# Contents

# 1 ./AssignmentToIndex.m

```
% AssignmentToIndex Convert assignment to index.
%
%   I = AssignmentToIndex(A, D) converts an assignment, A, over variables
%   with cardinality D to an index into the .val vector for a factor.
%   If A is a matrix then the function converts each row of A to an index.
%
%   See also IndexToAssignment.m
%
% Copyright (C) Daphne Koller, Stanford University, 2012

function I = AssignmentToIndex(A, D)

D = D(:)'; % ensure that D is a row vector
if (any(size(A) == 1)),
    I = cumprod([1, D(1:end - 1)]) * (A(:) - 1) + 1;
else
    I = sum(repmat(cumprod([1, D(1:end - 1)]), size(A, 1), 1) .* (A - 1), 2) + 1;
end;

end
```

# 2 ./CalculateExpectedUtilityFactor.m

```
% Copyright (C) Daphne Koller, Stanford University, 2012

function EUF = CalculateExpectedUtilityFactor( I )

  % Inputs: An influence diagram I with a single decision node and a single utility node.
  %         I.RandomFactors = list of factors for each random variable.  These are CPDs, with
  %              the child variable = D.var(1)
  %         I.DecisionFactors = factor for the decision node.
  %         I.UtilityFactors = list of factors representing conditional utilities.
  % Return value: A factor over the scope of the decision rule D from I that
  % gives the conditional utility given each assignment for D.var
  %
  % Note - We assume I has a single decision node and utility node.
  EUF = [];
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %
  % YOUR CODE HERE...
  %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  RF = I.RandomFactors;
  D = I.DecisionFactors;
  U = I.UtilityFactors;
  PaD = D.var(1:end);
  allV = unique([RF(:).var]);
  diff = setdiff(allV, PaD);

  F = RF(1);
  for i=2:length(RF)
      F = FactorProduct(F, RF(i));
  end;
  F = FactorProduct(F, U);
  EUF = FactorMarginalization(F, diff);




end
```

# 3 ./CompareData.m

```
function retval = CompareData(data1, data2, context, Params)
% function retval = comparedata(data1, data2, context, Params)
%   compares to see if data1 and data2 are roughly recursively equal. "Rough" here is defined by
%   the Params.  Note that matlabs ISEQUAL function is a test for exact equality.  comparedata
%   will print out intermediate results wheras ISEQUAL just returns 0 or 1.
% Output Arg
%   retval = 0 for not roughly equal 1 for roughly equal.
% Input Args: - Note context and Params are optional and have defaults as specified below.
%   data1, data2 - objects to be compared - arbitrarily nested structurs, cell arrays, numeric arrays.
%       Anything else gets compared with ISEQUAL.
%   context('Top') - This is current context string of the recursion.  Example: if the user calls
%       comparedata with an empty context or ommits it all together the top level context will
```

```
%       be set to "Top".  If then, for example, data1 and data2 are structures and data1 has
%       a "Fred" field but data2 doesn't the program will print out:
%       "Mismatch in Top.Fred not found in second structure.
%   Params - structure
%       outfileorfid(1) - Place to print progress. Either name of a file or a file handle open for
%           writing text. Default of 1 is handle 1 which means the screen.  String implies path
%           name.
%           Note that if there is a mismatch a message with the start word "Mismatch" will be output
%           so as to facilitate file searching.
%       displaycontextprogress(1) - outputs current context string at each new level of recursion
%           e.g Top.a.b.c would mean that data1.a.b.c is being compared to data2.a.b.c
%           Note that if turned on this gets written out to the screen and to the location specified
%           to outfileorfid (obviously if outfileorfid is 1 by user input or default it's not
%           output twice).
%       NumericTolerance(1e-10) - When comparing numeric arrays (at any level of comparison recursion)
%           equality will be determined by whether all elements of the two items have an
%           absolute value difference less than this tolerance. That is:
%           max(abs(Array1(:) - Array2(:))) < NumericTolerance
%           1e-10 should be good enough for practical equality for real life being small enough
%               to be beyond the accuracy of data yet account for typical numeric error growth
%      ignoreunmatchedfieldnames(0) - if at some level of recursion two structures are being compared
%           such that one has fields that the other doesn't have 0 will mean that the comparedata can
%           still return 1 (i.e. rough equality) if the common fields are recursively roughly equal.
%           A value of 0, the default, means that unique fields automatically disqualify rough equality.
%           Note that in any event, the field differences will be output.
%       showMinMaxAbsDiff(1) - show min/max/abs while comparing numeric arrays
% Notes:
%   If at some level of recursive comparison the rough equality test fails the
%   function will continue.  It will continue comparing as best it can.  This means
%   comparing just the common fields of structures even if some fields are not in common
%   but array elements (e.g. cell arrays or structur arrays) will not be compared (and
%   so not recursively compared if their size doesn't match.
%
%   Things that are defined as mismatches (in order):
%   If the type of the two objects differ.  This is determined by Matlab's CLASS function.  This means
%       that numeric arrays although numerically equal can fail if they have different numeric suptypes.
%       Example retval=comparedata(int16([1]), double([1]) fails because data1 is a int16 and data2 is
%       a double although they are numerically equal.  This could be rectified but not today.
%   size(data1) ~= size(data2) - Everything's an array internally and so all must have the same no. of
%       elements.
%   Unique fields in a structure - if ignoreunmatchedfieldnames is non-zero then this is not a mismatch.
%   Numeric arrays whose elements differ beyond tolerance as defined by NumericTolerance
%   isequal fails
%
%   Examples
%   >> retval=comparedata(int16([1]), double([1]))
%   context = Top
%   Top not the same data type int16 vs. double
%   retval =0
%
%   >> s1.a=1; s1.b=2; s1.c=3;          s2.a=1; s2.b=2;
%   >> retval=comparedata(s1, s2, [], struct('ignoreunmatchedfieldnames', 1))
%   context = Top
%   Mismatch in Top.c not found in second structure
%   context = Top.a
%   context = Top.b
%   retval =1
%
%   Comparing RR and RG structures output answers to comparedataRRRG.txt and comparing numeric arrays
%   with a tolerance of 1e-5.  Context will show top level as Fred.  Output is long so select pieces are below:
%   >> retval=comparedata(RR, RG, 'Fred', struct('outfileorfid', 'comparedataRRRG.txt', 'NumericTolerance',1e-5))
    ;
%   context = Fred
%   Mismatch in Fred.RepDataMR not found in second structure
%   Mismatch in Fred.fred not found in second structure
%   ....
%   context = Fred.grn
%   Mismatch in Fred.grn. Two objects not the same array size: 0 0 vs 1 49777
%   ...
%   Mismatch : at Fred.x numeric array comparison - abs(data 1 - data 2) > tolerance:
%       mindiff = -2115.180411 at [1, 45967]
%       maxdiff = 1880.794556 at [1, 45451]
%       max abs diff = 2115.180411 at [1, 45967]
%   context = Fred.y
%   Mismatch : at Fred.y numeric array comparison - abs(data 1 - data 2) > tolerance:
%       mindiff = -1882.577620 at [1, 44844]
%       maxdiff = 1667.421048 at [1, 26426]
%       max abs diff = 1882.577620 at [1, 44844]
%   --------------------------------
%   Author: Andrew Diamond of EnVision Systems LLC, Svyatoslav Zarutskiy
```

```
    defaultparams = struct('outfileorfid', 1, ...
                        'displaycontextprogress',1, ...
                        'NumericTolerance', 1e-10,...
                        'ignoreunmatchedfieldnames',0, ...
                        'showMinMaxAbsDiff', 1);

    if(~exist('Params', 'var'))
        Params = [];
    end

    Params = mergedefaultparams(Params, defaultparams);

    if(length(Params.outfileorfid) == 1 && isnumeric(Params.outfileorfid))
        Params.fid = Params.outfileorfid;
    elseif(~isempty(Params.outfileorfid) && ischar(Params.outfileorfid))
        [Params.fid, message] = fopen(Params.outfileorfid, 'wt');
        if(Params.fid < 3)
            error('Failed open file %s for reason %s', Params.outfileorfid, message);
        end
    end

    if(~exist('context', 'var') || isempty(context))
        context = 'Top';
    end

    retval = comparedatarecurse(data1, data2, context, Params);
end

function retval = comparedatarecurse(data1, data2, context, Params)
  persistent ParamsP;
  persistent iskindequalP;
  if(exist('Params', 'var'))
    iskindequalP = 1;
    ParamsP = Params;
  end

  if(ParamsP.displaycontextprogress)
    if(ParamsP.fid ~= 1)
      fprintf(1,'context = %s\n', context);
    end
    fprintf(ParamsP.fid,'context = %s\n', context);
  end

  if(~strcmp(class(data1), class(data2)))
    iskindequalP = 0;
    fprintf(ParamsP.fid,'%s not the same data type %s vs. %s\n',context, class(data1), class(data2));
  elseif(any(size(data1) ~= size(data2)))
    iskindequalP = 0;
    fprintf(ParamsP.fid,'Mismatch in %s. Two objects have different array sizes: ',context);
    fprintf(ParamsP.fid,'[%s] vs [%s]\n', num2str(size(data1)), num2str(size(data2)) );
  elseif(isstruct(data1))
    names1 = fieldnames(data1);
    names2 = fieldnames(data2);
    names1s = sort(names1);
    names2s = sort(names2);
    matchinds = zeros(1, min(length(names1s), length(names2s)));
    imatchind = 0;

    for inames1s = 1:length(names1s)
      if(isempty(strcmp(names1s{inames1s}, names2s)))
        fprintf(ParamsP.fid, ...
            'Mismatch in %s.%s not found in second structure\n', context, names1s{inames1s});
        if(~ParamsP.ignoreunmatchedfieldnames)
          iskindequalP = 0;
        end
      else
        imatchind = imatchind + 1;
         matchinds(imatchind) = inames1s;
      end
    end

    for inames2s = 1:length(names2s)
      if(isempty(strcmp(names2s{inames2s}, names1s)))
        fprintf(ParamsP.fid, ...
            'Mismatch in %s.%s not found in first structure\n', context, names2s{inames2s});
        if(~ParamsP.ignoreunmatchedfieldnames)
          iskindequalP = 0;
        end
      end
```

```
  end

  if (numel(data1) > 1)
    for iElt = 1:length(data1(:))
      [~, ind2subvretstr]=ind2subv(size(data1), iElt);
       comparestruct(data1(iElt), data2(iElt), names1s, ...
           matchinds(1:imatchind), sprintf('%s[%s]', context, ind2subvretstr));
    end
  else
    comparestruct(data1, data2, names1s, matchinds(1:imatchind), context);
  end

elseif(iscell(data1))
  for iElt=1:length(data1(:))
    [~, ind2subvretstr]=ind2subv(size(data1), iElt);
    comparedatarecurse(data1{iElt}, data2{iElt}, sprintf('%s{%d}', context, ind2subvretstr));
  end

elseif (isnumeric(data1))
  diff = data1(:) - data2(:);
  [mindiff, mindiffi] = min(diff);
  [maxdiff, maxdiffi] = max(diff);
  [maxabsdiff, maxabsdiffi] = max(abs(diff));

  if (maxabsdiff > ParamsP.NumericTolerance)
    if (numel(data1) == 1) % scalar
      fprintf(ParamsP.fid,...
          'Mismatch : at %s numeric scalar comparison - abs(data 1 - data 2) > tolerance of %g:\n', ...
          context, ParamsP.NumericTolerance);
      fprintf(ParamsP.fid, ...
          'Scalar 1 = %e, Scalar 2 = %e, Scalar1 - Scalar2 = %e\n', ...
          data1, data2, data1-data2);
    else
      fprintf(ParamsP.fid, ...
          'Mismatch : at %s numeric array comparison - abs(data 1 - data 2) > tolerance of %g:\n', ...
          context,ParamsP.NumericTolerance);

        fprintf(ParamsP.fid, '    Found: [ %s ]\n', num2str(data1, '%.3f '));
        fprintf(ParamsP.fid, 'Expected: [ %s ]\n', num2str(data2, '%.3f '));

        if ParamsP.showMinMaxAbsDiff == 1
          [~, ind2subvretstr] = ind2subv(size(data1), mindiffi);
          fprintf(ParamsP.fid,'    mindiff = %e at [%s]\n', mindiff, ind2subvretstr);
          [~, ind2subvretstr] = ind2subv(size(data1), maxdiffi);
          fprintf(ParamsP.fid,'    maxdiff = %e at [%s]\n', maxdiff, ind2subvretstr);
          [~, ind2subvretstr] = ind2subv(size(data1), maxabsdiffi);
          fprintf(ParamsP.fid,'    max abs diff = %e at [%s]\n', maxabsdiff, ind2subvretstr);
        end
     end
     iskindequalP = 0;
  end

elseif (islogical(data1))
  if ~isequal(data1, data2)
    if (numel(data1) == 1) % scalar
      fprintf(ParamsP.fid,...
          'Mismatch : at %s bool scalar comparison - bools are not equal:\n', ...
          context);
      fprintf(ParamsP.fid, 'Bool found = %s, Bool expected = %s\n', mat2str(data1), mat2str(data2));
    else
      fprintf(ParamsP.fid, ...
          'Mismatch : at %s bool array comparison - arrays are not equal:\n', ...
          context);

        fprintf(ParamsP.fid, '    Found: [ %s ]\n', mat2str(data1));
        fprintf(ParamsP.fid, 'Expected: [ %s ]\n', mat2str(data2));
     end
     iskindequalP = 0;
  end

elseif (~isequal(data1, data2))
  fprintf(ParamsP.fid, ...
      'Mismatch in %s. Two non-numeric, non-cell, non-structure arrays are not equal \n', ...
      context);
  iskindequalP = 0;
end

if(exist('Params', 'var'))
  if(ischar(ParamsP.outfileorfid))
    fclose(ParamsP.fid);
```

```
    end
    retval = iskindequalP;
  end
end

function comparestruct(data1, data2, names, matchinds, context)
    for imatchnames = 1:length(matchinds)
        namei = names{matchinds(imatchnames)};
        comparedatarecurse(data1.(namei), data2.(namei), sprintf('%s.%s',context,namei));
    end
end

function [ind2subvret, ind2subvretstr] = ind2subv(arraysize, ind1d)
    retstring = '[';
    for k=1:length(arraysize)-1
        retstring = sprintf('%sI%d,␣', retstring,k);
    end
    if(~isempty(arraysize))
        retstring = sprintf('%sI%d', retstring,length(arraysize));
    end

    retstring = sprintf('%s]', retstring);
    evalstring = [retstring, '=␣ind2sub(', 'arraysize,', num2str(ind1d), ');'];
    eval(evalstring);
    ind2subvret = eval(retstring);
    ind2subvretstr = sprintf('%d,␣',ind2subvret);
    commas = strfind(ind2subvretstr,','); % for backward compaitibility to 6.0, etc. find(ind2subvretstr == ',');

    if(~isempty(commas))
        ind2subvretstr = ind2subvretstr(1:commas(end)-1);
    end
end

function params = mergedefaultparams(params, defaultparams)
    if(isempty(params))
        params = defaultparams;
        return;
    end
    names = fieldnames(defaultparams);
    for iname=1:length(names)
        namei = names{iname};
        if(~isfield(params,namei)) % add the default
            params.(namei) = defaultparams.(namei);
        elseif( isstruct( defaultparams.(namei) ) )
            params.(namei) = mergedefaultparams( params.(namei), defaultparams.(namei) );
        end
    end
end
```

