PMBM filtering recursion

Task: Complete the code of the PMBM filter class, which contains necessary functions to implement a track-oriented Poisson multi-Bernoulli mixture (PMBM) filter:

1. Prediction of Bernoulli component.
2. Misdetection update of Bernoulli component.
3. Object detection update of Bernoulli component.
4. Prediction of Poisson Point Process (PPP).
5. Misdetection update of PPP.
6. Object detection update of PPP.
7. PMBM prediction.
8. PMBM update.
9. Object states extraction.

For task 4, i.e., prediction of PPP, your implementation should consist of the following steps:

1. Predict Poisson intensity for pre-existing objects.
2. Incorporate Poisson birth intensity into Poisson intensity for pre-existing objects.

For task 6, i.e., object detection update of PPP, your implementation should consist of the following steps:

1. For each mixture component in the PPP intensity, perform Kalman update and calculate the predicted likelihood for each detection inside the corresponding ellipsoidal gate.
2. Perform Gaussian moment matching for the updated object state densities resulted from being updated by the same detection.
3. The returned likelihood should be the sum of the predicted likelihoods calculated for each mixture component in the PPP intensity and the clutter intensity. (You can make use of the normalizeLogWeights function to achieve this.)
4. The returned existence probability of the Bernoulli component is the ratio between the sum of the predicted likelihoods and the returned likelihood. (Be careful that the returned existence probability is in decimal scale while the likelihoods you calculated beforehand are in logarithmic scale.)

For task 8, i.e., PMBM update, your implementation should consist of the following steps:

1. Perform ellipsoidal gating for each Bernoulli state density and each mixture component in the PPP intensity.
2. Bernoulli update. For each Bernoulli state density, create a misdetection hypothesis (Bernoulli component), and `m` object detection hypothesis (Bernoulli component), where `m` is the number of detections inside the ellipsoidal gate of the given state density.
3. Update PPP with detections. Note that for detections that are not inside the gate of undetected objects, create dummy Bernoulli components with existence probability `r = 0`; in this case, the corresponding likelihood is simply the clutter intensity.
4. For each global hypothesis, construct the corresponding cost matrix and use Murty's algorithm to obtain the `M` best global hypothesis with highest weights. Note that for detections that are only inside the gate of undetected objects, they do not need to be taken into account when forming the cost matrix.
5. Update PPP intensity with misdetection.
6. Update the global hypothesis look-up table.
7. Prune global hypotheses with small weights and cap the number.
8. Prune local hypotheses (or hypothesis trees) that do not appear in the maintained global hypotheses, and re-index the global hypothesis look-up table.

For task 9, i.e., object states extraction, your implementation should consist of the following steps:

1. Find the multi-Bernoulli with the highest weight.
2. Extract the mean of the object state density from Bernoulli components with probability of existence no less than a threshold.

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. It is assumed that the object survival probability `P_S` is constant.
2. We can use gating to further group objects into subsets and process each subset indepedently. However, this is **NOT** implemented in this task.
3. When normalising or summing weights in logarithmic scale, you can call function normalizeLogWeights, which has also been provided as a reference function.
4. When constructing the cost matrix, if measurement j does not fall inside the gate of object i, set the corresponding entry (i, j) to $+\infty$.
5. If the constructed cost matrix is empty, do not forget to consider the case that all the detected objects are missed detected.
6. Set parameter `k` used in kBest2DAssign to `ceil(`w^h`*obj.reduction.M)`, where w^h denotes the weight of global hypothesis h and it satisfies that $\sum_h w^h = 1$.
7. The hypothesis look-up table maintained in the track-oriented PMBM filter is similar to the one maintained in the track-oriented MHT. The difference is that the table in the PMBM filter can have entries with zero value.
8. Always append new hypothesis tree to the right side of the existing hypothesis trees. The same applies when you expand the hypothesis look-up table. This is in consistent with the video content and important for you to pass the tests.
9. When pruning local/global hypotheses, make sure that the number of rows of the global hypothesis table always matches the length of the global hypothesis weight vector, and that the number of columns of the global hypothesis table always matches the number of local hypothesis trees.

Example: re-index hypothesis look-up table.

Suppose the hypothesis look-up table before re-indexing is $[3,0,2;2,1,3;3,2,2]$. Then after re-indexing, the hypothesis look-up table may look like $[2,0,1;1,1,2;2,2,1]$.

Hint:

1. If you want to apply a function to each element of a struct/cell array, you can use MATLAB command `arrayfun/cellfun`, 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 also applies to `hypothesisReduction.cap`.
3. When re-indexing the look-up table, you might find MATLAB function `unique` useful.
4. For the maintainance of the hypothesis look-up table, you may take a look at the provided `recycling` function.

Files referenced:

- *`kBest2DAssign.m`*
- *`modelgen.m`*
- *`measmodel.m`*
- *`GaussianDensity.m`*
- *`hypothesisReduction.m`*
- *`motionmodel.m`*
- *`normalizeLogWeights.m`*
- *`log_mvnpdf.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 PMBMfilter
    %PMBMFILTER is a class containing necessary functions to implement the
    %PMBM filter
    %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: value of clutter pdf --- scalar
    %           intensity_c: Poisson clutter 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 target state
    %           R: measurement noise covariance matrix
    %       birthmodel: a structure array specifies the birth model (Gaussian
    %       mixture density) parameters --- (1 x number of birth components)
    %           w: weights of mixture components (in logarithm domain)
    %           x: mean of mixture components
    %           P: covariance of mixture components
    properties
        density %density class handle
        paras   %%parameters specify a PMBM
    end

    methods
        function obj = initialize(obj,density_class_handle,birthmodel)
```