if nargin == 1
   N=size(Y,1);
end

y = zeros(N*(N+1)/2,1);
count = 1;
for j=1:N
   for i=1:j
      if i~=j
         y(count) = sqrt(2)*Y(i,j);
      else
         y(count) = Y(i,j);
      end
      count = count + 1;
   end
end
```

### G.4.2 svectinv()

```
%convert vector into symmetric matrix.  m is dim of matrix.
% -Jon Dattorro
function A = svectinv(y)

m = round((sqrt(8*length(y)+1)-1)/2);
if length(y) ~= m*(m+1)/2
   disp('dimension error in svectinv()');
   pause
end

A = zeros(m,m);
count = 1;
for j=1:m
   for i=1:m
      if i<=j
         if i==j
            A(i,i) = y(count);
         else
            A(i,j) = y(count)/sqrt(2);
            A(j,i) = A(i,j);
         end
         count = count+1;
      end
   end
end
```

## G.5 Sturm's procedure

This is a demonstration program that can easily be transformed to a subroutine for decomposing positive semidefinite matrix $X$. Its attraction is a nonorthogonal alternative to eigen decomposition. (§A.7.5.0.1) That particular decomposition obtained is dependent on choice of matrix $A$.

```
%procedure to find dyad decomposition of X  -Jon Dattorro
clear all

N = 4;
r = 2;
X = 2*(ran(r,N)-0.5);
X = X'*X;

t = null(svect(X)');
A = svectinv(t(:,1));

%Suppose given matrix A is positive semidefinite (N=3, r=2)
%[v,d] = signeig(X);
%d(1,1)=0; d(2,2)=0; d(3,3)=pi;
%A = v*d*v';

tol = 1e-8;
Y = X;
y = zeros(size(X,1),r);
rho = r;
for k=1:r
    [v,d] = signeig(Y);
    v = v*sqrt(chop(d,1e-14));
    viol = 0;
    j = [ ];
    for i=2:rho
        if chop((v(:,1)'*A*v(:,1))*(v(:,i)'*A*v(:,i)),tol) ~= 0
            viol = 1;
        end
        if (v(:,1)'*A*v(:,1))*(v(:,i)'*A*v(:,i)) < 0
            j = i;
```

```
            break
        end
    end
    if ~viol
        y(:,k) = v(:,1);
    else
     alpha = (-2*(v(:,1)'*A*v(:,j)) + sqrt((2*v(:,1)'*A*v(:,j)).^2 ...
      -4*(v(:,j)'*A*v(:,j))*(v(:,1)'*A*v(:,1))))/(2*(v(:,j)'*A*v(:,j)));
    if chop(y(:,k)'*A*y(:,k),tol) ~= 0
     alpha = (-2*(v(:,1)'*A*v(:,j)) - sqrt((2*v(:,1)'*A*v(:,j)).^2 ...
      -4*(v(:,j)'*A*v(:,j))*(v(:,1)'*A*v(:,1))))/(2*(v(:,j)'*A*v(:,j)));
    end
     y(:,k) = (v(:,1) + alpha*v(:,j))/sqrt(1+alpha^2);
    end
    Y = Y - y(:,k)*y(:,k)';
    rho = rho - 1;
end

z = zeros(size(y));
e = zeros(N,N);
for i=1:r
    z(:,i) = y(:,i)/norm(y(:,i));
    e(i,i) = norm(y(:,i))^2;
end
lam = diag(e);
[junk id] = sort(real(lam));
id = id(length(id):-1:1);
z = [z(:,id(1:r)) null(z')]
e = diag(lam(id))
[v,d] = signeig(X)
X-z*e*z'
```