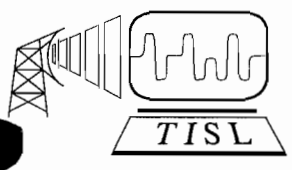
Beamform MATLAB Code

Shane M. Haas
David W. Petr

TISL Technical Report TISL-10920-03

January 1995



Telecommunications and Information Sciences Laboratory
The University of Kansas Center for Research, Inc.
2291 Irving Hill Drive Lawrence, Kansas 66045

Beamform MATLAB Code

Written by
Shane M. Haas

Supervisor
Dr. David Petr

Telecommunications and Information Sciences Laboratory
Department of Electrical Engineering and Computer Science
University of Kansas,
Lawrence, Kansas, USA 66045
Tel.: (913) 864-7757
email: shaas@tisl.ukans.edu

January 7, 1995

```
function GPS = AddRandPt(GPS,v)

%   ADDS A UNIFORM RANDOM PT BOUNDED BY v TO GPS
%   Syntax: GPS = AddRandPt(GPS,v) where v = [Xmin Xmax Ymin Ymax]

      %This function creates two uniformly distributed random
      %numbers on the interval specified boundaries in v

      %Input: GPS = Coordinate matrix
      %       v = [Xmin Xmax Ymin Ymax]
      %Output: GPS = New Coordinate Matrix

%   DEFINE VARIABLES

      SizeOfGPS = size(GPS);
      rand('seed',sum(clock*100));


%   GENERATE RANDOM X COORDINATE

      GPS(SizeOfGPS(1,1)+1,1) = v(1)+(v(2)-v(1))*rand(1);

%   GENERATE RANDOM Y COORDINATE

      GPS(SizeOfGPS(1,1)+1,2) = v(3)+(v(4)-v(3))*rand(1);

%   SORT MATRIX

      GPS = SortGPS(GPS);
```

```
function ClassNew = AddToClass(ClassOld,Add)

%    ADDS A GROUP OF STATES TO A CLASS
%    Syntax: ClassNew = AddToClass(ClassOld,Add)

        %This function appends ClassOld with the states
        %specified in Add.

        %Input: ClassOld = Current class needing appending (Row Vector)
        %       Add = States to add to ClassOld (Row Vector)
        %Output: ClassNew = Class containing all the states in ClassOld
        %                   and Add (Row Vector)

%    DEFINE VARIABLES

        OldSize = size(ClassOld);
        AddSize = size(Add);

%    APPEND ClassOld WITH THE STATES IN Add

        if AddSize(1,1) ~= 0
                ClassNew = ClassOld;
                ClassNew(1,OldSize(1,2)+1:OldSize(1,2)+AddSize(1,2)) = Add;
        else
                ClassNew = ClassOld;
        end %if%
```

```
function Next = AdjState(ProbMat,State)

%   RETURNS THE STATES WHICH State CAN COMMUNICATE DIRECTLY WITH
%   Syntax: Next = AdjState(ProbMat,State)

        %This function will determine which states can be reached
        %directly from the current state (State).  It does not
        %necessarily ensure 2 way communication between the two.

        %Input: ProbMat = Stochastic matrix of one step transition probabilities
        %       State = Current state of process
        %Output: Next = Row vector of adjacent states

%   DEFINE VARIABLES

        k = 0;
        SizeOfProbMat = size(ProbMat);

%   SEARCH State'S ROW AND DETERMINE ADJACENT STATES

        for i = 1:SizeOfProbMat(1,2)
                if ProbMat(State,i) > 0
                        k = k + 1;
                        Next(1,k) = i;
                end %if%
        end %for%
```

```
function C = Ang(p1,p2,p3)

%   CALCULATES ANGLE BETWEEN THREE PTS
%   Syntax: C = Ang(p1,p2,p3)    *** Radians ***

    %This function calculates the angle formed by the points p1, p2,
    %and p3 where p2 is the vertex.  Law of Cosines is applied.

    %Inputs: p1,p2,p3 = points where p2 is the vertex
    %Outputs: C = angle made by lines connecting the points

%   CALCULATE a, b, c (DISTANCES)

    a = Dist(p2,p3);
    b = Dist(p1,p2);
    c = Dist(p1,p3);

%   CALCULATE C

    C = acos((c.^2-a.^2-b.^2)./(-2*a.*b));
```

```
function CLTrue = AreCL(T,R,CLPropose)

%   DETERMINES IF A MATRIX OF PROPOSED LABELINGS ARE CONSISTENT WITH T AND R
%   Syntax: CLTrue = AreCL(T,R,CLPropose)

      %This function takes each proposed labeling in CLPropose
      %and tests it to determine if it is a consistent labeling
      %with respect to T and R.  Those labelings which pass
      %the test are add to CLTrue.

      %Input: T = Unit constraint relation
      %       R = Unit label constraint relation
      %       CLPropose = All labelings to be checked
      %Output: CLTrue = Those labelings from CLPropose which are
      %                  consistent labelings

%   DEFINE VARIABLES

      k = 0;
      SizeOfCLPropose = size(CLPropose);

%   SELECT AND TEST EACH LABELING

      for i = 1:SizeOfCLPropose(1,2)/2
              CL = SelectCL(CLPropose,i);
              X = IsCL(T,R,CL);
              if X
                      CLTrue = CLAdd(CLTrue,CL);
              end %if%
              Progress = i/(SizeOfCLPropose(1,2)/2)*100;
              clc
              fprintf('Checking Labelings...\n');
              fprintf('Progress: %1.2f\n',Progress);
      end %for%
```

```
function Mat = CL2Mat(CL,U,NetSize)

%   CONVERTS A CONSISTENT LABELING INTO A PSEUDO STOCHASTIC MATRIX
%   Syntax: Mat = CL2Mat(CL,U,NetSize)

    %This creates a matrix which shows the connectivity
    %of the network.  Ones represent a connection between
    %two nodes which have the numbers of the index values.
    %For example if Mat(2,4) = 1 then node 2 has a link
    %to node 4.

    %Input: CL = Consistent labeling
    %       NetSize = Number of nodes in the network
    %Output: Mat = Connectivity matrix

%   TRY EACH PAIR OF NODES AND DETERMINE HOW THEY COMMUNICATE

    for i = 1:NetSize
            for j = 1:NetSize
                    Link = Node2Link(i,j);
                    [X,Index] = IsElement(CL,Link);
                    Y = IsElement(U,Link);
                    if X
                            if CL(Index(1,1),2) == 0
                                    Mat(i,j) = 0;
                                    Mat(j,i) = 0;
                            else
                                    Mat(i,j) = 1;
                                    Mat(j,i) = 1;
                            end %if%
                    elseif Y
                            Mat(i,j) = 1;
                            Mat(j,i) = 1;
                    end %if%
            end %for%
    end %for%
```

```
function CLNew = CLAdd(CLOld,CL)

%   ADDS A CONSISTENT LABELING TO A CONSISTENT LABELING SOLUTION
%   Syntax: CLNew = CLAdd(CLOld,CL)

        %This function will add a found consistent labeling to
        %a matrix of consistent labeling previously found

        %Input: CLOld = Previously found consistent labelings
        %       CL = Consistent labeling to add
        %Output: CLNew = Combined matrix of CLOld and CL

%   DEFINE VARIABLES

        SizeOfCLOld = size(CLOld);
        SizeOfCL = size(CL);

%   APPEND CLOld WITH CL

        CLNew = CLOld;
        CLNew(1:SizeOfCL(1,1),SizeOfCLOld(1,2)+1:SizeOfCLOld(1,2)+2) = CL;
```

```
function CL = CLabel(U,R)

%   RETURNS A CONSISTENT LABELING OF A SINGLE VALUED R
%   Syntax: CL = CLabel(U,R)

      %This function will return a [unit label] solution given
      %an R which is single valued.  Single valued means that
      %for each unit in R there is only one label given to it
      %throughout R.

      %Input: U = Units
      %       R = Single valued unit label relation
      %Output: CL = Solution matrix consisting of units and their labels
      %             in a [Unit Label] format

%   DEFINE VARIABLES

      UList = UnitOfR(U,R);
      SizeOfUList = size(UList);

%   GET CORRESPONDING LABELING FOR EACH UNIT

      for i = 1:SizeOfUList(1,1)
              [X,Index] = IsElement(R,UList(i,1));
              CL(i,1) = UList(i,1);
              CL(i,2) = R(Index(1,1),Index(1,2)+1);
      end %for%
```

```
function List = CanAccess(ProbMat,State,List)

%   FINDS THE STATES THAT State CAN ACCESS OR REACH
%   Syntax: List = CanAccess(ProbMat,State,List)

        %This function finds all of the states which can be reached
        %both directly and indirectly from State.  This is a recursive
        %function.  Note: List is a ROW vector

        %Input: ProbMat = Stochastic matrix of one step transition prob
        %       State = Current state in search
        %       List = States which have been found to be in class
        %Output: List = States which have been found to be in class

%   DEFINE VARIABLES

        k = 0;
        Flag = 1;

%   ADD ADJACENT STATES TO List IF THEY ARE NOT ALREADY THERE

        NextState = AdjState(ProbMat,State);
        SizeOfNextState = size(NextState);
        for i = 1:SizeOfNextState(1,2)
                X = IsElement(List,NextState(1,i));
                if ~X
                        k = k + 1;
                        Temp(1,k) = NextState(1,i);
                end %if%
        end %for%
        NextState = Temp;
        List = AddToClass(List,NextState);

%   CHECK EACH NextState FOR STATES THEY CAN REACH

        if size(NextState) ~= 0
                while Flag
                        [s,NextState] = RemoveOneState(NextState);
                        SizeOfNextState = size(NextState);
                        if SizeOfNextState(1,2) == 0
                                Flag = 0;
                        end %if%
                        List = CanAccess(ProbMat,s,List);
                end %while%
        end %if%
```

```
function Class = CanComm(ProbMat,State)

%   DETERMINES WHICH STATES State CAN COMMUNICATE WITH
%   Syntax: Class = CanComm(ProbMat,State)

      %This function finds all of the states which State can
      %for two way communication with directly or indirectly.

      %Input: ProbMat = Stochastic matrix of one step probabilities
      %       State = State to consider
      %Output: List = List of states State can communicate with

%   WHO CAN State ACCESS?

      List = CanAccess(ProbMat,State,[]);
      SizeOfList = size(List);

%   CHECK EACH STATE State CAN REACH AND DETERMINE WHICH ONES CAN REACH State

      k = 0;
      for i = 1:SizeOfList(1,2)
            NewList = CanAccess(ProbMat,List(1,i),[]);
            X = IsElement(NewList,State);
            if X
                  k = k + 1;
                  Class(1,k) = List(1,i);
            end %if%
      end %for%
```

```
function ClearLinkGUI(l,hAllQuery)

%    CLEARS A LINK FROM THE QUERY CONTROL BOX
%    Syntax: ClearLinkGUI(l,hAllQuery)

        %Places null strings in all the text boxes of
        %the specified link

        %Input: l = Link to be cleared
        %       hAllQuery = Handles for objects in the box

%  CLEAR TEXT OBJECTS INVOLVING THE SPECIFIED LINK

        set(hAllQuery(25),'str','');
        if l == 1
                set(hAllQuery(13),'str','');
                set(hAllQuery(17),'str','');
                set(hAllQuery(14),'str','');
                set(hAllQuery(18),'str','');
                set(hAllQuery(22),'str','');
                set(hAllQuery(7),'str','');
        else
                set(hAllQuery(15),'str','');
                set(hAllQuery(19),'str','');
                set(hAllQuery(16),'str','');
                set(hAllQuery(20),'str','');
                set(hAllQuery(23),'str','');
                set(hAllQuery(8),'str','');
        end %if%
```

```
%   GUI FOR CONFIGURATION CONTROLS

%   CREATE BOX

        figure(hConfigBox);
        set(hConfigBox,'name','Configuration Controls');

%   LINK RADIUS

hAllConfig(1) = uicontrol('sty','text','str','Link Radius',...