# 4 ./CPDFromFactor.m

```
% Copyright (C) Daphne Koller, Stanford University, 2012

function [CPD] = CPDFromFactor(F, Y)
  nvars = length(F.var);

  % Reorder the var, card and val fields of Fnew so that the first var is the
  % child variable.
  Fnew = F;
  YIndexInF = find(F.var == Y);
  this.card = F.card( YIndexInF );

  % Parents is a dummy factor
  Parents.var = F.var(find(F.var ~= Y));
  Parents.card = F.card(find(F.var ~= Y));
  Parents.val = ones(prod(Parents.card),1);

  Fnew.var = [Y Parents.var];
  Fnew.card = [this.card Parents.card];
  for i=1:length(F.val)
    A = IndexToAssignment(i, F.card);
    y = A(YIndexInF);
    A( YIndexInF ) = [];
    A = [y A];
    j = AssignmentToIndex(A, Fnew.card);
    Fnew.val(j) = F.val(i);
  end

  % normalize
```

```
  CPD = NormalizeCPDFactors(Fnew);
```

# 5 ./EliminateVar.m

```
% Function used in production of clique trees
% F = list of factors
% E = adjacency matrix for variables
% Z = variable to eliminate
%
% Copyright (C) Daphne Koller, Stanford University, 2012

function [newF E] = EliminateVar(F, E, Z)

  % Index of factors to multiply (b/c they contain Z)
  useFactors = [];

  % Union of scopes of factors to multiply
  scope = [];

  for i=1:length(F)
    if any(F(i).var == Z)
      useFactors = [useFactors i];
      scope = union(scope, F(i).var);
    end
  end

  % update edge map
  % These represent the induced edges for the VE graph.
  for i=1:length(scope)
    for j=1:length(scope)

      if i~=j
        E(scope(i),scope(j)) = 1;
        E(scope(j),scope(i)) = 1;
      end
    end
  end

  % Remove all adjacencies for the variable to be eliminated
  E(Z,:) = 0;
  E(:,Z) = 0;


  % nonUseFactors = list of factors (not indices!) which are passed through
  % in this round
  nonUseFactors = setdiff(1:length(F),[useFactors]);

  for i=1:length(nonUseFactors)

    % newF = list of factors we will return
    newF(i) = F(nonUseFactors(i));

    % newmap = ?
    newmap(nonUseFactors(i)) = i;

  end

  % Multiply factors which involve Z -> newFactor
  newFactor = struct('var', [], 'card', [], 'val', []);
  for i=1:length(useFactors)
    newFactor = FactorProduct(newFactor,F(useFactors(i)));
  end

  newFactor = FactorMarginalization(newFactor,Z);
  newF(length(nonUseFactors)+1) = newFactor;
```

# 6 ./FactorMarginalization.m

```
% FactorMarginalization Sums given variables out of a factor.
%   B = FactorMarginalization(A,V) computes the factor with the variables
%   in V summed out. The factor data structure has the following fields:
%       .var    Vector of variables in the factor, e.g. [1 2 3]
%       .card   Vector of cardinalities corresponding to .var, e.g. [2 2 2]
%       .val    Value table of size prod(.card)
%
%   The resultant factor should have at least one variable remaining or this
%   function will throw an error.
%
```

```
%   See also FactorProduct.m, IndexToAssignment.m, and AssignmentToIndex.m
%
% Copyright (C) Daphne Koller, Stanford University, 2012

function B = FactorMarginalization(A, V)

% Check for empty factor or variable list
if (isempty(A.var) || isempty(V)), B = A; return; end;

% Construct the output factor over A.var \ V (the variables in A.var that are not in V)
% and mapping between variables in A and B
[B.var, mapB] = setdiff(A.var, V);

% Check for empty resultant factor
if isempty(B.var)
  %error('Error: Resultant factor has empty scope');
  B.var = [];
  B.card = [];
  B.val = [];
  return;
end;

% Initialize B.card and B.val
B.card = A.card(mapB);
B.val = zeros(1,prod(B.card));

% Compute some helper indices
% These will be very useful for calculating C.val
% so make sure you understand what these lines are doing
assignments = IndexToAssignment(1:length(A.val), A.card);
indxB = AssignmentToIndex(assignments(:, mapB), B.card);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% YOUR CODE HERE
% Correctly populate the factor values of B
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i = 1:length(A.val),
        B.val(indxB(i)) = B.val(indxB(i)) + A.val(i);
    end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end
```

# 7 ./FactorProduct.m

```
% FactorProduct Computes the product of two factors.
%   C = FactorProduct(A,B) computes the product between two factors, A and B,
%   where each factor is defined over a set of variables with given dimension.
%   The factor data structure has the following fields:
%       .var    Vector of variables in the factor, e.g. [1 2 3]
%       .card   Vector of cardinalities corresponding to .var, e.g. [2 2 2]
%       .val    Value table of size prod(.card)
%
%   See also FactorMarginalization.m, IndexToAssignment.m, and
%   AssignmentToIndex.m
%
% Copyright (C) Daphne Koller, Stanford University, 2012

function C = FactorProduct(A, B)

% Check for empty factors
if (isempty(A.var)), C = B; return; end;
if (isempty(B.var)), C = A; return; end;

% Check that variables in both A and B have the same cardinality
[dummy iA iB] = intersect(A.var, B.var);
if ~isempty(dummy)
        % A and B have at least 1 variable in common
        assert(all(A.card(iA) == B.card(iB)), 'Dimensionality␣mismatch␣in␣factors');
end

% Set the variables of C
C.var = union(A.var, B.var);

% Construct the mapping between variables in A and B and variables in C.
% In the code below, we have that
%
%   mapA(i) = j, if and only if, A.var(i) == C.var(j)
%
% and similarly
```

```
%
%   mapB(i) = j, if and only if, B.var(i) == C.var(j)
%
% For example, if A.var = [3 1 4], B.var = [4 5], and C.var = [1 3 4 5],
% then, mapA = [2 1 3] and mapB = [3 4]; mapA(1) = 2 because A.var(1) = 3
% and C.var(2) = 3, so A.var(1) == C.var(2).

[dummy, mapA] = ismember(A.var, C.var);
[dummy, mapB] = ismember(B.var, C.var);

% Set the cardinality of variables in C
C.card = zeros(1, length(C.var));
C.card(mapA) = A.card;
C.card(mapB) = B.card;

% Initialize the factor values of C:
%   prod(C.card) is the number of entries in C
C.val = zeros(1,prod(C.card));

% Compute some helper indices
% These will be very useful for calculating C.val
% so make sure you understand what these lines are doing.
assignments = IndexToAssignment(1:prod(C.card), C.card);
indxA = AssignmentToIndex(assignments(:, mapA), A.card);
indxB = AssignmentToIndex(assignments(:, mapB), B.card);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% YOUR CODE HERE:
% Correctly populate the factor values of C
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C.val = A.val(indxA) .* B.val(indxB);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

end
```

