Track-oriented multiple hypothesis tracker

Task: Write a MATLAB function that implements a n-object tracker using the track-oriented multiple hypothesis tracking algorithm.

Your implementation (for each filtering recursion) should follow these steps:

1. for each local hypothesis in each hypothesis tree: 1). implement ellipsoidal gating; 2). calculate missed detection and predicted likelihood for each measurement inside the gate and make sure to save these for future use; 3). create updated local hypotheses and make sure to save how these connects to the old hypotheses and to the new the measurements for future use;
2. for each predicted global hypothesis: 1). create 2D cost matrix; 2). obtain M best assignments using a provided M-best 2D assignment solver; 3). update global hypothesis look-up table according to the M best assignment matrices obtained and use your new local hypotheses indexing;
3. normalise global hypothesis weights and implement hypothesis reduction technique: pruning and capping;
4. prune local hypotheses that are not included in any of the global hypotheses;
5. Re-index global hypothesis look-up table;
6. extract object state estimates from the global hypothesis with the highest weight;
7. predict each local hypothesis in each hypothesis tree.

Example: re-index global hypothesis look-up table.

Suppose the global hypothesis look-up table before re-indexing is $[3, 1, 2; 2, 1, 3; 3, 2, 2]$. Then after re-indexing, the hypothesis look-up table may look like $[2, 1, 1; 1, 1, 2; 2, 2, 1]$.

The M-best 2D assignment solver has been provided as a reference function.

```
[col4rowBest,row4colBest,gainBest]=kBest2DAssign(C,k)
```

`KBEST2DASSIGN`: Find the k lowest cost 2D assignments for the two-dimensional assignment problem with a rectangular cost matrix `C`.

INPUT: `C`: A `numRowXnumCol` cost matrix.

OUTPUTS: `col4rowBest`: A `numRowXk` vector where the entry in each element is an assignment of the element in that row to a column. 0 entries signify unassigned rows.

`row4colbest`: A `numColXk` vector where the entry in each element is an assignment of the element in that column to a row. 0 entries signify unassigned columns.

`gainBest`: A `kX1` vector containing the sum of the values of the assigned elements in `C` for all of the hypotheses.

Note:

1. When constructing your 2D cost matrix, if measurement j does not fall inside the gate of object i, set the corresponding entry (i, j) to $+\infty$.
2. Set parameter k used in `kBest2DAssign` to `ceil(`w^h`*obj.reduction.M)`, where $\sum_{h \in H} w^h = 1$.
3. We can use gating to further group objects into subsets and process each subset independently. However, this is **NOT** covered in this task.

4. When normalising weights in logarithmic scale, you can call function normalizeLogWeights, which has also been provided as a reference function.

Hint:

1. If you want to apply a function to each element of a struct array, you can use MATLAB function `arrayfun`, which makes your implementation faster than using `for` loops.
2. When pruning low weight data association events, you can call the `hypothesisReduction.prune` method you have written in the first home assignment. You simply need to change the second input parameter from struct array to hypotheses indices. Similar trick can be applied to `hypothesisReduction.cap`.
3. When pruning local hypotheses and re-indexing the look-up table, you might find MATLAB command `unique` useful.

Files referenced:

- *GaussianDensity.m*
- *hypothesisReduction.m*
- *kBest2DAssign.m*
- *log_mvnpdf.m*
- *measmodel.m*
- *modelgen.m*
- *motionmodel.m*
- *normalizeLogWeights.m*

Note that `obj.density` is a function handle bound to MATLAB class `GaussianDensity`. For instance, you can simply call `obj.density.update` to perform a Kalman update instead of using your own code.