'pos',[.3 .9 .25 .05],'unit','norm');
hAllConfig(2) = uicontrol('sty','edit','pos',[.6 .9 .1 .05],'unit',...
'norm','back',CMap1,'call','Config=SetConfigGUI(hAllConfig,1,DefConfig);');

%   FREQ MAX

hAllConfig(3) = uicontrol('sty','text','str','Max Frequencies',...
'pos',[.3 .8 .25 .05],'unit','norm');
hAllConfig(4) = uicontrol('sty','edit','pos',[.6 .8 .1 .05],'unit',...
'norm','back',CMap1,'call','Config=SetConfigGUI(hAllConfig,1,DefConfig);');

%   INTERFERENCE MULTIPLIER

hAllConfig(5) = uicontrol('sty','text','str','Interference Mult',...
'pos',[.3 .7 .25 .05],'unit','norm');
hAllConfig(6) = uicontrol('sty','edit','pos',[.6 .7 .1 .05],'unit',...
'norm','back',CMap1,'call','Config=SetConfigGUI(hAllConfig,1,DefConfig);');

%   BEAM WIDTH

hAllConfig(7) = uicontrol('sty','text','str','Beam Width',...
'pos',[.3 .6 .25 .05],'unit','norm');
hAllConfig(8) = uicontrol('sty','edit','pos',[.6 .6 .1 .05],'unit',...
'norm','back',CMap1,'call','Config=SetConfigGUI(hAllConfig,0,DefConfig);');

%   BEAM WIDTH SLIDER

hAllConfig(9) = uicontrol('sty','slide','pos',[.35 .5 .3 .060],...
'min',0,'max',360,'call','Config=SetConfigGUI(hAllConfig,1,DefConfig);',...
'units','norm','back',CMap1);
hAllConfig(10) = uicontrol('sty','text','pos',[.275 .5 .05 .05],...
'units','norm','str','0');
hAllConfig(11) = uicontrol('sty','text','pos',[.675 .5 .05 .05],...
'units','norm','str','360');

%   CREATE U

hAllConfig(12) = uicontrol('sty','push','pos',[.1 .375 .35 .05],...
'unit','norm','str','Make U','call','U = MakeU(GPS,Config(1))',...
'back',CMap1);

%   CREATE T

hAllConfig(13) = uicontrol('sty','push','pos',[.55 .375 .35 .05],...
'unit','norm','str','Make T','back',CMap1,...
'call','[T,X1,X2] = MakeT2(GPS,U,Config(3),Config(4),Config(5))');

%   CREATE L

hAllConfig(14) = uicontrol('sty','push','pos',[.1 .305 .35 .05],...
'unit','norm','str','Make L','call','L = MakeL2(Config(2))',...
'back',CMap1);

%   CREATE R

hAllConfig(15) = uicontrol('sty','push','pos',[.55 .305 .35 .05],...
```

```
'unit','norm','str','Make R','call','R = MakeR2(L,T,X1,X2)',...
'back',CMap1);

%   CREATE ULTR

hAllConfig(16) = uicontrol('sty','push','pos',[.1 .17 .8 .1],...
'unit','norm','str','Make [U,L,T,R] Model',...
'call','tic;[U,L,T,R,X1,X2] = MakeULTR2(GPS,Config);t1=toc','back',CMap1);

%   SOLVE MODEL

hAllConfig(17) = uicontrol('sty','push','pos',[.1 .05 .8 .1],...
'unit','norm','str','Find All Network Configurations','back',CMap1,...
'call','tic;s=size(GPS);NetTot = SolveModel([],T(1,1),U,L,T,R,s(1,1));SetTotGUI(h
```

```
function Link = DefineLinkGUI(GPS,l,hNodePlot,hAllQuery)

%   FUNCTION WHICH DISPLAYS QUERY INFORMATION FROM QueryBox.m
%   Syntax: Link = DefineLinkGUI(GPS,l,hNodePlot,hAllQuery)

    %This function displays various information about
    %the link chosen.  If another link has already been
    %chosen or defined then the angle between the two
    %is computed.

    %Input: GPS = Coordinate Matrix
    %       l = 1 or 2 depending on which botton pressed
    %       hNodePlot = Figure handle of plot to consider
    %       hAllQuery = Object handles of QueryBox

%    DISPLAY INDIVIDUAL LINK INFORMATION

    CurFig = gcf;
    figure(hNodePlot)
    n1 = gSelectNode(GPS);
    n2 = gSelectNode(GPS);
    Link = Node2Link(n1,n2);
    if l == 1
            %Display Link
            set(hAllQuery(7),'str',int2str(Link));
            %Display Node Coordinates
            [n1,n2] = Link2Node(Link);
            [x1,y1] = Node2Coord(GPS,n1);
            [x2,y2] = Node2Coord(GPS,n2);
            set(hAllQuery(13),'str',num2str(x1));
            set(hAllQuery(17),'str',num2str(y1));
            set(hAllQuery(14),'str',num2str(x2));
            set(hAllQuery(18),'str',num2str(y2));
            set(hAllQuery(22),'str',num2str(LinkLength(GPS,Link)));
    else
            %Display Link
            set(hAllQuery(8),'str',int2str(Link));
            %Display Node Coordinates
            [n1,n2] = Link2Node(Link);
            [x1,y1] = Node2Coord(GPS,n1);
            [x2,y2] = Node2Coord(GPS,n2);
            set(hAllQuery(15),'str',num2str(x1));
            set(hAllQuery(19),'str',num2str(y1));
            set(hAllQuery(16),'str',num2str(x2));
            set(hAllQuery(20),'str',num2str(y2));
            set(hAllQuery(23),'str',num2str(LinkLength(GPS,Link)));
    end %if%

%   IF LINK 1 AND 2 HAVE BEEN DEFINED...CALCULATE ANGLE

    l1 = str2num(get(hAllQuery(7),'str'));
    l2 = str2num(get(hAllQuery(8),'str'));
    if (l1 ~= []) & (l2 ~= [])
            set(hAllQuery(25),'str',num2str(Rad2Deg(LinkAngle(GPS,l1,l2))));
    end %if%
    figure(CurFig);
```

```
function Rad = Deg2Rad(Deg)

%    CONVERTS: DEG -> RAD
%    Syntax: Rad = Deg2Rad(Deg)

        %This function converts an angle which is in degrees to radians

        %Input:  Deg = Angle in degrees
        %Output: Rad = Converted angle in radians

%    CONVERT ANGLE

        Rad = Deg/180*pi;
```

```
function GPS = DeleteNode(GPS,n)

%   REMOVES A NODE FROM THE GPS MATRIX
%   Syntax: GPS = DeleteNode(GPS,n)

    %This function removes node n from the GPS matrix

    %Input: GPS = Coordinate Matrix
    %       n = Node to be removed (index)
    %Output: GPS = New Coordinate Matrix

%   DEFINE VARIABLES

    SizeOfGPS = size(GPS);

%   3 CASES: n == 1 OR 1 < n < SizeOfGPS(1,1) OR n == SizeOfGPS(1,1)

    if n == 1  %Remove 1st node
            GPS = GPS(2:SizeOfGPS(1,1),:);
    elseif n == SizeOfGPS(1,1)
            GPS = GPS(1:SizeOfGPS(1,1)-1,:);
    else
            Top = GPS(1:n-1,:);
            Bot = GPS(n+1:SizeOfGPS(1,1),:);
            GPS = Top;
            GPS(n:SizeOfGPS(1,1)-1,:) = Bot;
    end %if%
```

```
%   DISPLAY CONTROLS FOR VIEWING NETWORK CONFIGURATIONS

%   TOTAL CONFIGS FOUND

figure(hDisplayBox);
set(hDisplayBox,'name','Network Display Controls');
hAllDisplay(1) = uicontrol('sty','text','pos',[.15 .75 .5 .2],...
'str','Total Configurations Found','unit','norm');
hAllDisplay(2) = uicontrol('sty','text','pos',[.7 .75 .15 .2],'unit','norm',...
'back',CMap2);

%   SHOW ALL CONFIGS

hAllDisplay(3) = uicontrol('sty','push','pos',[.15 .425 .7 .25],'unit','norm',...
'str','Show All Configurations',...
'call','ShowConfigGUI(hAllDisplay,hDispPlot,GPS,U,NetTot,FMap2,1)',...
'back',CMap1);

%   SELECT CONFIGURATION

hAllDisplay(4) = uicontrol('sty','text','pos',[.15 .1 .45 .25],'unit','norm',...
'str','Selected Configuration');
hAllDisplay(5) = uicontrol('sty','edit','pos',[.65 .1 .2 .25],'unit','norm',...
'call','ShowConfigGUI(hAllDisplay,hDispPlot,GPS,U,NetTot,FMap2,0);','back',CMap1)
```

```
function d = Dist(pt1,pt2)

%   CALCULATES DISTANCE BETWEEN TWO PTS
%   Syntax: d = Dist(pt1,pt2)

    %This function calculates the distance between two points

    %Inputs: pt1 = Row vector containing the coordinates of pt 1
    %        pt2 = Row vector containing the coordinates of pt 2
    %Outputs: D = distance between pt1 and pt2


%   CALCULATE DISTANCE

    d = sqrt(sum((pt1-pt2).^2));
```

```
function h = DrawAllU(GPS,U)

%   DRAWS ALL THE LINKS IN U
%   Syntax: h = DrawAllU(GPS,U)

        %This function allows a quick and easy glance at the network
        %to check for connectivity.

        %Input: GPS = Coordinate matrix
        %       U = Column vector of links
        %Output: h = handles for each link

%   DEFINE VARIABLES

        SizeOfU = size(U);

%   DRAW EACH LINK IN U

        for i = 1:SizeOfU(1,1)
                h(i,1) = DrawLink(GPS,U(i,1),[1 1 1]);
        end %for%
```

```
function h = DrawLink(GPS,Link,Color)

%   DRAWS A LINE REPRESENTING A LINK IN A GIVEN COLOR
%   Syntax: h = DrawLink(GPS,Link,Color)

    %This function draws a line between two nodes representing
    %a link.

    %Inputs: GPS = Coordinates of nodes
    %        Link = Link to be drawn
    %        Color = Color of line
    %Output: h = Handle of Line

%   GET COORDINATES

    [n1,n2] = Link2Node(Link);

    x1 = GPS(n1,1);
    x2 = GPS(n2,1);
    y1 = GPS(n1,2);
    y2 = GPS(n2,2);

%   DRAW LINE

    h = line([x1,x2],[y1,y2],'Color',Color);
```

```
function h = DrawLink2(GPS,Link,Label,FMap2)

%   DRAWS SEND/RECEIVE LINES REPRESENTING A LINK IN A GIVEN COLOR
%   THE SEND BEAM (f1) IS ON TOP
%   Syntax: h = DrawLink2(GPS,Link,Label,FMap2)

        %This function draws a line between two nodes representing
        %a link.

        %Inputs: GPS = Coordinates of nodes
        %        Link = Link to be drawn
        %        Label = Frequency label of link
        %        FMap2 = Map of colors for frequencies
        %        Note: 1st row is for no link
        %              2nd row is for f1 ...