# 8 ./FactorSum.m

```
% FactorProduct Computes the product of two factors.
%   C = FactorProduct(A,B) computes the product between two factors, A and B,
%   where each factor is defined over a set of variables with given dimension.
%   The factor data structure has the following fields:
%       .var    Vector of variables in the factor, e.g. [1 2 3]
%       .card   Vector of cardinalities corresponding to .var, e.g. [2 2 2]
%       .val    Value table of size prod(.card)
%
%   See also FactorMarginalization.m, IndexToAssignment.m, and
%   AssignmentToIndex.m
%
% Copyright (C) Daphne Koller, Stanford University, 2012

function C = FactorSum(A, B)

% Check for empty factors
if (isempty(A.var)), C = B; return; end;
if (isempty(B.var)), C = A; return; end;

% Check that variables in both A and B have the same cardinality
[dummy iA iB] = intersect(A.var, B.var);
if ~isempty(dummy)
        % A and B have at least 1 variable in common
        assert(all(A.card(iA) == B.card(iB)), 'Dimensionality␣mismatch␣in␣factors');
end

% Set the variables of C
C.var = union(A.var, B.var);

% Construct the mapping between variables in A and B and variables in C.
% In the code below, we have that
%
%   mapA(i) = j, if and only if, A.var(i) == C.var(j)
%
% and similarly
%
%   mapB(i) = j, if and only if, B.var(i) == C.var(j)
%
% For example, if A.var = [3 1 4], B.var = [4 5], and C.var = [1 3 4 5],
% then, mapA = [2 1 3] and mapB = [3 4]; mapA(1) = 2 because A.var(1) = 3
% and C.var(2) = 3, so A.var(1) == C.var(2).
```

```
[dummy, mapA] = ismember(A.var, C.var);
[dummy, mapB] = ismember(B.var, C.var);

% Set the cardinality of variables in C
C.card = zeros(1, length(C.var));
C.card(mapA) = A.card;
C.card(mapB) = B.card;

% Initialize the factor values of C:
%   prod(C.card) is the number of entries in C
C.val = zeros(1,prod(C.card));

% Compute some helper indices
% These will be very useful for calculating C.val
% so make sure you understand what these lines are doing.
assignments = IndexToAssignment(1:prod(C.card), C.card);
indxA = AssignmentToIndex(assignments(:, mapA), A.card);
indxB = AssignmentToIndex(assignments(:, mapB), B.card);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% YOUR CODE HERE:
% Correctly populate the factor values of C
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C.val = A.val(indxA) + B.val(indxB);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

end
```

# 9 ./GetValueOfAssignment.m

```
% GetValueOfAssignment Gets the value of a variable assignment in a factor.
%
%   v = GetValueOfAssignment(F, A) returns the value of a variable assignment,
%   A, in factor F. The order of the variables in A are assumed to be the
%   same as the order in F.var.
%
%   v = GetValueOfAssignment(F, A, VO) gets the value of a variable assignment,
%   A, in factor F. The order of the variables in A are given by the vector VO.
%
%   See also SetValueOfAssignment.m and SampleFactors.m
%
% Copyright (C) Daphne Koller, Stanford University, 2012

function v = GetValueOfAssignment(F, A, VO)

if (nargin == 2),
    indx = AssignmentToIndex(A, F.card);
else
    map = zeros(length(F.var), 1);
    for i = 1:length(F.var),
        map(i) = find(VO == F.var(i));
    end;
    indx = AssignmentToIndex(A(map), F.card);
end;

v = F.val(indx);

end
```

# 10 ./IndexToAssignment.m

```
% IndexToAssignment Convert index to variable assignment.
%
%   A = IndexToAssignment(I, D) converts an index, I, into the .val vector
%   into an assignment over variables with cardinality D. If I is a vector,
%   then the function produces a matrix of assignments, one assignment
%   per row.
%
%   See also AssignmentToIndex.m and SampleFactors.m
%
% Copyright (C) Daphne Koller, Stanford University, 2012

function A = IndexToAssignment(I, D)

D = D(:)'; % ensure that D is a row vector
A = mod(floor(repmat(I(:) - 1, 1, length(D)) ./ repmat(cumprod([1, D(1:end - 1)]), length(I), 1)), ...
        repmat(D, length(I), 1)) + 1;
```

```
end
```

# 11 ./NormalizeCPDFactors.m

```
% Copyright (C) Daphne Koller, Stanford University, 2012

function [F] = NormalizeCPDFactors(F)

  NumFactors = length(F);
  for i=1:NumFactors

    f = F(i);
    dummy.var = f.var(2:end);
    dummy.card = f.card(2:end);
    dummy.val = zeros(1,prod(dummy.card));

    % Now for each joint assignment to parents, renormalize the
    % values for that joint assignment to sum to 1.

    for a=1:length(dummy.val)
      A = IndexToAssignment(a, dummy.card);
      Indices = [];
      for d=1:f.card(1)
          Indices = [Indices AssignmentToIndex([d A], f.card);];
      end
      if sum(f.val(Indices)) == 0
          % Set f.val(Indices) to 0
          f.val(Indices) = 0;
      else
          f.val(Indices) = f.val(Indices) / sum(f.val(Indices));
      end
    end

    f.val(find(isnan(f.val))) = 0;

    F(i) = f;

  end
```

# 12 ./NormalizeFactorValues.m

```
% Copyright (C) Daphne Koller, Stanford University, 2012

function F = NormalizeFactorValues( F )

  for i=1:length(F)
    ThisFactor = F(i);
    ThisFactor.val = ThisFactor.val / sum(ThisFactor.val);
    F(i) = ThisFactor;
  end
```

# 13 ./ObserveEvidence.m

```
% ObserveEvidence Modify a vector of factors given some evidence.
%   F = ObserveEvidence(F, E) sets all entries in the vector of factors, F,
%   that are not consistent with the evidence, E, to zero. F is a vector of
%   factors, each a data structure with the following fields:
%     .var    Vector of variables in the factor, e.g. [1 2 3]
%     .card   Vector of cardinalities corresponding to .var, e.g. [2 2 2]
%     .val    Value table of size prod(.card)
%   E is an N-by-2 matrix, where each row consists of a variable/value pair.
%     Variables are in the first column and values are in the second column.
%   NOTE - DOES NOT RENORMALIZE THE FACTOR VALUES
%
% Copyright (C) Daphne Koller, Stanford University, 2012

function F = ObserveEvidence(F, E, normalize)

  % Iterate through all evidence
  for i = 1:size(E, 1),
    v = E(i, 1); % variable
    x = E(i, 2); % value

    % Check validity of evidence
    if (x == 0),
      warning(['Evidence␣not␣set␣for␣variable␣', int2str(v)]);
```

```
      continue;
    end;

    % Iterate through the factors
    for j = 1:length(F),
      % Does factor contain variable?
      indx = find(F(j).var == v);

      if (~isempty(indx)),

        % Check validity of evidence
        if (x > F(j).card(indx) || x < 0 ),
          error(['Invalid evidence, X_', int2str(v), ' = ', int2str(x)]);
        end;

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % YOUR CODE HERE
        % Adjust the factor F(j) to account for observed evidence
        % Hint: You might find it helpful to use IndexToAssignment
        %       and SetValueOfAssignment
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % For each value (1-1 map between assignment and values)
        for k = 1:length(F(j).val),

          % get assignment for this index
          A = IndexToAssignment(k, F(j).card);

          % indx = index of evidence variable in this factor
          if (A(indx) ~= x),
            F(j).val(k) = 0;
          end;

        end;
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % Check validity of evidence / resulting factor
        if (all(F(j).val == 0)),
          warning(['Factor ', int2str(j), ' makes variable assignment impossible']);
        end;

      end % if (!isempty(index))

    end % for j = 1:length(F),

  end % for i = 1:size(E, 1),

  if (nargin == 3)
    if (normalize)
      F = NormalizeCPDFactors(F);
    end
  end
```

# 14 ./OptimizeLinearExpectations.m

```
% Copyright (C) Daphne Koller, Stanford University, 2012

function [MEU OptimalDecisionRule] = OptimizeLinearExpectations( I )
  % Inputs: An influence diagram I with a single decision node and one or more utility nodes.
  %         I.RandomFactors = list of factors for each random variable.  These are CPDs, with
  %              the child variable = D.var(1)
  %         I.DecisionFactors = factor for the decision node.
  %         I.UtilityFactors = list of factors representing conditional utilities.
  % Return value: the maximum expected utility of I and an optimal decision rule
  % (represented again as a factor) that yields that expected utility.
  % You may assume that there is a unique optimal decision.
  %
  % This is similar to OptimizeMEU except that we will have to account for
  % multiple utility factors.  We will do this by calculating the expected
  % utility factors and combining them, then optimizing with respect to that
  % combined expected utility factor.
  MEU = [];
  OptimalDecisionRule = [];
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %
  % YOUR CODE HERE
  %
  % A decision rule for D assigns, for each joint assignment to D's parents,
  % probability 1 to the best option from the EUF for that joint assignment
  % to D's parents, and 0 otherwise.  Note that when D has no parents, it is
```

```
  % a degenerate case we can handle separately for convenience.
  %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  I2 = I;
  EUF = struct('var', [], 'card', [], 'val', []);
  for i=1:length(I2.UtilityFactors)
      I.UtilityFactors = I2.UtilityFactors(i);
      EUFs{i} = CalculateExpectedUtilityFactor(I);
      EUF = FactorSum(EUF, EUFs{i});
  end;

  D = I2.DecisionFactors;

  OptimalDecisionRule = EUF;
  OptimalDecisionRule.val = zeros(size(EUF.val));
  if(length(D.var) == 1)
      [~, id] = max(EUF.val);
      OptimalDecisionRule.val(id) = 1;
  else
      for i=1:length(D.var)
          map(i) = find(EUF.var == D.var(i));
          invMap(i) = find(D.var == EUF.var(i));
      end;
      assignEUF = IndexToAssignment(1:length(EUF.val), EUF.card);
      assignD = assignEUF(:, map);
      PAs = AssignmentToIndex(1:prod(D.card(2:end)), D.card(2:end));
      for i=1:size(PAs,1)
%           [~,ids] = find(ismember(assign(:,2:end), PAs(i,:)),1);
          newAssigns = [[1:D.card(1)]', repmat(PAs(i,:), length([1:D.card(1)]),1)];
          ids = AssignmentToIndex(newAssigns(:,invMap), EUF.card);
          [~,id] = max(EUF.val(ids));
          OptimalDecisionRule.val(ids(id)) = 1;
      end;



  end;
  F = FactorProduct(OptimalDecisionRule, EUF);
  MEU = sum(F.val(:));




end
```

