```
function [mu, sigma2] = posteriorGaussian(mu_x, sigma2_x, y, sigma2_r)
%posteriorGaussian performs a single scalar measurement update with a
%measurement model which is simply "y = x + noise".
%
%Input
%   MU_P            The mean of the (Gaussian) prior density.
%   SIGMA2_P        The variance of the (Gaussian) prior density.
%   SIGMA2_R        The variance of the measurement noise.
%   Y               The given measurement.
%
%Output
%   MU              The mean of the (Gaussian) posterior distribution
%   SIGMA2          The variance of the (Gaussian) posterior
 distribution

%Your code here
% Firstly f(x|y) is proportional to p(y|x)p(x) = N(y; x, sigma_r)
 * N(x; mu_x, sigma_x) which is proportional to e^(-(y-x)^2/
(2*sigma_r^2)) * e^(-(x-mu_x)^2/(2*sigma_x^2))
% This can be the exponent can be solved using cos: ax^2+bx+c = a(a
+d)-e

denominator = 2*sigma2_x * sigma2_r;

c = denominator\(y*y.'*sigma2_x + mu_x*mu_x.'*sigma2_r)*1;
b = denominator\(2*y*sigma2_x + 2*mu_x*sigma2_r)*-1;
a = denominator\(2*(sigma2_x + sigma2_r)*1);
d = b/(a);

mu = -d;
sigma2 = a^-1;
end

function [ xy ] = sigmaEllipse2D( mu, Sigma, level, npoints )
%SIGMAELLIPSE2D generates x,y-points which lie on the ellipse
 describing
% a sigma level in the Gaussian density defined by mean and
 covariance.
%
%Input:
%   MU          [2 x 1] Mean of the Gaussian density
%   SIGMA       [2 x 2] Covariance matrix of the Gaussian density
%   LEVEL       Which sigma level curve to plot. Can take any positive
 value,
%               but common choices are 1, 2 or 3. Default = 3.
%   NPOINTS     Number of points on the ellipse to generate. Default =
 32.
%
%Output:
%   XY          [2 x npoints] matrix. First row holds x-coordinates,
 second
```

```
%               row holds the y-coordinates. First and last columns
 should
%               be the same point, to create a closed curve.


%Setting default values, in case only mu and Sigma are specified.
if nargin < 3
    level = 3;
end
if nargin < 4
    npoints = 32;
end

% Create a vector of angles. The angles are those to create the level
 curve of the distribution.
% The vector starts at 0 and ends at 0 in order to creat a full
 elipse. In between the zeros there are npoints-2 points.
fi = [0:2*pi/(npoints-1):2*pi-2*pi/(npoints-1) 0];

xy = level*sqrtm(Sigma)*[cos(fi); sin(fi)] + mu;

end

function [mu, Sigma] = jointGaussian(mu_x, sigma2_x, sigma2_r)
%jointGaussian calculates the joint Gaussian density as defined
%in problem 1.3a.
%
%Input
%   MU_X        Expected value of x
%   SIGMA2_X    Covariance of x
%   SIGMA2_R    Covariance of the noise r
%
%Output
%   MU          Mean of joint density
%   SIGMA       Covariance of joint density


%Your code here

% mu makes intuitive sense. The first variable is just x so it's mu is
 mu_x. The second variable is x+r and since it's mu is mu_x+mu_r, mu_r
 is 0 so it's mu is also mu_x.
mu = [mu_x; mu_x];

% Element 1,1 in the cov matrix is simply sigma2_x.
% Element 1,2 in the cov matrix is cov(x,y) = E[xy] - E[x]E[y] = E[x^2
 + xr] - 0. Since they're uncorrelated E[x^2 + xr] = E[x^2] = sigma2_x
% Element 2,1 in the cov matrix is cov(x,y) = E[xy] - E[x]E[y] = E[x^2
 + xr] - 0. Since they're uncorrelated E[x^2 + xr] = E[x^2] = sigma2_x
% Element 2,2 in the cov matrix is simply sigma2_x.
Sigma = [sigma2_x sigma2_x; sigma2_x sigma2_r+sigma2_x];

end
```

```
function [mu_y, Sigma_y] = affineGaussianTransform(mu_x, Sigma_x, A,
 b)
%affineTransformGauss calculates the mean and covariance of y, the
%transformed variable, exactly when the function, f, is defined as
%y = f(x) = Ax + b, where A is a matrix, b is a vector of the same
%dimensions as y, and x is a Gaussian random variable.
%
%Input
%   MU_X        [n x 1] Expected value of x.
%   SIGMA_X     [n x n] Covariance of x.
%   A           [m x n] Linear transform matrix.
%   B           [m x 1] Constant part of the affine transformation.
%
%Output
%   MU_Y        [m x 1] Expected value of y.
%   SIGMA_Y     [m x m] Covariance of y.

%Your code here

% This comes from the definitions.
% E[Y] = E[L*X+b] = L*E[X]+b
% E[(Y-mu_y)(Y-mu_y)^T] = E[(L*X+b-E[Y])(L*X+b-E[Y])^T] =
 A*Sigma_x*A^T

mu_y = A*mu_x + b;
Sigma_y =  A*Sigma_x*A.';

end

function [mu_y, Sigma_y, y_s] = approxGaussianTransform(mu_x, Sigma_x,
 f, N)
%approxGaussianTransform takes a Gaussian density and a transformation
%function and calculates the mean and covariance of the transformed
 density.
%
%Inputs
%   MU_X        [m x 1] Expected value of x.
%   SIGMA_X     [m x m] Covariance of x.
%   F           [Function handle] Function which maps a [m x 1]
 dimensional
%               vector into another vector of size [n x 1].
%   N           Number of samples to draw. Default = 5000.
%
%Output
%   MU_Y        [n x 1] Approximated mean of y.
%   SIGMA_Y     [n x n] Approximated covariance of y.
%   ys          [n x N] Samples propagated through f


if nargin < 4
    N = 5000;
end

%Your code here
```

```
% Create N random samples
x_s = mvnrnd(mu_x, Sigma_x, N).';
% Take these samples through the function
y_s = f(x_s);

% Calculate the sample mean
mu_y = (sum(y_s.').')/N;
% Calculate the sample covariance
Sigma_y = ((y_s-mu_y)*(y_s-mu_y).')/(N-1);

end

function [ xHat ] = gaussMixMMSEEst( w, mu, sigma2 )
%GAUSSMIXMMSEEST calculates the MMSE estimate from a Gaussian mixture
%density with multiple components.
%
%Input
%   W           Vector of all the weights
%   MU          Vector containing the means of all components
%   SIGMA2      Vector containing the variances of all components
%
%Output
%   xHat        MMSE estimate

%YOUR CODE HERE

% All the expected values are known and their coefficients w are also
 known.
% So to get xhat the expected values and their coefficients need to be
 mulitplied and then added.
xHat = sum(w.*mu);
end
```

Published with MATLAB® R2019b