        %Output: h = Handle of Line

%   GET COORDINATES

        [n1,n2] = Link2Node(Link);

        x1 = GPS(n1,1);
        x2 = GPS(n2,1);
        y1 = GPS(n1,2);
        y2 = GPS(n2,2);

%   DEFINE OFFSETS

        v = axis;
        dy = (v(1,4)-v(1,3))/80;

%  DRAW EACH FREQUENCY

        [f1,f2] = Label2Freq(Label);
        h(1) = line([x1,x2],[y1+dy,y2+dy],'Color',FMap2(f1+1,:));
        h(2) = line([x1,x2],[y1-dy,y2-dy],'Color',FMap2(f2+1,:));
```

```
function NetHand = DrawNet(GPS,U,CL,FList,ColorMap)

%   DRAWS A NETWORK BASED ON A CONSISTENT LABELING SOLUTION
%   Syntax: NetHand = DrawNet(GPS,U,CL,FList,ColorMap)

    %This function draws a network using different color lines
    %representing links on send/receive frequency pairs

    %Input: GPS = Coordinate Matrix
    %       U = Units of ULTR model
    %       CL = Consistent labeling solution
    %       FList = Label list of frequency pairs
    %       ColorMap = Matrix of color maps for frequencies
    %Output: NetHand = [Link,LinkHandle] Matrix of line handles

%   PLOT GPS

    PlotGPS(GPS);

%   PLOT EACH LINK

    SizeOfU = size(U);
    for i = 1:SizeOfU(1,1)
            [InCL,Index] = IsElement(CL,U(i,1));
            if InCL
                    Index = Index(1,1);
                    Freq = CL(Index,2);
                    [X,FIndex] = IsElement(FList,Freq);
                    FIndex = FIndex(1,1);
                    Color = ColorMap(FIndex,:);
            else
                    Color = [1 1 1];
            end %if%
            NetHand(i,1) = U(i,1);
            NetHand(i,2) = DrawLink(GPS,U(i,1),Color);
    end %for%
```

```
function DrawNet(GPS,U,CL,FMap2)

%   DRAWS A NETWORK BASED ON A CONSISTENT LABELING SOLUTION WITH
%   SEND RECEIVE FREQUENCY LINES
%   Syntax: DrawNet2(GPS,U,CL,FMap2)

        %This function draws a network using different color lines
        %representing links on send/receive frequency pairs

        %Input: GPS = Coordinate Matrix
        %       U = Units of ULTR model
        %       CL = Consistent labeling solution
        %       FMap2 = Matrix of color maps for indiv frequencies

%   PLOT GPS

        PlotGPS(GPS);

%   PLOT EACH LINK

        SizeOfU = size(U);
        for i = 1:SizeOfU(1,1)
                [InCL,Index] = IsElement(CL,U(i,1));
                if InCL
                        Freq = CL(Index(1,1),2);
                        DrawLink2(GPS,U(i,1),Freq,FMap2);
                else
                        DrawLink(GPS,U(i,1),[1 1 1]);
                end %if%
        end %for%
```

```
function h = DrawPat(GPS,Link,Node,Length,Width,Map,Style)

%   DRAWS A BEAM PATTERN FOR A LINK
%   Syntax: h = DrawPat(GPS,Link,Node,Length,Width,Map,'Sytle')

    %This function draws the beam pattern for a link.  The radius
    %of this pattern is specified by Length, and the total
    %beam width is given by Width.

    %Input: GPS = Coordinate matrix
    %       Link = Link for pattern center
    %       Node = Pattern from this point
    %       Length = Radius of pattern
    %       Width = Beamwidth (Degrees)
    %       Map = Color of Lines
    %       Sytle = Sytle of line ( '-' = solid , '--' = dashed)
    %Output: h = Handles for both lines

%   DEFINE VARIABLES

    [n1,n2] = Link2Node(Link);
    if Node == n1  %Define: (x1,y1) as the coordinates of Node
            [x1,y1] = Node2Coord(GPS,n1);
            [x2,y2] = Node2Coord(GPS,n2);
    else
            [x1,y1] = Node2Coord(GPS,n2);
            [x2,y2] = Node2Coord(GPS,n1);
    end %if%
    l = LinkLength(GPS,Link);
    Theta = atan2((y2-y1),(x2-x1)); %Angle of elevation
    Tau = Deg2Rad(Width/2);         %Half the beamwidth

%   DRAW LINE 1

    x3 = Length*cos(Theta + Tau) + x1;
    y3 = Length*sin(Theta + Tau) + y1;
    h(1) = line([x1,x3],[y1,y3],'color',Map,'LineStyle',Style);

%   DRAW LINE 2

    x4 = Length*cos(Theta - Tau) + x1;
    y4 = Length*sin(Theta - Tau) + y1;
    h(2) = line([x1,x4],[y1,y4],'color',Map,'LineStyle',Style);
```

```
function d = FarDist(GPS,Link1,Link2)

%   CALCULATES THE DISTANCE BETWEEN FARTHEST NODES IN LINKS
%   Syntax: d = FarDist(GPS,Link1,Link2)

        %This function calculates the distance bewteen the node in
        %Link1 which is farthest from the the nodes in Link2

        %Input: GPS = Coordinate Matrix
        %       Link1, Link2 = Links to be considered
        %Output: d = distance

%   DEFINE VARIABLES

        [n_11,n_12] = Link2Node(Link1);
        [x_11,y_11] = Node2Coord(GPS,n_11);
        [x_12,y_12] = Node2Coord(GPS,n_12);
        [n_21,n_22] = Link2Node(Link2);
        [x_21,y_21] = Node2Coord(GPS,n_21);
        [x_22,y_22] = Node2Coord(GPS,n_22);

%   CALCULATE DISTANCES

        d11_21 = Dist([x_11,y_11],[x_21,y_21]);
        d11_22 = Dist([x_11,y_11],[x_22,y_22]);
        d12_21 = Dist([x_12,y_12],[x_21,y_21]);
        d12_22 = Dist([x_12,y_12],[x_22,y_22]);

        d = max([d11_21,d11_22,d12_21,d12_22]);
```

```
function Class = FindClass(ProbMat,State,Class)

%   FINDS THE CLASS WHICH HAS State AS A MEMBER
%   Syntax: Class = FindClass(ProbMat,State,Class)

    %This function finds all of the states which can be reached
    %both directly and indirectly from State.  This is a recursive
    %function.  Note: Class is a ROW vector

    %Input: ProbMat = Stochastic matrix of one step transition prob
    %       State = Current state in search
    %       Class = States which have been found to be in class
    %Output: Class = States which have been found to be in class

%   DEFINE VARIABLES

    k = 0;
    Flag = 1;

%   ADD ADJACENT STATES TO Class IF THEY ARE NOT ALREADY THERE

    NextState = AdjState(ProbMat,State);
    SizeOfNextState = size(NextState);
    for i = 1:SizeOfNextState(1,2)
            X = IsElement(Class,NextState(1,i));
            if ~X
                    k = k + 1;
                    Temp(1,k) = NextState(1,i);
            end %if%
    end %for%
    NextState = Temp;
    Class = AddToClass(Class,NextState);

%   CHECK EACH NextState FOR STATES THEY COMMUNICATE WITH

    if size(NextState) ~= 0
            while Flag
                    [s,NextState] = RemoveOneState(NextState);
                    SizeOfNextState = size(NextState);
                    if SizeOfNextState(1,2) == 0
                            Flag = 0;
                    end %if%
                    Class = FindClass(ProbMat,s,Class);
            end %while%
    end %if%
```

```
function [V,VList] = FindV(R,Fail)

%   FINDS THE UNIT WITH THE LEAST AMOUNT OF LABELS LEFT IN R
%   Syntax: [V,VList] = FindV(R,Fail)

      %This function will find the unit in R that has the
      %fewest possible permitted labelings left.  The
      %procedure uses Fail which is the output of the
      %IsSingle routine.

      %Input: R = Unit label constraint relation
      %       Fail = Units which have more than one label left
      %Output: V = Unit with the least number of labels left in R
      %        VList = Possible labels for V

%   DEFINE VARIABLES

      SizeOfFail = size(Fail);
      if SizeOfFail ~= 0
            V = Fail(1,1);
            [SizeOfVList,VList] = LabelLeft(R,Fail(1,1));


%   FIND EACH UNIT AND DETERMINE THE ONE WITH THE SMALLEST NUMBER OF LABELS

            for i = 1:SizeOfFail(1,1)
                  [N,List] = LabelLeft(R,Fail(i,1));
                  if N < SizeOfVList(1,1)
                        V = Fail(i,1);
                        VList = List;
                        SizeOfVList = N;
                  end %if%
            end %for%
      end %if%
```

```
function Color = Freq2Color(Freq,Map)

%   CONVERTS A FREQUENCY PAIR INTO THEIR CORRESPONDING COLOR REPRESENTATION
%   Syntax: Color = Freq2Color(Freq,Map)

    %This function will take a frequency and give it a color
    %based on Map.

    %Input: Freq = Index of frequency pair
    %       Map = Color maps of all frequency pairs with first row
    %             corresponding to the color of the first frequency pair.
    %Output: Color = Frequency pair color map

%   DETERMINE IF LINK IS NIL

    if Freq == 0
            Color = [0 0 0]
    else
            Color = Map(Freq,:)
    end %if%
```

```
function Label = Freq2Label(f1,f2)

%    CONVERTS: Frequencies -> Label
%    Syntax: Label = Freq2Label(f1,f2)

        %This function combines two frequencies into a label
        %in the form of n1777n2

        %Inputs: f1,f2 = Frequencies to be in link
        %Outputs: Label = Label formed

%    COMBINE n1,n2

        if (f1 == 0) & (f2 == 0)
                Label = 0;
        else
                Label = str2num([num2str(f1),'777',num2str(f2)]);
        end %if%
```

```
%   GUI INTERFACE FOR MANIPULATING NODE COORDINATES

%   MAKE PUSHBOTTON "Plot GPS" -> PlotGPS(GPS)

gure(hGPSBox);
t(hGPSBox,'name','GPS Coordinate Controls');
posPlot = [.1 .1 .8 .2];
hPlotGPS = uicontrol('sty','push','pos',posPlot,'unit','norm',...
'call','figure(hNodePlot);PlotGPS(GPS);','str','Plot GPS','background',...
[.5020 .5020 .5020]);

%   MAKE PUSHBOTTON "Add Node" -> GPS = gAddGPS(GPS,1)

figure(hGPSBox);
posAdd = [.1 .4 .35 .15];
hAdd = uicontrol('sty','push','pos',posAdd,'unit','norm',...
'call','figure(hNodePlot);GPS = gAddGPS(GPS,1);v=axis;PlotGPS(GPS);axis(v);ClearI
'back',[.5020 .5020 .5020]);

%   MAKE PUSHBOTTON "Add Node (Random)" -> AddRandPt(GPS,v)

figure(hGPSBox);
posAddR = [.55 .4 .35 .15];
hAddR = uicontrol('sty','push','pos',posAddR,'unit','norm',...
'call','figure(hNodePlot);v=axis;GPS=AddRandPt(GPS,v);PlotGPS(GPS);axis(v);ClearI
'str','Add Node (Random)','back',[.5020 .5020 .5020]);

%   MAKE PUSHBOTTON "Clear GPS" -> clear(GPS);axis([0 10 0 10])

figure(hGPSBox)
posClear = [.25 .8 .5 .15];
hClear = uicontrol('sty','push','pos',posClear,'unit','norm',...
all','figure(hNodePlot);clear GPS;clg;axis([0 10 0 10]);grid;ClearLinkGUI(1,hA]
tr','Clear GPS','back',[.5020 .5020 .5020]);


%   MAKE PUSHBOTTON "Delete Node" -> n=gSelectNode(GPS),GPS=DeleteNode(GPS,n)

figure(hGPSBox)
posDel1 = [.35 .65 .3 .1];
hDel1 = uicontrol('sty','push','pos',posDel1,'unit','norm',...