# 15 ./OptimizeMEU.m

```
% Copyright (C) Daphne Koller, Stanford University, 2012

function [MEU OptimalDecisionRule] = OptimizeMEU( I )

  % Inputs: An influence diagram I with a single decision node and a single utility node.
  %         I.RandomFactors = list of factors for each random variable.  These are CPDs, with
  %              the child variable = D.var(1)
  %         I.DecisionFactors = factor for the decision node.
  %         I.UtilityFactors = list of factors representing conditional utilities.
  % Return value: the maximum expected utility of I and an optimal decision rule
  % (represented again as a factor) that yields that expected utility.

  % We assume I has a single decision node.
  % You may assume that there is a unique optimal decision.
  D = I.DecisionFactors(1);

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %
  % YOUR CODE HERE...
  %
  % Some other information that might be useful for some implementations
  % (note that there are multiple ways to implement this):
  % 1.  It is probably easiest to think of two cases - D has parents and D
  %     has no parents.
  % 2.  You may find the Matlab/Octave function setdiff useful.
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  EUF = CalculateExpectedUtilityFactor(I);
  OptimalDecisionRule = EUF;
  OptimalDecisionRule.val = zeros(size(EUF.val));
```

```
  if(length(D.var) == 1)
      [~, id] = max(EUF.val);
      OptimalDecisionRule.val(id) = 1;
  else
      for i=1:length(D.var)
          map(i) = find(EUF.var == D.var(i));
          invMap(i) = find(D.var == EUF.var(i));
      end;
      assignEUF = IndexToAssignment(1:length(EUF.val), EUF.card);
      assignD = assignEUF(:, map);
      PAs = AssignmentToIndex(1:prod(D.card(2:end)), D.card(2:end));
      for i=1:size(PAs,1)
%           [~,ids] = find(ismember(assign(:,2:end), PAs(i,:)),1);
          newAssigns = [[1:D.card(1)]', repmat(PAs(i,:), length([1:D.card(1)]),1)];
          ids = AssignmentToIndex(newAssigns(:,invMap), EUF.card);
          [~,id] = max(EUF.val(ids));
          OptimalDecisionRule.val(ids(id)) = 1;
      end;



  end;
  F = FactorProduct(OptimalDecisionRule, EUF);
  MEU = sum(F.val(:));









end
```

# 16 ./OptimizeWithJointUtility.m

```
% Copyright (C) Daphne Koller, Stanford University, 2012

function [MEU OptimalDecisionRule] = OptimizeWithJointUtility( I )
  % Inputs: An influence diagram I with a single decision node and one or more utility nodes.
  %         I.RandomFactors = list of factors for each random variable.  These are CPDs, with
  %              the child variable = D.var(1)
  %         I.DecisionFactors = factor for the decision node.
  %         I.UtilityFactors = list of factors representing conditional utilities.
  % Return value: the maximum expected utility of I and an optimal decision rule
  % (represented again as a factor) that yields that expected utility.
  % You may assume that there is a unique optimal decision.

  % This is similar to OptimizeMEU except that we must find a way to
  % combine the multiple utility factors.  Note: This can be done with very
  % little code.

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %
  % YOUR CODE HERE
  %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Us = I.UtilityFactors;
    U = Us(1);
    for i=2:length(Us)
        U = FactorSum(U, Us(i));
    end;
    I.UtilityFactors = U;

    [MEU OptimalDecisionRule] = OptimizeMEU(I);

end
```

# 17 ./PA6_RunTests.m

```
% A simple test suite for PA 6
%
% Based on the code by Mihaly Barasz posted on the forum.
%
% copy the CompareData.m file from last weeks test suite
% into the directory for this weeks assignment and save this file
% as PA6_RunTests.m
%
function result = PA6_RunTests(anyway)
```

```
if ~exist('CompareData', 'file')
  fprintf('please␣install␣CompareData.m␣as␣indicated␣in␣the␣comment\n');
  fprintf('at␣the␣beginning␣of␣this␣file\n');
  result = false;
  return;
end

if (~exist('TS','var'))
  %% This is based on TestCases to make it all more testable.

  TS = repmat(struct('I', [], 'allDs', [], 'allEU', [], 'EUF', []), 1, 3);

  %% Test case 1.
  I.RandomFactors = struct('var', [1], 'card', [2], 'val', normval([7, 3]));
  I.DecisionFactors = struct('var', [2], 'card', [2], 'val', [1 0]);
  I.UtilityFactors = struct('var', [1, 2], 'card', [2, 2], 'val', [10, 1, 5, 1]);

  TS(1).I = I;
  TS(1).allDs = [1 0; 0 1];
  TS(1).EUF = struct('var', [2], 'card', [2], 'val', [7.3 3.8]);
  TS(1).allEU = [7.3 3.8];
  TS(1).MEU = 7.3;
  TS(1).OptDR = struct('var', [2], 'card', [2], 'val', [1 0]);

  %% Test case 2.
  I.RandomFactors = ...
      [struct('var', [1], 'card', [2], 'val', normval([7, 3])), ...
       CPDFromFactor(struct('var', [3,1,2], 'card', [2,2,2], 'val', [4 4 1 1 1 1 4 4]), 3)];
  I.DecisionFactors = struct('var', [2], 'card', [2], 'val', [1 0]);
  I.UtilityFactors = struct('var', [2,3], 'card', [2, 2], 'val', [10, 1, 5, 1]);

  TS(2).I = I;
  TS(2).allDs = [1 0; 0 1];
  TS(2).EUF = struct('var', [2], 'card', [2], 'val', [7.5 1.0]);
  TS(2).allEU = [7.5 1.0];
  TS(2).MEU = 7.5;
  TS(2).OptDR = struct('var', [2], 'card', [2], 'val', [1 0]);

  %% Test case 3.
  I.RandomFactors = ...
      [struct('var', [1], 'card', [2], 'val', normval([7, 3])), ...
       CPDFromFactor(struct('var', [3,1,2], 'card', [2,2,2], 'val', [4 4 1 1 1 1 4 4]), 3)];
  I.DecisionFactors = struct('var', [2,1], 'card', [2,2], 'val', [1,0,0,1]);
  I.UtilityFactors = struct('var', [2,3], 'card', [2, 2], 'val', [10, 1, 5, 1]);

  TS(3).I = I;
  TS(3).allDs = [1 0 1 0; 1 0 0 1; 0 1 1 0; 0 1 0 1];
  TS(3).EUF = struct('var', [1,2], 'card', [2 2], 'val', [5.25 2.25 0.7 0.3]);
  TS(3).allEU = [7.5 5.55 2.95 1.0];
  TS(3).MEU = 7.5;
  TS(3).OptDR = struct('var', [1,2], 'card', [2,2], 'val', [1,1,0,0]);

  %% Test case 4.
  I.RandomFactors = ...
      [struct('var', [1], 'card', [2], 'val', normval([7, 3])), ...
       CPDFromFactor(struct('var', [3,1,2], 'card', [2,2,2], 'val', [4 4 1 1 1 1 4 4]), 3)];
  I.DecisionFactors = struct('var', [2,1], 'card', [2,2], 'val', [1,0,0,1]);
  I.UtilityFactors = ...
      [struct('var', [2,3], 'card', [2, 2], 'val', [10, 1, 5, 1]), ...
       struct('var', [2], 'card', [2], 'val', [1, 10])];

  T4.I = I;
  T4.MEU = 11;
  T4.OptDR = struct('var', [1,2], 'card', [2,2], 'val', [0,0,1,1]);

end

if nargin == 1
  run_all = anyway;
else
  run_all = false;
end

ok = true;
passed = 0;
skipped = 0;
failed = 0;
for test = 1:5
  if ~(run_all || ok)
```

```
    skipped = skipped + 1;
    continue;
  end

  switch test
    case 1
      for t = 1:length(TS)
        T = TS(t);
        for d = 1:size(T.allDs, 1)
          T.I.DecisionFactors.val = T.allDs(d, :);
          EU = SimpleCalcExpectedUtility(T.I);
          ok = checkResult('SimpleCalcExpectedUtility', EU, T.allEU(d));
        end
      end

    case 2
      for t = 1:length(TS)
        T = TS(t);
        EUF = CalculateExpectedUtilityFactor(T.I);
        ok = checkResult('CalculateExpectedUtilityFactor', EUF, T.EUF);
      end

    case 3
      for t = 1:length(TS)
        T = TS(t);
        [meu optdr] = OptimizeMEU(T.I);
        ok = checkResult('OptimizeMEU meu', meu, T.MEU);
        ok = ok && checkResult('OptimizeMEU optdr', optdr, T.OptDR);
      end

    case 4
      [meu optdr] = OptimizeWithJointUtility(T4.I);
      ok = checkResult('OptimizeWithJointUtility meu', meu, T4.MEU);
      ok = ok && checkResult('OptimizeWithJointUtility optdr', optdr, T4.OptDR);
      %% Also, see if it works with single utility:
      for t = 1:length(TS)
        T = TS(t);
        [meu optdr] = OptimizeWithJointUtility(T.I);
        ok = ok && checkResult('OptimizeWithJointUtility sng meu', meu, T.MEU);
        ok = ok && checkResult('OptimizeWithJointUtility sng optdr', optdr, T.OptDR);
      end

    case 5
      [meu optdr] = OptimizeLinearExpectations(T4.I);
      ok = checkResult('OptimizeLinearExpectations meu', meu, T4.MEU);
      ok = ok && checkResult('OptimizeLinearExpectations optdr', optdr, T4.OptDR);
      %% Also, see if it works with single utility:
      for t = 1:length(TS)
        T = TS(t);
        [meu optdr] = OptimizeLinearExpectations(T.I);
        ok = ok && checkResult('OptimizeLinearExpectation sng meu', meu, T.MEU);
        ok = ok && checkResult('OptimizeLinearExpectation sng optdr', optdr, T.OptDR);
      end
  end

  if ok
    passed = passed + 1;
  else
    failed = failed + 1;
  end
end

fprintf('%d tests OK, %d skipped, %d failed\n', passed, skipped, failed);
result = ok;
end

function res = checkResult(label, expected, observed)
params = struct('displaycontextprogress', 0, 'NumericTolerance', 1e-6);
  cmp = CompareData(expected, observed, [], params);
  fprintf('%s: ', label);
  if cmp
    fprintf('OK\n');
    res = true;
  else
    fprintf('FAIL\n');
    res = false;
  end
end

function v = normval(v)
```

```
  v = v / sum(v);
end
```