Function

Save Reset MATLAB Documentation (https://www.mathworks.com/help/)

```
classdef n_objectracker
    %N_OBJECTRACKER is a class containing functions to track n object in
    %clutter.
    %Model structures need to be called:
    %sensormodel: a structure specifies the sensor parameters
    %           P_D: object detection probability --- scalar
    %           lambda_c: average number of clutter measurements per time
    %           scan, Poisson distributed --- scalar
    %           pdf_c: clutter (Poisson) intensity --- scalar
    %           intensity_c: clutter (Poisson) intensity --- scalar
    %motionmodel: a structure specifies the motion model parameters
    %           d: object state dimension --- scalar
    %           F: function handle return transition/Jacobian matrix
    %           f: function handle return predicted object state
    %           Q: motion noise covariance matrix
    %measmodel: a structure specifies the measurement model parameters
    %           d: measurement dimension --- scalar
    %           H: function handle return transition/Jacobian matrix
```

```
    %           h: function handle return the observation of the object
    %           state
    %           R: measurement noise covariance matrix

    properties
        gating      %specify gating parameter
        reduction   %specify hypothesis reduction parameter
        density     %density class handle
    end

    methods

        function obj = initialize(obj,density_class_handle,P_G,m_d,w_min,mer
            %INITIATOR initializes n_objectracker class
            %INPUT: density_class_handle: density class handle
            %       P_D: object detection probability
            %       P_G: gating size in decimal --- scalar
            %       m_d: measurement dimension --- scalar
            %       wmin: allowed minimum hypothesis weight --- scalar
            %       merging_threshold: merging threshold --- scalar
            %       M: allowed maximum number of hypotheses --- scalar
            %OUTPUT:  obj.density: density class handle
            %         obj.gating.P_G: gating size in decimal --- scalar
            %         obj.gating.size: gating size --- scalar
            %         obj.reduction.w_min: allowed minimum hypothesis
            %         weight in logarithmic scale --- scalar
            %         obj.reduction.merging_threshold: merging threshold
            %         --- scalar
            %         obj.reduction.M: allowed maximum number of hypotheses
            %         --- scalar
            obj.density = density_class_handle;
            obj.gating.P_G = P_G;
            obj.gating.size = chi2inv(obj.gating.P_G,m_d);
            obj.reduction.w_min = log(w_min);
            obj.reduction.merging_threshold = merging_threshold;
            obj.reduction.M = M;
        end

        function estimates = TOMHT(obj, states, Z, sensormodel, motionmodel,
            %TOMHT tracks n object using track-oriented multi-hypothesis tra
            %INPUT: obj: an instantiation of n_objectracker class
            %       states: structure array of size (1, number of objects)
            %       with two fields:
            %                x: object initial state mean --- (object state
            %                dimension) x 1 vector
            %                P: object initial state covariance --- (object
            %                state dimension) x (object state dimension)
            %                matrix
            %       Z: cell array of size (total tracking time, 1), each
            %       cell stores measurements of size (measurement
            %       dimension) x (number of measurements at corresponding
            %       time step)
```

```
            %OUTPUT:estimates: cell array of size (total tracking time, 1),
            %       each cell stores estimated object state of size (object
            %       state dimension) x (number of objects)

        end
    end
end


```

Code to call your function

Reset

```
%Choose object detection probability
P_D = 0.9;
%Choose clutter rate
lambda_c = 10;

%Creat sensor model
range_c = [-1000 1000;-1000 1000];
sensor_model = modelgen.sensormodel(P_D,lambda_c,range_c);

%Creat linear motion model
T = 1;
sigma_q = 5;
motion_model = motionmodel.cvmodel(T,sigma_q);

%Create linear measurement model
sigma_r = 10;
meas_model = measmodel.cvmeasmodel(sigma_r);

%Creat ground truth model
nbirths = 5;
K = 20;
tbirth = zeros(nbirths,1);
tdeath = zeros(nbirths,1);

initial_state = repmat(struct('x',[],'P',eye(motion_model.d)),[1,nbirths]);

initial_state(1).x = [0; 0; 0; -10];        tbirth(1) = 1;   tdeath(1) = K;
initial_state(2).x = [400; -600; -10; 5];   tbirth(2) = 1;   tdeath(2) = K;
initial_state(3).x = [-800; -200; 20; -5];  tbirth(3) = 1;   tdeath(3) = K;
initial_state(4).x = [0; 0; 7.5; -5];       tbirth(4) = 1;   tdeath(4) = K;
initial_state(5).x = [-200; 800; -3; -15];  tbirth(5) = 1;   tdeath(5) = K;

%% Generate true object data (noisy or noiseless) and measurement data
ground_truth = modelgen.groundtruth(nbirths,[initial_state.x],tbirth,tdeath,
ifnoisy = 0;
objectdata = objectdatagen(ground_truth,motion_model,ifnoisy);
measdata = measdatagen(objectdata,sensor_model,meas_model);

```

```
%% N-object tracker parameter setting
P_G = 0.999;            %gating size in percentage
w_min = 1e-3;           %hypothesis pruning threshold
merging_threshold = 2;  %hypothesis merging threshold
M = 100;                %maximum number of hypotheses kept in MHT
density_class_handle = feval(@GaussianDensity);    %density class handle
tracker = n_objectracker();
tracker = tracker.initialize(density_class_handle,P_G,meas_model.d,w_min,mer

TOMHTestimates = TOMHT(tracker, initial_state, measdata, sensor_model, motic

figure
hold on
grid on

for i = 1:nbirths
    h1 = plot(cell2mat(cellfun(@(x) x(1,i), objectdata.X, 'UniformOutput', f
        cell2mat(cellfun(@(x) x(2,i), objectdata.X, 'UniformOutput', false))
    h4 = plot(cell2mat(cellfun(@(x) x(1,i), TOMHTestimates, 'UniformOutput',
        cell2mat(cellfun(@(x) x(2,i), TOMHTestimates, 'UniformOutput', false
end

xlabel('x'); ylabel('y')

xlim([-1000 1000])
ylim([-1000 1000])

legend([h1 h4],'Ground Truth','TOMHT Estimates', 'Location', 'best')

set(gca,'FontSize',12)
```

Run Function

Assessment:

Submit

Is TOMHT correctly implemented (Nonlinear motion/measurement model, high SNR)?

Is TOMHT correctly implemented (Nonlinear motion/measurement model, low SNR)?