'call','figure(hNodePlot);n=gSelectNode(GPS);GPS=DeleteNode(GPS,n);ClearLinkGUI(1
'str','Delete Node','back',[.5020 .5020 .5020]);
```

```
function [X,Fail] = IS(U,R)

%   DETERMINES IF R IS A SINGLE VALUED R
%   Syntax: [X,Fail] = IS(R)

        %This function will return a 1 if R contains only one label
        %for each unit present.  If R is not a single labeling
        %then Fail will contain those units in R that have more than one
        %label left

        %Input: R = Unit label constraint relation
        %Output: X = Boolean value (Consistent labeling = 1)
        %        Fail = Those units which have more than one label
        %               left in R

%   DEFINE VARIABLES

        X = 1;
        k = 0;
        UList = UnitOfR(U,R);
        SizeOfUList = size(UList);

%   FOR EACH UNIT DETERMINE IF IT HAS ONLY ONE LABEL PRESENT IN R

        if SizeOfUList(1,1) ~= 0
                for i = 1:SizeOfUList(1,1)
                        [E,Index] = IsElement(R,UList(i,1));
                        SizeOfIndex = size(Index);
                        Label = R(Index(1,1),Index(1,2)+1);
                        for j = 1:SizeOfIndex(1,1)
                                LPrime = R(Index(j,1),Index(j,2)+1);
                                if Label ~= LPrime
                                        X = 0;
                                        Y = IsElement(Fail,UList(i,1));
                                        if ~Y
                                                k = k + 1;
                                                Fail(k,1) = UList(i,1);
                                        end %if%
                                end %if%
                        end %for%
                end %for%
        else
                X = 0;
        end %if%
```

```
function X = IntRange2(GPS,Link1,Link2,IMult,TransmitWidth,ReceiveWidth)

%   DETERMINES IF Link1 WILL POTENTIALLY INTERFERE WITH Link2
%   NEW AND IMPROVED IntRange.m
    Syntax: X = IntRange2(GPS,Link1,Link2,IMult,TransmitWidth,ReceiveWidth)
    ALL ANGLES IN DEGREES

        %Link1 will interfere with Link2 iff any transmitting pattern
        %from a node in Link1 can be received by a node in Link2.
        %This means that the receiving node is in the transmission
        %pattern and the transmitting node is in the receiving pattern.
        %A transmit pattern has a finite range directly proportional
        %to the link length (IMult*LinkLength(GPS,Link1)).  The receive
        %pattern has an infinite range.  In addition the angles
        %(in degrees) are the total angle of propagation.

%   DEFINE VARIABLES

        [N(1),N(2)] = Link2Node(Link1);
        [N(3),N(4)] = Link2Node(Link2);
        IntDist = LinkLength(GPS,Link1)*IMult;
        TAng = Deg2Rad(TransmitWidth/2);
        RAng = Deg2Rad(ReceiveWidth/2);
        k = 0;
        X = [0 0 0 0];

%   CHECK EACH NODE FROM Link1 TO SEE IF ITS TRANSMITTING PATTERN
%   CAN BE RECEIVED BY A NODE FROM Link2

        for i = 1:2
                Tn = N(i);
                for j = 3:4
                        k = k + 1;
                        Rn = N(j);
                        TestLink = Node2Link(Tn,Rn);
                        if Rn ~= Tn
                                % Is Rn close enough to Tn
                                TFlag1 = LinkLength(GPS,TestLink) <= IntDist;
                                % Can TPattern from Tn reach Rn
                                TFlag2 = LinkAngle(GPS,TestLink,Link1) <= TAng;
                                % Can RPattern from Rn reach Tn
                                RFlag = LinkAngle(GPS,TestLink,Link2) <= RAng;
                                if TFlag1 & TFlag2 & RFlag
                                        X(k) = 1;
                                else
                                        X(k) = 0;
                                end %if%
                        end %if%
                end %for%
        end %for%

%disp('X = [1w3   1w4    2w3    2w4]');
```

```
function X = IsCL(T,R,CL)

%   DETERMINES IF A LABELING IS A CONSISTENT LABELING WITH RESPECT TO T AND R
%   FOR A ULTR MODEL WITH N = 2
%   Syntax: X = IsCL(T,R,CL)

      %This function determines if the proposed labeling (CL) is
      %a consistent labeling with respect to T and R.  A consistent
      %labeling means that when the labeling is assigned to T the
      %2N-tuples formed are members of R

      %Input: T = Unit constraint relation
      %       R = Unit label constraint relation
      %       CL = Proposed labeling
      %Output: X = Boolean value (Consistent labeling = 1)

%   DEFINE VARIABLES

      X = 1;
      SizeOfT = size(T);

%   ASSIGN LABELS TO UNITS IN T AND CHECK FOR THEIR OCCURENCE IN R

      for i = 1:SizeOfT(1,1)
              u1 = T(i,1);
              u2 = T(i,2);
              [Nil,I1] = IsElement(CL,u1);
              [Nil,I2] = IsElement(CL,u2);
              l1 = CL(I1(1,1),2);
              l2 = CL(I2(1,1),2);
              Y = IsMember(R,[u1 l1 u2 l2]);
              y = IsMember(R,[u2 l2 u1 l1]);
              if ~(y|Y) & (u1~=u2)
                      X = 0;
              end %if%
      end %for%
```

```
function [x,Index] = IsElement(Matrix,Element)

%   DETERMINES WHETHER AN ELEMENT EXISTS IN A MATRIX
%   Syntax: [x,Index] = IsElement(Matrix,Element)

    %This function is used to determine whether an element exists
    %in any cell of the matrix

    %Input: Matrix = Matrix to be considered
    %       Element = Element to be searched for
    %Output: x = boolean value (1 -> Exist)
    %        Index = Indices of Element's locations in Matrix


%   SUBTRACT DESIRED VALUE AND THEN CHECK FOR ZERO ELEMENTS

    [Row,Col] = find(~(Matrix-Element));
    [x] = ~isempty(Row);
    if x
            Index(:,1) = Row;
            Index(:,2) = Col;
    end %if%
```

```
function [X,Index] = IsMember(Matrix,Member)

%   DETERMINES IF Member IS EXISTS AS A ROW IN Matrix
%   Syntax: [X,Index] = IsMember(Matrix,Member)

        %The function finds a set of elements within a row of a matrix.
        %A 1 is returned if Member is found within Matrix.

        %Input: Matrix = Matrix to be searched
        %       Member = Elements to be searched for
        %Output: X = Boolean value (1 -> Member is in Matrix)

%   SEARCH EVERY ROW FOR Member

        SizeOfMatrix = size(Matrix);
        SizeOfMember = size(Member);
        if Matrix ~= [] & Member ~= []
                for i = 1:SizeOfMember(1,2)
                        SubMat(:,i) = Matrix(:,i)-Member(1,i);
                end %for%
                X = any(find(all(~SubMat')));
                Index = find(all(~SubMat'));
        else
                X = 0;
        end %if%
```

```
function X = IsOneClass(CL,U,NetSize)

%   DETERMINES IF ALL OF THE NODES CAN COMMUNICATE WITH EACH OTHER
%   Syntax: X = IsOneClass(CL,U,NetSize)

      %This function converts a consistent labeling into a
      %connectivity matrix of ones and zeros.  It then
      %checks to see if all of the nodes are in communication
      %with each other.

%   DEFINE VARIABLES

      X = 0;
      PMat = CL2Mat(CL,U,NetSize);
      Class = FindClass(PMat,1,[]);
      SizeOfClass = size(Class);

%   ARE ALL NODES INCLUDED IN THE CLASS?

      if SizeOfClass(1,2) == NetSize
            X = 1;
      end %if%
```

```
function [X,Fail] = IsTInR(T,R)

%   DETERMINES IF ALL MEMBERS OF T ARE IN R
%   Syntax: [X,Fail] = IsTInR(T,R)

        %This function takes each member of T and searches
        %R to see if it is contained within.

        %Input: T = Unit Constraint Relation (N = 2)
        %       R = Unit Label Constraint Relation (N = 2)
        %Ouput: X = Boolean value (1 = T was found in R)
        %       Fail = Members of T not in R

%   DEFINE VARIABLES

        R2(:,1) = R(:,1);
        R2(:,2) = R(:,3);
        X = 1;
        k = 0;
        SizeOfT = size(T);

%   TAKE EACH MEMBER OF T AND SEARCH R

        for i = 1:SizeOfT(1,1)
                M = IsMember(R2,T(i,:));
                if ~M
                        k = k + 1;
                        Fail(k,:) = T(i,:);
                        X = 0;
                end %if%
        end %for%
```

```
function [X,Index] = IsULPair(R,Unit,Label)

%   DETERMINES IF THE UNIT LABEL PAIR OCCURS IN R
%   Syntax: [X,Index] = IsULPair(R,Unit,Label)

    %This function will determine if and where a unit label
    %pair occurs in R.

    %Input: Unit, Label = Pair to be searched for
    %       R = Unit Label Constraint Relation
    %Output: X = Boolean value (1 = Pair exists)
    %        Index = (Row,Col) Index of every occurence of the pair
    %                Note: The index is of the unit

%  DEFINE VARIABLES

    X = 0;
    k = 0;
    SizeOfR = size(R);

%  SEARCH EVERY ROW AND COLUMN FOR THE PAIR

    for i = 1:SizeOfR(1,1)                    %  Do for each row
        for j = 1:2:SizeOfR(1,2)              %  Do for every odd column
            if (R(i,j)==Unit) & (R(i,j+1)==Label)
                k = k + 1;
                X = 1;
                Index(k,1) = i;
                Index(k,2) = j;
            end %if%
        end %for%
    end %for%
```

```
function [f1,f2] = Label2Freq(Label)

%   CONVERTS: Label -> Frequencies
%   Syntax: [f1,f2] = Label2Freq(Label)

        %This function converts a label (format of 'freq 1' '777' 'freq 2')
        %back into its frequency components


%   GET FREQ 1

        if Label == 0
                f1 = '0';
        else
                Label = abs(int2str(Label));
                SizeOfLabel = size(Label);
                for i = 1:max(findstr(Label,'777'))-1
                        f1(1,i) = Label(1,i);
                end %for%
        end %if%

%   GET FREQ 2

        if Label == 0
                f2 = '0';
        else
                j = 0;
                for i = max(findstr(Label,'777'))+3:SizeOfLabel(1,2)
                        j = j + 1;
                        f2(1,j) = Label(1,i);
                end %for%
        end %if%

%   CONVERT NODES TO NUMBERS


        f1 = str2num(setstr(f1));
        f2 = str2num(setstr(f2));
```

```
function LabelGPS(GPS)

%   DISPLAYS INDEX LABELS ON NODES
%   Syntax: LabelGPS(GPS)

      %This function labels the nodes defined by GPS.  Labels are the
      %index of each node

      %Input: GPS = Coordinate Matrix

%   DEFINE VARIABLES

      x = GPS(:,1);
      y = GPS(:,2);
      SizeOfGPS = size(GPS); SizeOfGPS = SizeOfGPS(1,1);

%   LABEL NODES

      for i = 1:SizeOfGPS
            text(x(i)+max(x)/100,y(i),num2str(i));
      end %for%
```

```
function [N,List] = LabelLeft(R,Unit)

%   RETURNS A THE NUMBER OF LABELS LEFT AND A LIST OF THOSE IN R
%   Syntax: [N,List] = LabelLeft(R,Unit)

        %This function returns all the labels associated with a unit in
        %R as well as the number of labels left.