# 18 ./PrintFactor.m

```
% Copyright (C) Daphne Koller, Stanford University, 2012

function [] = PrintFactor(F)
  % Pretty print the factor F.
  % The first row lists the variables and subsequent rows are
  % the joint assignment and their associated factor value in
  % the last column.

  for i=1:length(F.var)
    fprintf(1, '%d\t', F.var(i));
  end
  fprintf(1, '\n');

  for i=1:length(F.val)
    A = IndexToAssignment(i, F.card);
    for j=1:length(A)
      fprintf(1, '%d\t', A(j));
    end
    fprintf(1, '%f\n', F.val(i));
  end


end
```

# 19 ./SetValueOfAssignment.m

```
% SetValueOfAssignment Sets the value of a variable assignment in a factor.
%
%   F = SetValueOfAssignment(F, A, v) sets the value of a variable assignment,
%   A, in factor F to v. The order of the variables in A are assumed to be the
%   same as the order in F.var.
%
%   F = SetValueOfAssignment(F, A, v, VO) sets the value of a variable
%   assignment, A, in factor F to v. The order of the variables in A are given
%   by the vector VO.
%
%   Note that SetValueOfAssignment *does not modify* the factor F that is
%   passed into the function, but instead returns a modified factor with the
%   new value(s) for the specified assignment(s). This is why we have to
%   reassign F to the result of SetValueOfAssignment in the code snippets
%   shown above.
%
%   See also GetValueOfAssignment.m and SampleFactors.m
%
% Copyright (C) Daphne Koller, Stanford University, 2012

function F = SetValueOfAssignment(F, A, v, VO)

if (nargin == 3),
    indx = AssignmentToIndex(A, F.card);
else
    map = zeros(length(F.var), 1);
    for i = 1:length(F.var),
        map(i) = find(VO == F.var(i));
    end;
    indx = AssignmentToIndex(A(map), F.card);
end;

F.val(indx) = v;

end
```

# 20 ./SimpleCalcExpectedUtility.m

```
% Copyright (C) Daphne Koller, Stanford University, 2012

function EU = SimpleCalcExpectedUtility(I)

  % Inputs: An influence diagram, I (as described in the writeup).
  %         I.RandomFactors = list of factors for each random variable.  These are CPDs, with
  %              the child variable = D.var(1)
```

```
  %         I.DecisionFactors = factor for the decision node.
  %         I.UtilityFactors = list of factors representing conditional utilities.
  % Return Value: the expected utility of I
  % Given a fully instantiated influence diagram with a single utility node and decision node,
  % calculate and return the expected utility.  Note - assumes that the decision rule for the
  % decision node is fully assigned.

  % In this function, we assume there is only one utility node.
  F = [I.RandomFactors I.DecisionFactors];
  U = I.UtilityFactors(1);
  EU = [];
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %
  % YOUR CODE HERE
  %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  PaU = U.var;
  V = unique([F(:).var]);
  diff = setdiff(V, PaU);
  Fnew = VariableElimination(F, diff);
  F = Fnew(1);
  for i=2:length(Fnew)
      F = FactorProduct(F, Fnew(i));
  end;
  P = FactorProduct(F, U);
  EU = sum(P.val);



end
```

# 21 ./SimpleOptimizeMEU.m

```
% Copyright (C) Daphne Koller, Stanford University, 2012

function [MEU OptimalDecisionRule] = SimpleOptimizeMEU(I)

  % We assume there is only one decision rule in this function.
  D = I.DecisionFactors(1);

  PossibleDecisionRules = EnumerateDecisionRules(D);

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %
  % YOUR CODE HERE
  % 1.  You must find which of the decision rules you have enumerated has the
  %     highest expected utility.  You should use your implementation of
  %     SimpleCalcExpectedUtility from P1.  Set the values of MEU and OptimalDecisionRule
  %     to the best achieved expected utility and the corresponding decision
  %     rule respectively.
  %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


end
```