        %Input: R = Unit label constraint relation
        %       Unit = Unit to be considered
        %Output: N = Number of labels paired with Unit
        %        List = List of those labels paired with Unit

%   DEFINE VARIABLES

        N = 0;
        [X,Index] = IsElement(R,Unit);
        SizeOfIndex = size(Index);

%   SEARCH R FOR Unit AND RECORD THE LABEL

        for i = 1:SizeOfIndex(1,1)
                Label = R(Index(i,1),Index(i,2)+1);
                Y = IsElement(List,Label);
                if ~Y
                        N = N + 1;
                        List(N,1) = Label;
                end %if%
        end %for%
```

```
function [n1,n2] = Link2Node(Link)

%    CONVERTS: Link -> Node
%    Syntax: [n1,n2] = Link2Node(Link)

        %This function converts a link (format of 'node 1' '000' 'node 2')
        %back into its node components


%    GET NODE 1

        Link = abs(int2str(Link));
        SizeOfLink = size(Link);
        for i = 1:max(findstr(Link,'000'))-1
                n1(1,i) = Link(1,i);
        end %for%

%    GET NODE 2

        j = 0;
        for i = max(findstr(Link,'000'))+3:SizeOfLink(1,2)
                j = j + 1;
                n2(1,j) = Link(1,i);
        end %for%

%    CONVERT NODES TO NUMBERS

        n1 = str2num(setstr(n1));
        n2 = str2num(setstr(n2));
```

```
function Theta = LinkAngle(GPS,Link1,Link2)

%   CALCULATES THE ANGLE BETWEEN TWO LINKS
%   Syntax: Theta = LinkAngle(GPS,Link1,Link2)  *** Radians ***

      %This function determines the angle between two links

      %Inputs: GPS = Matrix of node coordinates
      %        Link1,Link2 = Links to be considered
      %Outputs: Theta = angle between two links

%   DEFINE VARIABLES

      [n_11,n_12] = Link2Node(Link1);
      [x_11,y_11] = Node2Coord(GPS,n_11);
      [x_12,y_12] = Node2Coord(GPS,n_12);
      [n_21,n_22] = Link2Node(Link2);
      [x_21,y_21] = Node2Coord(GPS,n_21);
      [x_22,y_22] = Node2Coord(GPS,n_22);
      [x_i,y_i] = LinkInter(GPS,Link1,Link2);

%   DETERMINE ANGLE

      if n_11 == n_21
            Theta = Ang([x_12,y_12],[x_11,y_11],[x_22,y_22]);
      elseif n_11 == n_22
            Theta = Ang([x_12,y_12],[x_11,y_11],[x_21,y_21]);
      elseif n_12 == n_21
            Theta = Ang([x_11,y_11],[x_12,y_12],[x_22,y_22]);
      elseif n_12 == n_22
            Theta = Ang([x_11,y_11],[x_12,y_12],[x_21,y_21]);
      elseif (x_i == Inf) & (y_i == Inf)  %Parallel links
            Theta = Inf;
      else
            Theta = Ang([x_11,y_11],[x_i,y_i],[x_21,y_21]);
      end %if%
```

```
function [x,y] = LinkInter(GPS,Link1,Link2)

%   FINDS THE PT OF INTERSECTION BETWEEN TWO NODES
%   Syntax: [x,y] = LinkInter(GPS,Link1,Link2)

    %This function finds the point of intersection of the lines
    %defined by Link1,Link2)

    %Input: GPS = Matrix of Coordinates
    %       Link1,Link2 = Links which define lines
    %Output: [x,y] =  Point of intersection

%   DEFINE VARIABLES

    [a1,b1] = LinkLine(GPS,Link1);
    [a2,b2] = LinkLine(GPS,Link2);

%   FIND INTERSECTION

    if a1 ~= a2     %Non Parallel Lines
            if abs(a1) == Inf       % Vertical Line
                    x = b1;
                    y = a2*x + b2;
            elseif abs(a2) == Inf   %Vertical Line
                    x = b2;
                    y = a1*x + b1;
            else                    %Non Vertical Line
                    x = (b2-b1)/(a1-a2);
                    y = a1*x + b1;
            end %if%
    else            % Parallel Lines
            x = Inf;
            y = Inf;
    end %if%
```

```
function d = LinkLength(GPS,Link)

%   RETURNS THE LENGTH OF A LINK
%   Syntax: d = LinkLength(GPS,Link)

        %This function calculates the distance between the two nodes
        %of the Link.

        %Input: GPS = Coordinat Matrix
        %       Link = Link to be considered
        %Output: d = Length of link

%  DEFINE VARIABLES

        [n1,n2] = Link2Node(Link);
        [x1,y1] = Node2Coord(GPS,n1);
        [x2,y2] = Node2Coord(GPS,n2);

%  CALCULATE DISTANCE

        d = Dist([x1,y1],[x2,y2]);
```

```
function [a,b] = LinkLine(GPS,Link)

%   CALCULATES THE COEFF OF THE LINE THRO LINK
%   Syntax: [a,b] = LinkLine(GPS,Link)

    %This function returns the coeffiecients of the link
    %that is formed by a link

    %Inputs: GPS = Coordinate Matrix
    %        Link = Link from which line is defined
    %Outputs: [a,b] = [Slope,Y-Intercept] or for a vertical line
    %                 [Inf,X-Intercept]

%   DEFINE VARIABLES

    [n1,n2] = Link2Node(Link);
    x1 = GPS(n1,1);
    y1 = GPS(n1,2);
    x2 = GPS(n2,1);
    y2 = GPS(n2,2);

%   CALCULATE SLOPE

    m = (y2-y1)/(x2-x1);

%   CALCULATE LINE

    if abs(m) ~= Inf
            a = [m];
            b = [(-m*x1 + y1)];
    else
            a = [m];
            b = [x1];
    end %if%
```

```
function [x,y] = LinkMid(GPS,Link)

%   CALCULATES THE MIDPOINT OF A LINK
%   Syntax: [x,y] = LinkMid(GPS,Link)

        %This function calculates the midpoint of a link

        %Input: GPS = Coordinate matrix
        %       Link = Link to be considered
        %Output: [x,y] = Coordinates of MidPoint


%   CALCULATE MIDPOINT

%   DEFINE VARIABLES

        [n1,n2] = Link2Node(Link);
        [x1,y1] = Node2Coord(GPS,n1);
        [x2,y2] = Node2Coord(GPS,n2);

%   CALCULATE

        x = (x1 + x2)/2;
        y = (y1 + y2)/2;
```

```
function [GPS,x,y] = MakeGPS(Size)

%   CREATES, SORTS, GRAPHS, AND LABELS RANDOM GPS COORDINATES
%   Syntax: [GPS,x,y] = MakeGPS(Size)

    %This function creates a matrix of size (Size) of random
    %coordinates.

%   MAKE COORDINATES

    GPS = 100*SortGPS(rand(Size));
    x = GPS(:,1);
    y = GPS(:,2);

%   PLOT AND LABEL NOBES

    plot(x,y,'*');
    axis('equal');
    axis('square');
    SizeX = size(x);
    for i = 1:SizeX(1,1);
            text(x(i)+1,y(i),num2str(i));
    end %for%
```

```
function L = MakeL2(MaxFreq)

%    CREATES A MATRIX OF LABELS FOR LINKS
%    NEW AND IMPROVED MakeL.m
%    Syntax: L = MakeL2(MaxFreq)

        %This function creates the label column vector (L) by permutating
        %the possible frequencies taking 2 at a time

        %Input: MaxFreq = Maximum Frequencies Permitted
        %Output: L = Label list of frequencies

%    INITIALIZE VARIABLES

        Flag = 1;
        k = 1;
        L(1,1) = 0;
%    CREATE L

        fprintf('**** MakeL2.m Started ****\n\n')
        for f1 = 1:MaxFreq
                for f2 = 1:MaxFreq
                        if f1 ~= f2
                                k = k + 1;
                                L(k,1) = Freq2Label(f1,f2);
                        end %if%
                end %for%
        end %for%
```

```
function R = MakeR2(L,T,X1,X2)

%    CREATES THE UNIT-LABEL CONSTRAINT R
%    NEW AND IMPROVED MakeR.m
     Syntax: R = MakeR2(L,T,X1,X2)

         %This function assigns allowable frequency labels to members
         %of T based on the constraining nodes in X (X is created by MakeT2.m)
         %If link 10002 and link 30004 operating on f1777f2 and
         %f3777f4 contain nodes which interfere, the
         %assignment of frequencies would go as follows.
         %If 1 interferes with 3 then f1 ~= f4
         %If 1 interferes with 4 then f1 ~= f3
         %If 2 interferes with 3 then f2 ~= f4
         %If 2 interferes with 4 then f2 ~= f3

         %Input: L = Set of Labels (Pairs in this case)
         %       T = Unit Constraint Relation
         %       X1,X2 = Explicit node constraints from MakeT2.m
         %Output: R = Unit-Label Constraint Relation

%    CREATE R

         fprintf('**** MakeR2.m Started ****\n\n')
         z = 0;
         SizeOfL = size(L);SizeOfL = SizeOfL(1,1);
         SizeOfT = size(T);SizeOfT = SizeOfT(1,1);

         for i = 1:SizeOfT
                 tic
                 for j = 1:SizeOfL
                         [f1 f2] = Label2Freq(L(j,1));
                         for k = 1:SizeOfL
                                 [f3 f4] = Label2Freq(L(k,1));
                                 %Restrict frequencies based
                                 %on nodes from link1 interfering
                                 %with nodes on link2
                                 I13 = ~((X1(i,1) > 0) & (f1 == f4));
                                 I14 = ~((X1(i,2) > 0) & (f1 == f3));
                                 I23 = ~((X1(i,3) > 0) & (f2 == f4));
                                 I24 = ~((X1(i,4) > 0) & (f2 == f3));
                                 C1 = (I13 & I14 & I23 & I24);
                                 %Restrict frequencies based on nodes
                                 %from link2 interfering
                                 %with nodes on link1
                                 I31 = ~((X2(i,1) > 0) & (f3 == f2));
                                 I32 = ~((X2(i,2) > 0) & (f3 == f1));
                                 I41 = ~((X2(i,3) > 0) & (f4 == f2));
                                 I42 = ~((X2(i,4) > 0) & (f4 == f1));
                                 C2 = (I31 & I32 & I41 & I42);
                                 if (C1 & C2) | (f1==0 & f2==0)
                                         z = z + 1;
                                         R(z,1) = T(i,1);
                                         R(z,2) = L(j,1);
                                         R(z,3) = T(i,2);
                                         R(z,4) = L(k,1);
                                 end %if%
                         end %for%
                 end %for%
                 fprintf('Progress: %3.2f percent\n',i/SizeOfT*100);
                 fprintf('Time per Outer Loop: %1.3f seconds\n',toc)
         end %for%
```

```
function [T,X1,X2] = MakeT2(GPS,U,IMult,TWidth,RWidth)

%   CREATES MATRIX OF UNIT CONSTRAINT RELATIONS
%   NEW AND IMPROVED MakeT.m
%   Syntax: [T,X1,X2] = MakeT2(GPS,U,IMult,TWidth,RWidth)
%   ALL ANGLES IN DEGREES

      %This function produces the matrix T which contains all of the
      %links that constrain each other.  Two links will constrain
      %each other iff they are potentially interfering.
      %Link1 will interfere with Link2 iff any transmitting pattern
      %from a node in Link1 can be received by a node in Link2.
      %This means that the receiving node is in the transmission
      %pattern and the transmitting node is in the receiving pattern.
      %A transmit pattern has a finite range directly proportional
      %to the link length (IMult*LinkLength(GPS,Link1)).  The receive
      %pattern has an infinite range.  In addition the angles
      %(in degrees) are the total angle of propagation.

      %Inputs: GPS = Coordinates of nodes (One row per node)
      %        U = Units or possible links
      %        IMult = The multiplier used to determine
      %                the interference radius.