# 22 ./submit.m

```
function submit(partId, webSubmit)
%SUBMIT Submit your code and output to the pgm-class servers
%   SUBMIT() will connect to the pgm-class server and submit your solution
%   There is no penalty for submitting, so go ahead and try this!
%
%   If this function does not work for you, use the web-submission mechanism.
%   Call the submitWeb function, which will produce a file for the part you
%   wish to submit. Then, submit the file to the class servers using the
%   "Web Submission" button on the Programming Assignments page on the course
%   website.
%
%   webSubmit is a boolean variable that specifies whether to prepare
%   a file for web-submission (if webSubmit = 1) or to submit directly
%   to the server (if webSubmit = 0, or is not specified); you should call
%   submitWeb if you want to do a web-submission.
%
% Copyright (C) Daphne Koller, Stanford University, 2012
 warning off all;
 fprintf('==========================================================\n');
```

```
fprintf('                    Honor Code Statement:\n');
fprintf('------------------------------------------------------\n');
fprintf('The Honor Code is an undertaking of the students,\n');
fprintf('individually and collectively, that they will not\n');
fprintf('give or receive unpermitted aid in class work and\n');
fprintf('will do their share and take an active part in seeing\n');
fprintf('to it that others as well as themselves uphold the\n');
fprintf('spirit and letter of the Honor Code.\n\n');
fprintf('Sharing, collaboration, or looking at any code related\n');
fprintf('to Programming Assignments that is not your own is\n');
fprintf('considered a violation of the Honor Code.\n');
fprintf('------------------------------------------------------\n');
fprintf('By submitting the Programming Assignment, I\n');
fprintf('acknowledge and accept the Honor Code.\n');
fprintf('======================================================\n\n');

fprintf('==\n== [pgm-class] Submitting Solutions | Programming Assignment %s\n==\n', ...
        homework_id());

 if ~exist('partId', 'var') || isempty(partId)
   partId = promptPart();
 end

 if ~exist('webSubmit', 'var') || isempty(webSubmit)
   webSubmit = 0; % submit directly by default
 end

 % Check valid partId
 partNames = validParts();
 if ~isValidPartId(partId)
   fprintf('!! Invalid assignment part selected.\n');
   fprintf('!! Expected an integer from 1 to %d.\n', numel(partNames) + 1);
   fprintf('!! Submission Cancelled\n');
   return
 end

 if ~exist('pgm_login_data.mat','file')
   [login password] = loginPrompt();
   save('pgm_login_data.mat','login','password');
 else
   load('pgm_login_data.mat');
   [login password] = quickLogin(login,password);
   save('pgm_login_data.mat','login','password');
 end

 if isempty(login)
   fprintf('!! Submission Cancelled\n');
   return
 end

 fprintf('\n== Connecting to pgm-class ... ');
 if exist('OCTAVE_VERSION')
   fflush(stdout);
 end


 % Setup submit list
 if partId == numel(partNames) + 1
   submitParts = 1:numel(partNames);
 else
   submitParts = [partId];
 end

 for s = 1:numel(submitParts)
   % Submit this part
   partId = submitParts(s);

   for thisPartId = subParts(partId)
     if (~webSubmit) % submit directly to server

       [login, ch, signature, auxstring] = getChallenge(login, thisPartId);
       if isempty(login) || isempty(ch) || isempty(signature)
         % Some error occured, error string in first return element.
         fprintf('\n!! Error: %s\n\n', login);
         return
       end

       % Attempt Submission with Challenge
       ch_resp = challengeResponse(login, password, ch);
```

```
        [result, str] = submitSolution(login, ch_resp, thisPartId, ...
               output(thisPartId, auxstring), source(partId), signature);

        if (~isTest(thisPartId))
          partName = partNames{partId};
        else
          partName = [partNames{partId} ' (test)'];
        end

        fprintf('\n== [pgm-class] Submitted Assignment %s - Part %d - %s\n', ...
          homework_id(), partId, partName);
        fprintf('== %s\n', strtrim(str));

        if exist('OCTAVE_VERSION')
          fflush(stdout);
        end

      else % make web submission files

        [result] = submitSolutionWeb(login, thisPartId, output(thisPartId), ...
                              source(partId));
        result = base64encode(result);

        if (~isTest(thisPartId))
          partType = 'sample';
        else
          partType = 'test';
        end

        fprintf('\nSave as submission file [submit_pa%s_part%d_%s.txt (enter to accept default)]:', ...
          homework_id(), partId, partType);
        saveAsFile = input('', 's');
        if (isempty(saveAsFile))
          saveAsFile = sprintf('submit_pa%s_part%d_%s.txt', homework_id(), partId, partType);
        end

        fid = fopen(saveAsFile, 'w');
        if (fid)
          fwrite(fid, result);
          fclose(fid);
          fprintf('\nSaved your solutions to %s.\n\n', saveAsFile);
          fprintf(['You can now submit your solutions using the' ...
             'Web Submission button\non the Assignments page\n']);
        else
          fprintf('Unable to save to %s\n\n', saveAsFile);
          fprintf(['You can create a submission file by saving the\n' ...
             'following text in a file: (press enter to continue)\n\n']);
          pause;
          fprintf(result);
        end

      end
    end
  end
end

% ================== CONFIGURABLES FOR EACH HOMEWORK ==================

function id = homework_id()
  id = '6';
end

function [partNames] = validParts()
  partNames = { 'SimpleCalcExpectedUtility', ...
                'CalculateExpectedUtilityFactor', ...
                'OptimizeMEU', ...
                'OptimizeWithJointUtility', ...
                'OptimizeLinearExpectations'
              };
end

function srcs = sources()
  % Separated by part
  srcs = { { 'SimpleCalcExpectedUtility.m'}, ...
           { 'CalculateExpectedUtilityFactor.m'}, ...
           { 'OptimizeMEU.m'}, ...
           { 'OptimizeWithJointUtility.m'}, ...
           { 'OptimizeLinearExpectations.m'}
         };
end
```

```
% defines the shown part to back-end part mappings
function parts = subParts(partId)
  first = 2 * (partId - 1) + 1;
  parts = [first, first + 1];
end

% specifies which parts are test parts
function result = isTest(partId)
  if (mod(partId, 2) == 0)
      result = true;
  else
      result = false;
  end
end

function out = output(partId, auxstring)
 if partId == 1
    load 'FullI.mat';
    out = SerializeFloat(SimpleCalcExpectedUtility(FullI));
    clear FullI;

  elseif partId == 2

    load 'FullI.mat';
    Fe = FullI.RandomFactors;
    Fe = ObserveEvidence(Fe, [3 2], 1);
    FullI.RandomFactors = Fe;
    out = SerializeFloat(SimpleCalcExpectedUtility(FullI));
    clear FullI;
    clear Fe;

  elseif partId == 3

    load 'FullI.mat';
    out = SerializeFactorsFg(CalculateExpectedUtilityFactor(FullI));
    clear FullI;

  elseif partId == 4

    load 'FullI.mat';
    Fe = FullI.RandomFactors;
    Fe = ObserveEvidence(Fe, [3 2], 1);
    FullI.RandomFactors = Fe;
    out = SerializeFactorsFg(CalculateExpectedUtilityFactor(FullI));
    clear FullI;
    clear Fe;

  elseif partId == 5

    load 'FullI.mat';
    [meu optdr] = OptimizeMEU(FullI);
    out = SerializeMEUOptimization(meu, optdr);
    clear FullI;

  elseif partId == 6

    load 'FullI.mat';
    Fe = FullI.RandomFactors;
    Fe = ObserveEvidence(Fe, [3 2], 1);
    FullI.RandomFactors = Fe;
    [meu optdr] = OptimizeMEU(FullI);
    out = SerializeMEUOptimization(meu, optdr);
    clear FullI;
    clear Fe;

  elseif partId == 7

    load 'MultipleUtilityI.mat';
    [meu optdr] = OptimizeWithJointUtility(MultipleUtilityI);
    out = SerializeMEUOptimization(meu, optdr);
    clear MultipleUtility;

  elseif partId == 8

    load 'MultipleUtilityI.mat';
    Fe = MultipleUtilityI.RandomFactors;
    Fe = ObserveEvidence(Fe, [3 1], 1);
    MultipleUtilityI.RandomFactors = Fe;
    [meu optdr] = OptimizeWithJointUtility(MultipleUtilityI);
```

```
    out = SerializeMEUOptimization(meu, optdr);
    clear MultipleUtilityI;
    clear Fe;

  elseif partId == 9

    load 'MultipleUtilityI.mat';
    [meu optdr] = OptimizeLinearExpectations(MultipleUtilityI);
    out = SerializeMEUOptimization(meu, optdr);
    clear MultipleUtilityI;

  elseif partId == 10

    load 'MultipleUtilityI.mat';
    Fe = MultipleUtilityI.RandomFactors;
    Fe = ObserveEvidence(Fe, [3 1], 1);
    MultipleUtilityI.RandomFactors = Fe;
    [meu optdr] = OptimizeLinearExpectations(MultipleUtilityI);
    out = SerializeMEUOptimization(meu, optdr);
    clear MultipleUtilityI;
    clear Fe;

  end
% end of output function.
end


function out = SerializeFloat( x )
  out = sprintf('%.4f', x);
end

function out = SerializeTreeFg(C)
% Serializes a factor struct array into the .fg format for libDAI
% http://cs.ru.nl/~jorism/libDAI/doc/fileformats.html
%
% To avoid incompatibilities with EOL markers, make sure you write the
% string to a file using the appropriate file type ('wt' for windows, 'w'
% for unix)
totalLines = length(C.nodes) + length(C.edges(1,:)) + 3;

lines = cell(totalLines, 1);

newInd = 1;
lines{newInd} = sprintf('%d\n', numel(C.nodes));

for i = 1 : length(C.nodes),
    newInd = newInd + 1;
    lines{newInd} = sprintf('%s\n', num2str(C.nodes{i}));
end


newInd = newInd + 1;
lines{newInd} = sprintf('\n%d\n', numel(C.edges(1,:)));

for j = 1 : length(C.edges),
    newInd = newInd + 1;
    lines{newInd} = sprintf('%s\n', num2str(C.edges(j, :)));
end


lines{newInd + 1} = SerializeFactorsFg(C.factorList);

out = sprintf('%s', lines{:});

end

function out = SerializeCompactTree(C)
% Serializes a factor struct array into the .fg format for libDAI
% http://cs.ru.nl/~jorism/libDAI/doc/fileformats.html
%
% To avoid incompatibilities with EOL markers, make sure you write the
% string to a file using the appropriate file type ('wt' for windows, 'w'
% for unix)
totalLines = length(C.edges(1,:)) + 2;
lines = cell(totalLines, 1);

newInd = 1;
lines{newInd} = sprintf('\n%d\n', numel(C.edges(1,:)));

for j = 1 : length(C.edges),
```

```
    newInd = newInd + 1;
    lines{newInd} = sprintf('%s\n', num2str(C.edges(j, :)));
end


lines{newInd + 1} = SerializeFactorsFg(C.cliqueList);
out = sprintf('%s', lines{:});

end

function out = SerializeFactorsFg(F)
% Serializes a factor struct array into the .fg format for libDAI
% http://cs.ru.nl/~jorism/libDAI/doc/fileformats.html
%
% To avoid incompatibilities with EOL markers, make sure you write the
% string to a file using the appropriate file type ('wt' for windows, 'w'
% for unix)

  lines = cell(5*numel(F) + 1, 1);

  lines{1} = sprintf('%d\n', numel(F));
  lineIdx = 2;
  for i = 1:numel(F)
    lines{lineIdx} = sprintf('\n%d\n', numel(F(i).var));
    lineIdx = lineIdx + 1;

    lines{lineIdx} = sprintf('%s\n', num2str(F(i).var(:)')); % ensure that we put in a row vector
    lineIdx = lineIdx + 1;

    lines{lineIdx} = sprintf('%s\n', num2str(F(i).card(:)')); % ensure that we put in a row vector
    lineIdx = lineIdx + 1;

    lines{lineIdx} = sprintf('%d\n', numel(F(i).val));
    lineIdx = lineIdx + 1;

    % Internal storage of factor vals is already in the same indexing order
    % as what libDAI expects, so we don't need to convert the indices.
    vals = [0:(numel(F(i).val) - 1); F(i).val(:)'];
    lines{lineIdx} = sprintf('%d␣%0.8g\n', vals);
    lineIdx = lineIdx + 1;
  end

  out = sprintf('%s', lines{:});

end


function out = SerializeMEUOptimization(meu, optdr)
  optdr = SortFactorVars(optdr);
  optdr_part = SerializeFactorsFg(optdr);
  out = sprintf('%s\n%.4f\n', optdr_part, meu);
end


function [P, messages] = UnserializeTreeAndMessagesFg(str)
    index = find(str == '#');
    P = UnserializeCompactTree(str(1:index-1));
    remaining = str(index+1 : length(str));

    factors = UnserializeFactorsFgOctave(remaining);
    s = (length(factors))^0.5;
    messages = reshape(factors, s,s);

end


function F = UnserializeFactorsFgOctave(str)
%UnserializeFactorsFg Converts a string representing factors in the libDAI
%.fg format into a struct array of factors
% http://cs.ru.nl/~jorism/libDAI/doc/fileformats.html
% Rewritten for Octave compatibility using strtok instead of textscan
% In Octave, double quoted strings allow for escape sequences!

[tok, str] = strtok(str);
numFactors = sscanf(tok, '%d');

while (~nnz(numFactors))
    [tok, str] = strtok(str, char(10));
    numFactors = sscanf(tok, '%d');
end
```

```
F(numFactors) = struct('var', [], 'card', [], 'val', []);

for i = 1:numFactors
  [tok, str] = strtok(str);
  numVar = sscanf(tok, '%d');

  F(i).var = zeros(1, numVar);
  F(i).card = zeros(1, numVar);

  for j = 1:numVar
    [tok, str] = strtok(str);
    F(i).var(j) = sscanf(tok, '%f');
  end

  for j = 1:numVar
    [tok, str] = strtok(str);
    F(i).card(j) = sscanf(tok, '%f');
  end

  [tok, str] = strtok(str);
  nnzX = sscanf(tok, '%d');

  % libDAI's .fg format assumes that non-specified entries are zeros.
  % In addition, although the ordering of values is the same as in our 228
  % factor format, the indices start from 0 in the .fg format.
  F(i).val = zeros(1, prod(F(i).card));

  for j = 1:nnzX
    [tok, str] = strtok(str);
    idx = sscanf(tok, '%d');

    [tok, str] = strtok(str);
    val = sscanf(tok, '%f');

    F(i).val(idx + 1) = val;
  end
end

end

function F = UnserializeFactorsFgMATLAB(str)
%UnserializeFactorsFg Converts a string representing factors in the libDAI
%.fg format into a struct array of factors
% http://cs.ru.nl/~jorism/libDAI/doc/fileformats.html

[numFactors, pos] = textscan(str, '%d', 1);
idx = pos;
numFactors = numFactors{1};



F(numFactors) = struct('var', [], 'card', [], 'val', []);

for i = 1:numFactors
  [numVar, pos] = textscan(str(idx+1:end), '%d', 1);
  idx = idx + pos;
  numVar = numVar{1};

  [var, pos] = textscan(str(idx+1:end), '%f', numVar);
  idx = idx + pos;

  [card, pos] = textscan(str(idx+1:end), '%f', numVar);
  idx = idx + pos;

  [nnz, pos] = textscan(str(idx+1:end), '%d', 1);
  idx = idx + pos;
  nnz = nnz{1};

  [entries, pos] = textscan(str(idx+1:end), '%d␣%f', nnz);
  idx = idx + pos;

  F(i).var = var{:}';
  F(i).card = card{:}';

  % libDAI's .fg format assumes that non-specified entries are zeros.
  % In addition, although the ordering of values is the same as in our 228