      %        TWidth = Total bandwidth of transmitting pattern
      %        RWidth = Total bandwidth of receiving pattern
      %Outputs: T = Unit Constraint Relationship of (U,L,T,R) model
      %         X1 = Explicitly shows which nodes on Link 1
      %              constrain those nodes on Link 2
      %              (X1 = [1w3 1w4 2w3 2w4]) where Link1 = 10002
      %                                            Link2 = 30004
      %              and the member of T = [Link1 Link2]
      %         X2 = Explicitly shows which nodes on Link2 constrain
      %              those nodes on Link1. X2 is the same
      %              as above but now Link1 = 30004, Link2 = 10002

%   FIND ALL LINKS WHICH CONSTRAIN EACH OTHER

      fprintf('**** MakeT2.m Started ****\n\n');
      SizeOfU = size(U);SizeOfU = SizeOfU(1,1);
      k = 0;
      for i = 1:SizeOfU
              tic
              for j = (i+1):SizeOfU
                   %tic
                   if i~=j
                       I1 = IntRange2(GPS,U(i),U(j),IMult,TWidth,RWidth);
                       I2 = IntRange2(GPS,U(j),U(i),IMult,TWidth,RWidth);
                              %Rearrange I2 to match node pairs of I1
                              %and add
                       %I = I1 + 2*[I2(1),I2(3),I2(2),I2(4)];
                       M = ~(IsMember(T,[U(i),U(j)])|IsMember(T,[U(j),U(i)]));
                       if (max(I1) >= 1 | max(I2) >= 1) & M
                              k = k + 1;
                              X1(k,:) = I1;
                              X2(k,:) = I2;
                              T(k,1) = U(i);
                              T(k,2) = U(j);
                       end %if%
                      end %if%
                      %ILoop = toc;
                      %fprintf('Inner Loop Time %3.1f\n',ILoop);
                   end %for%
              Progress = i/SizeOfU*100;
              fprintf('MakeT Progress: %3.2f percent\n',Progress);
              OLoop = toc;
              fprintf('Time per Outer Loop: %1.3f seconds\n',OLoop);
```

```
end %for%
```

```
function U = MakeU(GPS,LRad)

%    CREATES COLUMN VECTOR OF UNITS
%    Syntax: U = MakeU(GPS,LRad)

        %This function produces the vector U which contains all of the
        %links which are possible given LRad.

        %Inputs: GPS = Coordinates of nodes (One row per node)
        %        LRad = Maximum Link Radius
        %Outputs: U = Unit Vecor of the (U,L,T,R) model

%    UNITS BASED ON LINK RADIUS

        fprintf('**** MakeU.m Started ****\n\n');
        SizeOfGPS = size(GPS);
        k = 0;
        for i = 1:SizeOfGPS(1,1)
                for j = i:SizeOfGPS(1,1)
                        if i ~= j
                                p1 = GPS(i,:);
                                p2 = GPS(j,:);
                                d = Dist(p1,p2);
                                if d <= LRad
                                        k = k + 1;
                                        U(k,1) = Node2Link(i,j);
                                end %if%
                        end %if%
                end %for%
        end %for%
```

```
function [U,L,T,R,X1,X2] = MakeULTR2(GPS,Config)

%   GENERATES THE HARALICK U,L,T,R MODEL
%   NEW AND IMPROVED MakeULTR.m
%   Syntax: [U,L,T,R,X1,X2] = MakeULTR2(GPS,Config)

    %This function generates a (U,L,T,R) Model as described by Haralick
    %in his paper "Consistent Labeling".  The model describes a radio
    %network in which the links interfere with each other

    %Input: GPS = Coordinate Matrix (Row Vector)
    %       Config = Configuration Matrix
    %                [LRad = Max linking radius
    %                 FreqMax = Maximum number of frequencies
    %                 IMult = Multiplier for determining interference rad
    %                 TWidth = Total beamwidth of transmitting pattern
    %                 RWidth = Total beamwidth of receiving pattern
    %Output: U = Set of units (Links)
    %        L = Set of labels for units (Frequencies)
    %        T = Unit constraint relation
    %        R = Unit-Label constrain relation

%   READ Config MATRIX

    clc;
    disp('Loading Configuration');
    LRad = Config(1,1);
    FreqMax = Config(1,2);
    IMult = Config(1,3);
    TWidth = Config(1,4);
    RWidth = Config(1,5);

    CREATE U

    disp('Creating U:  Units');
    U = MakeU(GPS,LRad);

%   CREATE L

    disp('Creating L:  Labels');
    L = MakeL2(FreqMax);

%   CREATE T

    clc;
    disp('Creating T:  Unit Constraint Relation');
    [T,X1,X2] = MakeT2(GPS,U,IMult,TWidth,RWidth);

%   CREATE R

    clc;
    disp('Creating R:  Unit-Label Constraint Relation');
    R = MakeR2(L,T,X1,X2);
```

```
function d = MidDist(GPS,Link1,Link2)

%   CALCULATE DISTANCE BETWEEN LINK MIDPOINTS
%   Syntax: d = MidDist(GPS,Link1,Link2)

        %This function returns the distance between the midpoints of
        %two links

        %Input: GPS = Matrix of Coordinates
        %       Link1,Link2 = Links to be considered
        %Output: d = Distance between midpoints

%   DEFINE VARIABLES

        [x1,y1] = LinkMid(GPS,Link1);
        [x2,y2] = LinkMid(GPS,Link2);

%   CALCULATE DISTANCE

        d = Dist([x1,y1],[x2,y2]);
```

```
function [x,y] = Node2Coord(GPS,Node)

%    CONVERTS: Node -> Coord
%    Syntax: [x,y] = Node2Coord(GPS,Node)

        %This function converts a the Node (index) into its GPS
        %coordinates

        %Input: GPS = Matrix of Coordinates
        %       Node = Index of desired node
        %Output: [x,y] = Coordinates of node

%    GET COORDINATES

        x = GPS(Node,1);
        y = GPS(Node,2);
```

```
function Out = Node2Link(n1,n2)

%    CONVERTS: Node -> Link
%    Syntax: Node2Link(n1,n2)

        %This function combines two node indices into a link
        %in the form of n1000n2

        %Inputs: n1,n2 = Nodes to be incorporated into link
        %Outputs: Out = Link formed from nodes

%    COMBINE n1,n2

        if n1 <= n2
                Out = str2num([num2str(n1),'000',num2str(n2)]);
        else
                Out = str2num([num2str(n2),'000',num2str(n1)]);
        end %if%
```

```
function d = PDist(Coeff,pt)

%   CALCULATES THE PERPENDICULAR DISTANCE BETWEEN LINE AND PT
%   Syntax: d = PDist(Coeff,pt)

      %This function finds the perpendicular distance from the line
      %y = ax + b and the point pt.

      %Inputs: Coeff = Coefficients of line
      %        pt = Point to be considered
      %Outputs: d = Perpendicular distance

%  DEFINE VARIABLES AND MATRICES

      xo = pt(1,1);
      yo = pt(1,2);
      a = Coeff(1,1);
      b = Coeff(1,2);

% FIND INTERSECTION OF LINE (y = ax + b) AND ITS PERP LINE THRO pt

      if a == 0                %horizonal line
            x = xo;
            y = a*x + b;
      elseif abs(a) == Inf            %vertical line
            x = b;
            y = yo;
      else
            x = (yo+1/a*xo-b)/(a+1/a);
            y = a*x+b;
      end %if%

%  CALCULATE DISTANCE BETWEEN [x,y] AND [xo,yo]

      d = Dist([x,y],pt);
```

```
function PlotGPS(GPS)

%   PLOTS AND LABELS THE NODES IN GPS
%   Syntax: PlotGPS(GPS)

        %This function plots the nodes defined by GPS.

        %Input: GPS = Coordinate Matrix

%   DEFINE VARIABLES

        x = GPS(:,1);
        y = GPS(:,2);

%   PLOT

        plot(x,y,'*')
        axis('square')
        axis('equal')

%   LABEL

        LabelGPS(GPS);
        title('Network of ATM Switch Nodes');...
        xlabel('Distance Units');...
        ylabel('Distance Units');...
        grid;
```

```
%   GUI FOR QUERY CONTROLS

%   "Select Link 1" PUSHBOTTON

CMap1 = [.5020 .5020 .5020];
CMap2 = [.85 .85 .85];

figure(hQueryBox)
set(hQueryBox,'name','Link Query Controls')

posSel1 = [.1 .85 .35 .1];
hAllQuery(1) = uicontrol('sty','push','pos',posSel1,'unit','norm',...
'call','Link1=SelectLinkGUI(GPS,U,1,hNodePlot,hAllQuery);ZoomBox',...
'str','Select Link 1','back',CMap1);

%   "Select Link 2" PUSHBOTTON

figure(hQueryBox)
posSel2 = [.55 .85 .35 .1];
hAllQuery(2) = uicontrol('sty','push','pos',posSel2,'unit','norm',...
'call','Link2=SelectLinkGUI(GPS,U,2,hNodePlot,hAllQuery);ZoomBox',...
'str','Select Link 2','back',CMap1);

%   "Define Link 1" PUSHBOTTON

figure(hQueryBox)
posDef1 = [.1 .7 .35 .1];
hAllQuery(3) = uicontrol('sty','push','pos',posDef1,'unit','norm',...
'call','Link1=DefineLinkGUI(GPS,1,hNodePlot,hAllQuery);ZoomBox',...
'str','Define Link 1','back',CMap1);

%   "Define Link 2" PUSHBOTTON

figure(hQueryBox)
posDef2 = [.55 .7 .35 .1];
hAllQuery(4) = uicontrol('sty','push','pos',posDef2,'unit','norm',...
'call','Link2=DefineLinkGUI(GPS,2,hNodePlot,hAllQuery);ZoomBox',...
'str','Define Link 2','back',CMap1);

%   TEXT DISPLAY "Link 1"

figure(hQueryBox)
posLink1Txt = [.18 .625 .2 .05];
hAllQuery(5) = uicontrol('sty','text','pos',posLink1Txt,'unit','norm',...
'str','Link 1');

%   TEXT DISPLAY "Link 2"

figure(hQueryBox)
posLink2Txt = [.62 .625 .2 .05];
hAllQuery(6) = uicontrol('sty','text','pos',posLink2Txt,'unit','norm',...
'str','Link 2');

%   TEXT DISPLAY of Link1

figure(hQueryBox)
posLink1Val = [.18 .55 .2 .05];
hAllQuery(7) = uicontrol('sty','text','pos',posLink1Val,'unit','norm',...
'back',CMap2);

%   TEXT DISPLAY of Link2

figure(hQueryBox)
posLink2Val = [.62 .55 .2 .05];
hAllQuery(8) = uicontrol('sty','text','pos',posLink2Val,'unit','norm',...
'back',CMap2);
```

```
%   NODE COORDINATES

figure(hQueryBox)
hAllQuery(9) = uicontrol('sty','text','pos',[.4 .45 .2 .065],'unit','norm',...
'str','Node Information');
hAllQuery(10) = uicontrol('sty','text','pos',[.225 .4 .05 .05],...
'unit','norm','str','1');
hAllQuery(11) = uicontrol('sty','text','pos',[.725 .4 .05 .05],...
'unit','norm','str','2');
hAllQuery(12) = uicontrol('sty','text','pos',[.45 .39 .1 .05],...
'unit','norm','str','[x;y]');
hAllQuery(13) = uicontrol('sty','text','pos',[.05 .325 .15 .05],...
'back',CMap2,'unit','norm');
hAllQuery(14) = uicontrol('sty','text','pos',[.3  .325 .15 .05],...
'back',CMap2,'unit','norm');
hAllQuery(17) = uicontrol('sty','text','pos',[.05 .265 .15 .05],...
'back',CMap2,'unit','norm');
hAllQuery(18) = uicontrol('sty','text','pos',[.3  .265 .15 .05],...
'back',CMap2,'unit','norm');
hAllQuery(15) = uicontrol('sty','text','pos',[.55 .325 .15 .05],...
'back',CMap2,'unit','norm');
hAllQuery(16) = uicontrol('sty','text','pos',[.8  .325 .15 .05],...
'back',CMap2,'unit','norm');
hAllQuery(19) = uicontrol('sty','text','pos',[.55 .265 .15 .05],...
'back',CMap2,'unit','norm');
hAllQuery(20) = uicontrol('sty','text','pos',[.8  .265 .15 .05],...
'back',CMap2,'unit','norm');

%   "LinkLength"

hAllQuery(21) = uicontrol('sty','text','pos',[.4 .175 .2 .065],'unit',...
'norm', 'str','Length');
hAllQuery(22) = uicontrol('sty','text','pos',[.15 .165 .2 .05],...
'back',CMap2,'unit','norm');
hAllQuery(23) = uicontrol('sty','text','pos',[.645 .165 .2 .05],...
'back',CMap2,'unit','norm');

%   "Angle"

hAllQuery(24) = uicontrol('sty','text','pos',[.4 .08 .2 .065],'unit','norm',...
'str','Angle (Deg)');
hAllQuery(25) = uicontrol('sty','text','pos',[.4 .01 .2 .05],...
'back',CMap2,'unit','norm');

%   "Clear Link 1"

hAllQuery(26) = uicontrol('sty','push','pos',[.1 .01 .15 .037],...
'unit','norm','call','ClearLinkGUI(1,hAllQuery);ZoomBox',...
'str','Clear Link 1',...
'back',CMap1);

%   "Clear Link 2"

hAllQuery(27) = uicontrol('sty','push','pos',[.75 .01 .15 .037],...
'unit','norm','call','ClearLinkGUI(2,hAllQuery);ZoomBox',...
'str','Clear Link 2',...
'back',CMap1);
```

```
%   MAIN CONTROL FOR RDRN

    %This m file loads a graphic user interface
    %for use with a consistent labeling algorithm for
    %frequency assignments for RDRN


%   SET FIGURE HANDLES

    hGPSBox = 1;
    hQueryBox = 2;
    hZoomBox = 3;
    hConfigBox = 4;
    hNodePlot = 5;
    hDisplayBox = 6;
    hDispPlot = 7;

%   DEFAULT CONFIGURATIONS

    DefConfig = [5 3 3 10 10];
    FMap = [0 0 0;1 0 0;0 1 0;0 0 1;1 1 0;1 0 1;0 1 1];
    FMap2 = [0 0 0;1 0 0;0 1 0;0 0 1];

%   BRING UP GPSBox

    figure(hGPSBox);...
    set(hGPSBox,'pos',[6 830 553 164]);
    GPSBox;

%   BRING UP QUERY BOX;

    figure(hQueryBox);...
    set(hQueryBox,'pos',[4 375 557 425]);
    QueryBox;

%   BRING UP ZOOM BOX

    figure(hZoomBox);
    set(hZoomBox,'pos',[828 42 440 383],...
    'name','Zoom Window of Chosen Links');

%   BRING UP CONFIG BOX

    figure(hConfigBox);...
    set(hConfigBox,'pos',[571 337 479 309]);
    ConfigBox;
    Config = SetConfigGUI(hAllConfig,0,DefConfig);

%   BRING UP MAIN NODE PLOT

    figure(hNodePlot);...
    set(hNodePlot,'pos',[575 578 560 420],'name','Main Plot of ATM Nodes');
    axis([0 10 0 10]);...
    grid

%   BRING UP DISPLAY CONTROLS

    figure(hDisplayBox);...
    set(hDisplayBox,'pos',[700   344   392    90]);...
    set(hDisplayBox,'name','Network Display Controls');...
    DisplayBox;

%   BRING UP NETWORK DISPLAY PLOT

    figure(hDispPlot);...
    set(hDispPlot,'pos',[658   466   438   341]);...
```

```
set(hDispPlot,'name','Network Display Plot');
```

```
function Deg = Rad2Deg(Rad)

%  CONVERTS: Radians -> Degrees
%  Syntax: Deg = Rad2Deg(Rad)

        %This function converts an angle which is in degrees to radians

        %Input:  Rad = Angle in radians
        %Output: Deg = Converted angle in degrees

%   CONVERT ANGLE

        Deg = Rad*180/pi;
```

```
function [l,NewList] = RemoveOneLabel(OldList)

%    REMOVES THE FIRST LABEL FROM A LIST AND RETURNS BOTH
%    Syntax: [l, NewList] = RemoveOneLabel(OldList)

     %This function will remove the first label from a list
     %and then return that label and the list minus the removed
     %label

     %Input: OldList = List of labels (Column Vector)
     %Output: l = First label from OldList
     %        NewList = OldList minus l (Column Vector)

%    DEFINE VARIABLES

     SizeOfOldList = size(OldList);

%    REMOVE 1st LABEL

     l = OldList(1,1);

%    GENERATE NEWLIST

     for i = 2:SizeOfOldList(1,1)
          NewList(i-1,1) = OldList(i,1);
     end %for%
```

```
function [s,NewList] = RemoveOneState(List)

%    REMOVES ONE STATE FROM A LIST
%    Syntax: [s,NewList] = RemoveOneState(List)

        %This function removes the first state of a row of states
        %and then returns the row minus the taken state.

        %Input: List = List of states (Row Vector)
        %Output: s = First state in List
        %         NewList = List minus s

%    DEFINE VARIABLES

        SizeOfList = size(List);

%    SELECT s

        s = List(1,1);

%    CREATE NewList

        for i = 2:SizeOfList(1,2)
                NewList(1,i-1) = List(1,i);
        end %for%
```

```
function [RPrime,Index] = RestrictR(R,Unit,Label)

%   RESTRICTS R TO THOSE 2N-TUPLES WHERE Unit HAS Label
%   Syntax: [RPrime,Index] = RestrictR(R,Unit,Label)

     %This function reduces the unit label constraint relation (R)
     %into RPrime.  RPrime contains those 2N-tuples where Unit has
     %Label.  It also has those 2N-tuples in which Unit does not
     %appear.

     %Input: R = Unit label constraint relation
     %       Unit, Label = Unit label pair that restricts R
     %Output: RPrime = Restricted R whose Unit has Label
     %        Index = Column vector containing the removed row index

%   DEFINE VARIABLES

     k = 0;
     SizeOfR = size(R);
     Flag = 1;

%   SEARCH R FOR THOSE 2N-TUPLES ALLOWED IN RPrime

     for i = 1:SizeOfR(1,1)                          %Do for every row
          for j = 1:2:SizeOfR(1,2)                   %Do for every odd column
               if R(i,j) == Unit
                    if R(i,j+1) == Label
                         Flag = 1;
                    else
                         Flag = 0;
                    end %if%
               end %if%
          end %for%
          if Flag                                    %If row meets criteria
               k = k + 1;
               RPrime(k,:) = R(i,:);
               Index(k,1) = i;
          end %if%
          Flag = 1;
     end %for%
```

```
function CL = SelectCL(CLFound,n)

%   SELECTS A CONSISTENT LABELING FROM ALL THE CONSISTENT LABELINGS FOUND
    Syntax: CL = SelectCL(CLFound,n)

        %This function selects the [Unit Label] consistent
        %labelings from those in CLFound.  CLFound consists
        %of a matrix in which labelings are stored in column pairs

        %Input: CLFound = All consistent labelings found
        %       n = Number of labeling to choose
        %Output: CL = Labeling chosen

%   CHOOSE CORRECT LABELING

        SizeOfCLFound = size(CLFound);
        Row = SizeOfCLFound(1,1);
        Col = SizeOfCLFound(1,2);
        if n*2 <= Col
                CL = CLFound(1:Row, (n*2-1):n*2);
        else
                CL = [NaN NaN];
        end %if%
```

```
function Link = SelectLinkGUI(GPS,U,l,hNodePlot,hAllQuery)

%   FUNCTION WHICH DISPLAYS QUERY INFORMATION FROM QueryBox.m
%   Syntax: Link = SelectLinkGUI(GPS,U,l,hNodePlot,hAllQuery)

        %This function displays various information about
        %the link chosen.  If another link has already been
        %chosen or defined then the angle between the two
        %is computed.

        %Input: GPS = Coordinate Matrix
        %       U = Column vectors of links
        %       l = 1 or 2 depending on which botton pressed
        %       hNodePlot = Figure handle of plot to consider
        %       hAllQuery = Object handles of QueryBox

%    DISPLAY INDIVIDUAL LINK INFORMATION

        CurFig = gcf;
        figure(hNodePlot)
        Link = gSelectLink(GPS,U);
        if l == 1
                %Display Link
                set(hAllQuery(7),'str',int2str(Link));
                %Display Node Coordinates
                [n1,n2] = Link2Node(Link);
                [x1,y1] = Node2Coord(GPS,n1);
                [x2,y2] = Node2Coord(GPS,n2);
                set(hAllQuery(13),'str',num2str(x1));
                set(hAllQuery(17),'str',num2str(y1));
                set(hAllQuery(14),'str',num2str(x2));
                set(hAllQuery(18),'str',num2str(y2));
                set(hAllQuery(22),'str',num2str(LinkLength(GPS,Link)));
        else
                %Display Link
                set(hAllQuery(8),'str',int2str(Link));
                %Display Node Coordinates
                [n1,n2] = Link2Node(Link);
                [x1,y1] = Node2Coord(GPS,n1);
                [x2,y2] = Node2Coord(GPS,n2);
                set(hAllQuery(15),'str',num2str(x1));
                set(hAllQuery(19),'str',num2str(y1));
                set(hAllQuery(16),'str',num2str(x2));
                set(hAllQuery(20),'str',num2str(y2));
                set(hAllQuery(23),'str',num2str(LinkLength(GPS,Link)));
        end %if%

%   IF LINK 1 AND 2 HAVE BEEN DEFINED...CALCULATE ANGLE

        l1 = str2num(get(hAllQuery(7),'str'));
        l2 = str2num(get(hAllQuery(8),'str'));
        if (l1 ~= []) & (l2 ~= [])
                set(hAllQuery(25),'str',num2str(Rad2Deg(LinkAngle(GPS,l1,l2))));
        end %if%
        figure(CurFig);
```

```
function Config = SetConfigGUI(hAllConfig,Slider,Default)

%    THIS FUNCTION TAKES THE VALUES FROM ConfigBox
%    AND WRITES THEM TO Config
%    Sytnax: SetConfigGUI(hAllConfig,Slider)
%    THE CONFIG VECTOR
%        [LRad,MaxFreq,IMult,TWidth,RWidth]

        %This function creates or updates the Config
        %matrix which has the configuation parameters.