  % factor format, the indices start from 0 in the .fg format.
  F(i).val = zeros(prod(F(i).card), 1);
  F(i).val(entries{1} + 1) = entries{2};
```

```
  F(i).val = F(i).val';
end

end

function f = SortAllFactors(factors)

for i = 1:length(factors)
    factors(i) = SortFactorVars(factors(i));
end

varMat = vertcat(factors(:).var);
[unused, order] = sortrows(varMat);

f = factors(order);

end

function G = SortFactorVars(F)

[sortedVars, order] = sort(F.var);
G.var = sortedVars;

G.card = F.card(order);
G.val = zeros(numel(F.val), 1);

assignmentsInF = IndexToAssignment(1:numel(F.val), F.card);
assignmentsInG = assignmentsInF(:,order);
G.val(AssignmentToIndex(assignmentsInG, G.card)) = F.val;

end


function C = UnserializeTreeFg(str)
%UnserializeFactorsFg Converts a string representing factors in the libDAI
%.fg format into a struct array of factors
% http://cs.ru.nl/~jorism/libDAI/doc/fileformats.html
% Rewritten for Octave compatibility using strtok instead of textscan
% In Octave, double quoted strings allow for escape sequences!

[tok, str] = strtok(str, char(10));
numNodes = sscanf(tok, '%d');

if (~nnz(numNodes))
    [tok, str] = strtok(str, char(10));
    numNodes = sscanf(tok, '%d');
end

C.nodes = cell(1, numNodes);

for i = 1 : numNodes,
  [tok, str] = strtok(str, char(10));
  C.nodes{i} = str2num(tok);
end

[tok, str] = strtok(str, char(10));
numEdges = sscanf(tok, '%d');

if (~nnz(numEdges))
    [tok, str] = strtok(str, char(10));
    numEdges = sscanf(tok, '%d');
end
C.edges = zeros(numEdges, numEdges);

for i = 1 : numEdges,
    [tok, str] = strtok(str, char(10));
    C.edges(i,:) = str2num(tok);
end

C.factorList = UnserializeFactorsFgOctave(str);


end

function C = UnserializeCompactTree(str)
%UnserializeFactorsFg Converts a string representing factors in the libDAI
%.fg format into a struct array of factors
% http://cs.ru.nl/~jorism/libDAI/doc/fileformats.html
% Rewritten for Octave compatibility using strtok instead of textscan
% In Octave, double quoted strings allow for escape sequences!
```

```
[tok, str] = strtok(str);
numEdges = sscanf(tok, '%d');

C.edges = zeros(numEdges, numEdges);

for i = 1 : numEdges,

    [tok, str] = strtok(str, char(10));
    if (length(tok) == 1)
        % need to re-parse
        [tok, str] = strtok(str, char(10));
    end
    C.edges(i,:) = str2num(tok);
end

C.cliqueList = UnserializeFactorsFgOctave(str);


end



function url = site_url()
  url = 'http://class.coursera.org/pgm-2012-002';
end

function url = challenge_url()

  url = [site_url() '/assignment/challenge'];
end

function url = submit_url()
  url = [site_url() '/assignment/submit'];
end



% ========================= CHALLENGE HELPERS =========================

function src = source(partId)
  src = '';
  src_files = sources();
  if partId <= numel(src_files)
      flist = src_files{partId};
      for i = 1:numel(flist)
          fid = fopen(flist{i});
          if (fid == -1)
            error('Error opening %s (is it missing?)', flist{i});
          end
          line = fgets(fid);
          while ischar(line)
            src = [src line];
            line = fgets(fid);
          end
          fclose(fid);
          src = [src '||||||||'];
      end
  end
end

function ret = isValidPartId(partId)
  partNames = validParts();
  ret = (~isempty(partId)) && (partId >= 1) && (partId <= numel(partNames) + 1);
end

function partId = promptPart()
  fprintf('== Select which part(s) to submit:\n', ...
          homework_id());
  partNames = validParts();
  srcFiles = sources();
  for i = 1:numel(partNames)
    fprintf('==  %d) %s [', i, partNames{i});
    fprintf(' %s ', srcFiles{i}{:});
    fprintf(']\n');
  end
  fprintf('==  %d) All of the above \n==\nEnter your choice [1-%d]: ', ...
          numel(partNames) + 1, numel(partNames) + 1);
  selPart = input('', 's');
  partId = str2num(selPart);
```

```
  if ~isValidPartId(partId)
    partId = -1;
  end
end

function [email,ch,signature,auxstring] = getChallenge(email, part)
  str = urlread(challenge_url(), 'post', {'email_address', email, 'assignment_part_sid', [homework_id() '-'
      num2str(part)], 'response_encoding', 'delim'});

  str = strtrim(str);
  r = struct;
  while(numel(str) > 0)
    [f, str] = strtok (str, '|');
    [v, str] = strtok (str, '|');
    r = setfield(r, f, v);
  end

  email = getfield(r, 'email_address');
  ch = getfield(r, 'challenge_key');
  signature = getfield(r, 'state');
  auxstring = getfield(r, 'challenge_aux_data');
end

function [result, str] = submitSolutionWeb(email, part, output, source)

  result = ['{"assignment_part_sid":"' base64encode([homework_id() '-' num2str(part)], '') '",' ...
            '"email_address":"' base64encode(email, '') '",' ...
            '"submission":"' base64encode(output, '') '",' ...
            '"submission_aux":"' base64encode(source, '') '"' ...
            '}'];
  str = 'Web-submission';
end

function [result, str] = submitSolution(email, ch_resp, part, output, ...
                                        source, signature)

  params = {'assignment_part_sid', [homework_id() '-' num2str(part)], ...
            'email_address', email, ...
            'submission', base64encode(output, ''), ...
            'submission_aux', base64encode(source, ''), ...
            'challenge_response', ch_resp, ...
            'state', signature};

  str = urlread(submit_url(), 'post', params);

  % Parse str to read for success / failure
  result = 0;

end

% =========================== LOGIN HELPERS ===========================

function [login password] = loginPrompt()
  % Prompt for password
  [login password] = basicPrompt();

  if isempty(login) || isempty(password)
    login = []; password = [];
  end
end


function [login password] = basicPrompt()
  login = input('Login␣(Email␣address):␣', 's');
  password = input('Submission␣Password␣(from␣Assignments␣page):␣', 's');
end

function [login password] = quickLogin(login,password)
  cont_token = input(['You␣are␣currently␣logged␣in␣as␣' login '.\nIs␣this␣you?␣(y/n␣-␣type␣n␣to␣reenter␣password)
      '],'s');
  if(isempty(cont_token) || cont_token(1)=='Y'||cont_token(1)=='y')
    return;
  else
    [login password] = loginPrompt();
  end
end

function [str] = challengeResponse(email, passwd, challenge)
  str = sha1([challenge passwd]);
end
```

```
% ================================ SHA-1 ================================

function hash = sha1(str)

  % Initialize variables
  h0 = uint32(1732584193);
  h1 = uint32(4023233417);
  h2 = uint32(2562383102);
  h3 = uint32(271733878);
  h4 = uint32(3285377520);

  % Convert to word array
  strlen = numel(str);

  % Break string into chars and append the bit 1 to the message
  mC = [double(str) 128];
  mC = [mC zeros(1, 4-mod(numel(mC), 4), 'uint8')];

  numB = strlen * 8;
  if exist('idivide')
    numC = idivide(uint32(numB + 65), 512, 'ceil');
  else
    numC = ceil(double(numB + 65)/512);
  end
  numW = numC * 16;
  mW = zeros(numW, 1, 'uint32');

  idx = 1;
  for i = 1:4:strlen + 1
    mW(idx) = bitor(bitor(bitor( ...
                  bitshift(uint32(mC(i)), 24), ...
                  bitshift(uint32(mC(i+1)), 16)), ...
                  bitshift(uint32(mC(i+2)), 8)), ...
                  uint32(mC(i+3)));
    idx = idx + 1;
  end

  % Append length of message
  mW(numW - 1) = uint32(bitshift(uint64(numB), -32));
  mW(numW) = uint32(bitshift(bitshift(uint64(numB), 32), -32));

  % Process the message in successive 512-bit chs
  for cId = 1 : double(numC)
    cSt = (cId - 1) * 16 + 1;
    cEnd = cId * 16;
    ch = mW(cSt : cEnd);

    % Extend the sixteen 32-bit words into eighty 32-bit words
    for j = 17 : 80
      ch(j) = ch(j - 3);
      ch(j) = bitxor(ch(j), ch(j - 8));
      ch(j) = bitxor(ch(j), ch(j - 14));
      ch(j) = bitxor(ch(j), ch(j - 16));
      ch(j) = bitrotate(ch(j), 1);
    end

    % Initialize hash value for this ch
    a = h0;
    b = h1;
    c = h2;
    d = h3;
    e = h4;

    % Main loop
    for i = 1 : 80
      if(i >= 1 && i <= 20)
        f = bitor(bitand(b, c), bitand(bitcmp(b), d));
        k = uint32(1518500249);
      elseif(i >= 21 && i <= 40)
        f = bitxor(bitxor(b, c), d);
        k = uint32(1859775393);
      elseif(i >= 41 && i <= 60)
        f = bitor(bitor(bitand(b, c), bitand(b, d)), bitand(c, d));
        k = uint32(2400959708);
      elseif(i >= 61 && i <= 80)
        f = bitxor(bitxor(b, c), d);
        k = uint32(3395469782);
      end
```

```
      t = bitrotate(a, 5);
      t = bitadd(t, f);
      t = bitadd(t, e);
      t = bitadd(t, k);
      t = bitadd(t, ch(i));
      e = d;
      d = c;
      c = bitrotate(b, 30);
      b = a;
      a = t;

    end
    h0 = bitadd(h0, a);
    h1 = bitadd(h1, b);
    h2 = bitadd(h2, c);
    h3 = bitadd(h3, d);
    h4 = bitadd(h4, e);

  end

  hash = reshape(dec2hex(double([h0 h1 h2 h3 h4]), 8)', [1 40]);

  hash = lower(hash);

end

function ret = bitadd(iA, iB)
  ret = double(iA) + double(iB);
  ret = bitset(ret, 33, 0);
  ret = uint32(ret);
end

function ret = bitrotate(iA, places)
  t = bitshift(iA, places - 32);
  ret = bitshift(iA, places);
  ret = bitor(ret, t);
end

% ============================ Base64 Encoder ============================
% Thanks to Peter John Acklam
%

function y = base64encode(x, eol)
%BASE64ENCODE Perform base64 encoding on a string.
%
%   BASE64ENCODE(STR, EOL) encode the given string STR.  EOL is the line ending
%   sequence to use; it is optional and defaults to '\n' (ASCII decimal 10).
%   The returned encoded string is broken into lines of no more than 76
%   characters each, and each line will end with EOL unless it is empty.  Let
%   EOL be empty if you do not want the encoded string broken into lines.
%
%   STR and EOL don't have to be strings (i.e., char arrays).  The only
%   requirement is that they are vectors containing values in the range 0-255.
%
%   This function may be used to encode strings into the Base64 encoding
%   specified in RFC 2045 - MIME (Multipurpose Internet Mail Extensions).  The
%   Base64 encoding is designed to represent arbitrary sequences of octets in a
%   form that need not be humanly readable.  A 65-character subset
%   ([A-Za-z0-9+/=]) of US-ASCII is used, enabling 6 bits to be represented per
%   printable character.
%
%   Examples
%   --------
%
%   If you want to encode a large file, you should encode it in chunks that are
%   a multiple of 57 bytes.  This ensures that the base64 lines line up and
%   that you do not end up with padding in the middle.  57 bytes of data fills
%   one complete base64 line (76 == 57*4/3):
%
%   If ifid and ofid are two file identifiers opened for reading and writing,
%   respectively, then you can base64 encode the data with
%
%      while ~feof(ifid)
%         fwrite(ofid, base64encode(fread(ifid, 60*57)));
%      end
%
%   or, if you have enough memory,
%
%      fwrite(ofid, base64encode(fread(ifid)));
```

```
%
%   See also BASE64DECODE.

%   Author:      Peter John Acklam
%   Time-stamp:  2004-02-03 21:36:56 +0100
%   E-mail:      pjacklam@online.no
%   URL:         http://home.online.no/~pjacklam

   if isnumeric(x)
      x = num2str(x);
   end

   % make sure we have the EOL value
   if nargin < 2
      eol = sprintf('\n');
   else
      if sum(size(eol) > 1) > 1
         error('EOL must be a vector.');
      end
      if any(eol(:) > 255)
         error('EOL can not contain values larger than 255.');
      end
   end

   if sum(size(x) > 1) > 1
      error('STR must be a vector.');
   end

   x   = uint8(x);
   eol = uint8(eol);

   ndbytes = length(x);                % number of decoded bytes
   nchunks = ceil(ndbytes / 3);        % number of chunks/groups
   nebytes = 4 * nchunks;              % number of encoded bytes

   % add padding if necessary, to make the length of x a multiple of 3
   if rem(ndbytes, 3)
      x(end+1 : 3*nchunks) = 0;
   end

   x = reshape(x, [3, nchunks]);       % reshape the data
   y = repmat(uint8(0), 4, nchunks);   % for the encoded data

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Split up every 3 bytes into 4 pieces
   %
   %    aaaaaabb bbbbcccc ccdddddd
   %
   % to form
   %
   %    00aaaaaa 00bbbbbb 00cccccc 00dddddd
   %
   y(1,:) = bitshift(x(1,:), -2);                  % 6 highest bits of x(1,:)

   y(2,:) = bitshift(bitand(x(1,:), 3), 4);        % 2 lowest bits of x(1,:)
   y(2,:) = bitor(y(2,:), bitshift(x(2,:), -4));   % 4 highest bits of x(2,:)

   y(3,:) = bitshift(bitand(x(2,:), 15), 2);       % 4 lowest bits of x(2,:)
   y(3,:) = bitor(y(3,:), bitshift(x(3,:), -6));   % 2 highest bits of x(3,:)

   y(4,:) = bitand(x(3,:), 63);                    % 6 lowest bits of x(3,:)

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Now perform the following mapping
   %
   %   0  - 25  ->  A-Z
   %   26 - 51  ->  a-z
   %   52 - 61  ->  0-9
   %   62       ->  +
   %   63       ->  /
   %
   % We could use a mapping vector like
   %
   %   ['A':'Z', 'a':'z', '0':'9', '+/']
   %
   % but that would require an index vector of class double.
   %
   z = repmat(uint8(0), size(y));
   i =           y <= 25;  z(i) = 'A'      + double(y(i));
   i = 26 <= y & y <= 51;  z(i) = 'a' - 26 + double(y(i));
```

```
    i = 52 <= y & y <= 61;  z(i) = '0' - 52 + double(y(i));
    i =           y == 62;  z(i) = '+';
    i =           y == 63;  z(i) = '/';
    y = z;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Add padding if necessary.
    %
    npbytes = 3 * nchunks - ndbytes;      % number of padding bytes
    if npbytes
       y(end-npbytes+1 : end) = '=';      % '=' is used for padding
    end

    if isempty(eol)

       % reshape to a row vector
       y = reshape(y, [1, nebytes]);

    else

       nlines = ceil(nebytes / 76);       % number of lines
       neolbytes = length(eol);           % number of bytes in eol string

       % pad data so it becomes a multiple of 76 elements
       y = [y(:) ; zeros(76 * nlines - numel(y), 1)];
       y(nebytes + 1 : 76 * nlines) = 0;
       y = reshape(y, 76, nlines);

       % insert eol strings
       eol = eol(:);
       y(end + 1 : end + neolbytes, :) = eol(:, ones(1, nlines));

       % remove padding, but keep the last eol string
       m = nebytes + neolbytes * (nlines - 1);
       n = (76+neolbytes)*nlines - neolbytes;
       y(m+1 : n) = '';

       % extract and reshape to row vector
       y = reshape(y, 1, m+neolbytes);

    end

    % output is a character array
    y = char(y);

end
```