        %Input: hAllConfig = Handles for the ConfigBox
        %       Slider = Boolean Value (1 = Take slider
        %                               value for beamwidth)
        %       Default = Default Values

%    LINK RADIUS

        Config = Default;
        LRad = str2num(get(hAllConfig(2),'str'));
        if LRad ~= []
                Config(1) = LRad;
        end %if%
        set(hAllConfig(2),'str',num2str(Config(1)));

%    MAX FREQUENCIES

        MaxFreq = str2num(get(hAllConfig(4),'str'));
        if MaxFreq ~= []
                Config(2) = MaxFreq;
        end %if%
        set(hAllConfig(4),'str',num2str(Config(2)));

    INTERFERENCE MULTIPLIER

        IMult = str2num(get(hAllConfig(6),'str'));
        if IMult ~= []
                Config(3) = IMult;
        end %if%
        set(hAllConfig(6),'str',num2str(Config(3)));

%    BEAM WIDTH

        if Slider
                BW = get(hAllConfig(9),'val');
                Config(4) = BW;
                Config(5) = BW;
        else
                BW = str2num(get(hAllConfig(8),'str'));
                if BW ~= []
                        Config(4) = BW;
                        Config(5) = BW;
                end %if%
        end %if%
        set(hAllConfig(9),'val',Config(4));
        set(hAllConfig(8),'str',num2str(Config(4)));
```

```
function SetTotGUI(hAllDisplay,NetTot)

%    THIS FUNCTION UPDATES THE TOTAL CONFIGS FOUND BOX
%    Syntax: SetTotGUI(hAllDisplay,NetTot)

        %Inputs: hAllDisplay = Handles from DisplayBox
        %        NetTot = Total Network Configuations

        Tot = size(NetTot)/2;
        set(hAllDisplay(2),'str',num2str(Tot(1,2)));
```

```
function ShowConfigGUI(hAllDisplay,hDispPlot,GPS,U,NetTot,FMap2,All)

%    THIS FUNCTION SHOWS NETWORK CONFIGURATIONS
%    Syntax: ShowConfigGUI(hAllDisplay,hDispPlot,GPS,U,NetTot,CMap,All)

        %This function allows the user to select viewing of all
        %or one network configurations in NetTot.

        %Input: hAllDisplay = Handles to the Display Box
        %       hDispPlot = Handle to the Network Display Plot
        %       GPS = Coordinate matrix
        %       U = Column vector of all links
        %       NetTot = Matrix of all network configurations
        %       FMap2 = Color map for each indiv frequency
        %       All = Switch to display all configurations

%  DISPLAY SINGLE CONFIG

        figure(hDispPlot);
        Tot = size(NetTot)/2;
        if ~All
                n = str2num(get(hAllDisplay(5),'string'));
                if n <= Tot(1,2)
                        CL = SelectCL(NetTot,n);
                        DrawNet2(GPS,U,CL,FMap2);
                end %if%
        else
                for n = 1:Tot(1,2)
                        CL = SelectCL(NetTot,n);
                        DrawNet2(GPS,U,CL,FMap2);
                        set(hAllDisplay(5),'string',num2str(n));
                        pause
                end %for%
        end %if%
```

```
function NetTot = SolveModel(NetTot,u,U,L,T,R,NetSize)

%   RECURSIVE TREE SEARCH ROUTINE TO FIND A Connected Network
%   Syntax: NetTot = SolveModel(NetTot,u,U,L,T,R,NetSize)

        %This function performs a tree search through R in order
        %to find all of the consistent labelings located within R.
        %If a single labeling is found to be consistent, it is checked
        %to see if it is a connected network.  If so, the CL is
        %recorded.  The branch of the tree terminates when a member
        %of T cannot find a consistent labeling within R.  It also
        %stops when RPrime is a single valued labeling.
        %This is a recursive procedure which keeps calling itself
        %until L = [].

        %Input: NetTot = Previous configurations found
        %       u = Unit to be considered
        %       U = All units of the model
        %       L = List of labels allowable for Unit (COLUMN VECTOR)
        %       T = Unit constraint relation
        %       R = Unit label constraint relation
        %       u = Unit which will restrict R into RPrime
        %Output: NetTot = Current list of all solutions found

%   DEFINE VARIABLES

        if L == [] | R == []
                Flag = 0;
        else
                Flag = 1;
        end %if%

%   STAY IN LOOP UNTIL SizeOfList = [0,0] OR A UNIT OF T IS ELIMINATED
%   FROM RPrime
%tic
        while Flag

                [l,L] = RemoveOneLabel(L);
                RPrime = RestrictR(R,u,l);
                [TInR,Fail] = IsTInR(T,RPrime);
                if (L == [])
                        Flag = 0;
                end %if%
                [S,Fail] = IS(U,RPrime);
%toc
                if S & (TInR)
                        CL = CLabel(U,RPrime);
                        C1 = IsOneClass(CL,U,NetSize);
                        if C1  %If the network is connected
                                NetTot = CLAdd(NetTot,CL);
                                s = size(NetTot);s = s(1,2)/2;
                                disp(['Configuration Found: ',num2str(s)]);
                        end %if%
%toc
                elseif (R ~= []) & (TInR)
                        [v,NewL] = FindV(RPrime,Fail);
%toc
%disp('Calling Myself again')
                        NetTot = SolveModel(NetTot,v,U,NewL,T,RPrime,NetSize);
                end %if%
        end %while%
```

```
function GPSOut = SortGPS(GPSIn)

%   SORTS THE GPS MATRIX IN ASCENDING X VALUES
%   Syntax: GPSOut = SortGPS(GPSIn)

        %This function sorts the GPS coordinates based on ascending
        %x value

        %Inputs: GPSIn = Coordinate Matrix (one node per row)
        %Outputs: GPSOut = Sorted Matrix

%  SORT GPS

        SizeOfGPS = size(GPSIn);
        if SizeOfGPS(1,1) == 1
                GPSOut = GPSIn;
        else
                [GPSOut,i] = sort(GPSIn);
                for j = 1:SizeOfGPS(1,1)
                        GPSOut(j,:) = GPSIn(i(j),:);
                end %for%
        end %if%
```

```
%  TRIAL 1: 20 Node Network

%  LOAD GPS DATA

       load data/20Node

%  MAKE MODEL

       tic;
       [U,L,T,R,X1,X2] = MakeULTR2(GPS,Config);
       Model_Time = toc;

%  SOLVE MODEL

       tic;
       NetTot = SolveModel([],T(1,1),U,L,T,R,20);
       TS_Time = toc;

%  SAVE DATA

       save data/20Node

%  EXIT

       exit
```

```
function [List] = UnitOfR(U,R)

%    RETURNS A LIST OF UNITS IN R
%    Syntax: [List] = UnitsOfR(U,R)

     %This function returns a list of those units which are
     %constrained in R.

     %Input: R = Unit label constraint relation
        %    U = All possible units
     %Output: List = List of units in R

%    DEFINE VARIABLES

     k = 0;
     SizeOfU = size(U);

%    SEARCH R TO GATHER LIST

         for i = 1:SizeOfU(1,1)
                X = IsElement(R,U(i,1));
                 if X
                         k = k + 1;
                         List(k,1) = U(i,1);
                 end %if%
         end %for%
```

```
function List = UnitOfT(T)

%    RETURNS A LIST OF UNITS IN T
%    Syntax: List = UnitsOfT(T)

     %This function returns a list of those units which are
     %constrained in T.

     %Input: T = Unit constraint relation
     %Output: List = List of units in T

%    DEFINE VATIABLES

     k = 0;
     SizeOfT = size(T);

%    SEARCH T TO GATHER LIST

     for i = 1:SizeOfT(1,1)
          for j = 1:SizeOfT(1,2)
               X = IsElement(List,T(i,j));
               if ~X
                    k = k + 1;
                    List(k,1) = T(i,j);
               end %if%
          end %for%
     end %for%
```

```
%   ZOOM WINDOW FOR SHOWING PAIRS OF LINKS

%   SET AXIS

    figure(hZoomBox)
    set(hZoomBox,'name','Zoom Window of Chosen Links');
    PlotGPS(GPS);
    title('Plot of Beam Patterns')
    l1 = str2num(get(hAllQuery(7),'str'));
    if l1 ~= []
        hl1 = DrawLink(GPS,l1,[1 0 0]);
        set(hl1,'LineWidth',3);
    end %if%
    l2 = str2num(get(hAllQuery(8),'str'));
    if l2 ~= []
        hl2 = DrawLink(GPS,l2,[0 0 1]);
        set(hl2,'LineWidth',3);
    end %if%

    [n11,n12] = Link2Node(l1);
    [n21,n22] = Link2Node(l2);
    [x11,y11] = Node2Coord(GPS,n11);
    [x12,y12] = Node2Coord(GPS,n12);
    [x21,y21] = Node2Coord(GPS,n21);
    [x22,y22] = Node2Coord(GPS,n22);
    Xmax = max([x11 x12 x21 x22]);
    Xmin = min([x11 x12 x21 x22]);
    Ymax = max([y11 y12 y21 y22]);
    Ymin = min([y11 y12 y21 y22]);

    %if Xmax ~= [] & Xmin ~= [] & Ymax ~= [] & Ymin ~= [] & (Xmin~=Xmax)
    %       axis([Xmin,Xmax,Ymin,Ymax]);
    %else
    %       axis('normal')
    %end %if%

%   DRAW PATTERNS

    IMult = Config(3);
    Width = Config(4);
    IntL1 = IMult*LinkLength(GPS,l1);
    IntL2 = IMult*LinkLength(GPS,l2);

    if l1 ~= []
        hPat(1,:) = DrawPat(GPS,l1,n11,IntL1,Width,[1 0 0],'--');
        hPat(2,:) = DrawPat(GPS,l1,n12,IntL1,Width,[1 0 0],'--');
    end %if%
    if l2 ~=[]
        hPat(3,:) = DrawPat(GPS,l2,n21,IntL2,Width,[0 0 1],'-.');
        hPat(4,:) = DrawPat(GPS,l2,n22,IntL2,Width,[0 0 1],'-.');
    end %if%

    axis('auto')
```

```
function GPSNew = gAddGPS(GPSOld,n)

%    ADDS COORDINATES TO GPS VIA THE MOUSE
%    Syntax: GPSNew = gAddGPS(GPSOld,n)

        %This function adds a new coordinate to GPS by allowing
        %the user to specify it using the mouse

        %Input: GPSOld = Coordinate matrix to be added on
        %       n = Number of coordinates to add
        %Output: GPSNew = Coordinate matrix with added value

%    DEFINE VARIABLES

        if ~exist('GPSOld')
                GPSOld = [];
        end %if%
        SizeOfGPS = size(GPSOld);

%    GET NEW COORDINATE

        [x,y] = ginput(n);

%    ADD TO GPS AND SORT

        for i = 1:n
                GPSOld(SizeOfGPS(1,1)+i,:) = [x(i),y(i)];
        end %for%
        GPSNew = SortGPS(GPSOld);
```

```
function Link = gSelectLink(GPS,U)

%    SELECTS THE NEAREST LINK TO THE MOUSE CLICK
%    Syntax: Link = gSelectLink(GPS,U)

        %This function finds the link whose perpendicular
        %distance from the mouse click is least.

        %Input: GPS = Coordinate matrix
        %       U = Column vector of links
        %Output: Link = Nearest link

%    GET COORDINATE

        [x,y] = ginput(1);

%    FIND CLOSEST LINK

        dMin = inf;
        SizeOfU = size(U);
        for i = 1:SizeOfU(1,1)
                [n1,n2] = Link2Node(U(i,1));
                [x1,y1] = Node2Coord(GPS,n1);
                [x2,y2] = Node2Coord(GPS,n2);
                xm = (x1+x2)/2;
                ym = (y1+y2)/2;
                d = Dist([x,y],[xm,ym]);
                if d < dMin
                        Link = U(i,1);
                        dMin = d;
                end %if%
        end %for%
```

```
function [Node,NCoord] = gSelectNode(GPS)

%   GRAPHICALLY SELECTS THE NODE CLOSEST TO THE MOUSE CLICK
%   Syntax: [Node,NCoord] = gSelectNode(GPS)

        %This node gets a ginput from the figure and returns
        %the node closest to it

        %Input: GPS = Coordinate matrix
        %Output: Node = Closest node to point selected

%  GET INPUT FROM MOUSE

        [x,y] = ginput(1);

%  DETERMINE WHICH NODE IS CLOSEST

        DMin = inf;
        SizeOfGPS = size(GPS);
        for i = 1:SizeOfGPS(1,1)
                d = Dist([x,y],GPS(i,:));
                if d < DMin
                        DMin = d;
                        Node = i;
                        NCoord = GPS(i,:);
                end %if%
        end %for%
```