# 23 ./submitWeb.m

```
% submitWeb Creates files from your code and output for web submission.
%
%   If the submit function does not work for you, use the web-submission mechanism.
%   Call this function to produce a file for the part you wish to submit. Then,
%   submit the file to the class servers using the "Web Submission" button on the
%   Programming Assignments page on the course website.
%
% Copyright (C) Daphne Koller, Stanford University, 2012

function submitWeb(partId)
  if ~exist('partId', 'var') || isempty(partId)
    partId = [];
  end

  submit(partId, 1);
end
```

# 24 ./TestCases.m

```
% Copyright (C) Daphne Koller, Stanford University, 2012

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Test case 1 - a very simple influence diagram in which X1 is a random variable
% and D is a decision.  The utility U is a function of X1 and D.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

X1 = struct('var', [1], 'card', [2], 'val', [7, 3]);
X1.val = X1.val / sum(X1.val);
D = struct('var', [2], 'card', [2], 'val', [1 0]);
```

```
U1 = struct('var', [1, 2], 'card', [2, 2], 'val', [10, 1, 5, 1]);

I1.RandomFactors = X1;
I1.DecisionFactors = D;
I1.UtilityFactors = U1;

% All possible decision rules.
D1 = D;
D2 = D;
D2.val = [0 1];
AllDs = [D1 D2];

allEU = zeros(length(AllDs),1);
for i=1:length(AllDs)
  I1.DecisionFactors = AllDs(i);
  allEU(i) = SimpleCalcExpectedUtility(I1);
end

% OUTPUT
% allEU => [7.3000, 3.8000]

% Get EUF...
euf = CalculateExpectedUtilityFactor(I1);
% PrintFactor(euf) =>
% 2
% 1      7.300000
% 2      3.800000

[meu optdr] = OptimizeMEU(I1)
[meu optdr] = OptimizeWithJointUtility(I1)
[meu optdr] = OptimizeLinearExpectations(I1)
% OUTPUT
% All should have the same results:
% meu => 7.3000
% PrintFactor(optdr) =>
%      2
%      1      1.000000
%      2      0.000000

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Test case 2 - Introduce a random variable node X3 between U and the
% variable X1.  The new random variable X3 has parents X1 and D.
% The utility now has parents D and X2.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Add node between 1 and 2 and the utility
X1 = struct('var', [1], 'card', [2], 'val', [7, 3]);
X1.val = X1.val / sum(X1.val);
D = struct('var', [2], 'card', [2], 'val', [1 0]);
X3 = struct('var', [3,1,2], 'card', [2,2,2], 'val', [4 4 1 1 1 1 4 4]);
X3 = CPDFromFactor(X3,3);

% U is now a function of 3 instead of 2.
U1 = struct('var', [2,3], 'card', [2, 2], 'val', [10, 1, 5, 1]);

I2.RandomFactors = [X1 X3];
I2.DecisionFactors = D;
I2.UtilityFactors = U1;

% All possible decision rules.
D1 = D;
D2 = D;
D2.val = [0 1];
AllDs = [D1 D2];

allEU = zeros(length(AllDs),1);
for i=1:length(AllDs)
  I2.DecisionFactors = AllDs(i);
  allEU(i) = SimpleCalcExpectedUtility(I2);
end
% OUTPUT
% allEU => [7.5000, 1.0000]

% Get EUF...
euf = CalculateExpectedUtilityFactor(I2);
% PrintFactor(euf) =>
% 2
% 1      7.500000
% 2      1.000000
```

```
[meu optdr] = OptimizeMEU(I2)
[meu optdr] = OptimizeWithJointUtility(I2)
[meu optdr] = OptimizeLinearExpectations(I2)
% OUTPUT
% meu => 7.5000
% PrintFactor(optdr) =>
% 2
% 1      1.000000
% 2      0.000000

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Test case 3 - Make D a function of X1.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
X1 = struct('var', [1], 'card', [2], 'val', [7, 3]);
X1.val = X1.val / sum(X1.val);
D = struct('var', [2,1], 'card', [2,2], 'val', [1,0,0,1]);
X3 = struct('var', [3,1,2], 'card', [2,2,2], 'val', [4 4 1 1 1 1 4 4]);
X3 = CPDFromFactor(X3,3);

% U is now a function of 3 instead of 2.
U1 = struct('var', [2,3], 'card', [2, 2], 'val', [10, 1, 5, 1]);

I3.RandomFactors = [X1 X3];
I3.DecisionFactors = D;
I3.UtilityFactors = U1;

% All possible decision rules
D1 = D;D2 = D;D3 = D;D4 = D;
D1.val = [1 0 1 0];
D2.val = [1 0 0 1];
D3.val = [0 1 1 0];
D4.val = [0 1 0 1];

AllDs = [D1 D2 D3 D4];
allEU = zeros(length(AllDs),1);
for i=1:length(AllDs)
  I3.DecisionFactors = AllDs(i);
  allEU(i) = SimpleCalcExpectedUtility(I3);
end

% Get EUF...
euf = CalculateExpectedUtilityFactor(I3);
% PrintFactor(euf) =>
% 1      2
% 1      1        5.250000
% 2      1        2.250000
% 1      2        0.700000
% 2      2        0.300000

[meu optdr] = OptimizeMEU(I3)
[meu optdr] = OptimizeWithJointUtility(I3)
[meu optdr] = OptimizeLinearExpectations(I3)

% OUTPUT
% allEU =
% 7.5000
% 5.5500
% 2.9500
% 1.0000
% meu = 7.5000
% PrintFactor(optdr) =>
% 1      2
% 1      1        1.000000
% 2      1        1.000000
% 1      2        0.000000
% 2      2        0.000000

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Test case 4 - Add another utility node that is a function of D
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
X1 = struct('var', [1], 'card', [2], 'val', [7, 3]);
X1.val = X1.val / sum(X1.val);
D = struct('var', [2,1], 'card', [2,2], 'val', [1,0,0,1]);
X3 = struct('var', [3,1,2], 'card', [2,2,2], 'val', [4 4 1 1 1 1 4 4]);
X3 = CPDFromFactor(X3,3);

% U is now a function of 3 instead of 2.
U1 = struct('var', [2,3], 'card', [2, 2], 'val', [10, 1, 5, 1]);
U2 = struct('var', [2], 'card', [2], 'val', [1, 10]);
```

```
I4.RandomFactors = [X1 X3];
I4.DecisionFactors = D;
I4.UtilityFactors = [U1 U2];

[meu optdr] = OptimizeWithJointUtility(I4)
[meu optdr] = OptimizeLinearExpectations(I4)
% OUTPUT
% meu => 11
% PrintFactor(optdr) =>
% 1     2
% 1     1       0.000000
% 2     1       0.000000
% 1     2       1.000000
% 2     2       1.000000
```

# 25 ./VariableElimination.m

```
% VariableElimination takes in a list of factors F and a list of variables to eliminate
% and returns the resulting factors after running sum-product to eliminate
% the given variables.  Note that it may return more than one
% factor.
%
%   Fnew = VariableElimination(F, Z)
%   F = list of factors
%   Z = list of variables to eliminate
%
% Copyright (C) Daphne Koller, Stanford University, 2012

function Fnew = VariableElimination(F, Z)

% List of all variables
V = unique([F(:).var]);

% Setting up the adjacency matrix.
edges = zeros(length(V));

for i = 1:length(F)
    for j = 1:length(F(i).var)
        for k = 1:length(F(i).var)
            edges(F(i).var(j), F(i).var(k)) = 1;
        end
    end
end

variablesConsidered = 0;

while variablesConsidered < length(Z)

    % Using Min-Neighbors where you prefer to eliminate the variable that has
    % the smallest number of edges connected to it.
    % Everytime you enter the loop, you look at the state of the graph and
    % pick the variable to be eliminated.
    bestVariable = 0;
    bestScore = inf;
    for i=1:length(Z)
      idx = Z(i);
      score = sum(edges(idx,:));
      if score > 0 && score < bestScore
        bestScore = score;
        bestVariable = idx;
      end
    end

    variablesConsidered = variablesConsidered + 1;
    [F, edges] = EliminateVar(F, edges, bestVariable);

end

Fnew = F;
```