SUNDAR MURUGAN RAMASWAMY
murugan@net.chalmers.se

ANALYSIS CODE:

```
clc
clear
close all
%% TASK 1 Transformation of gaussian random variables
mu_x = [10;0];
Sigma_x = [0.3,0;0,8];

A = [1,1;-1,1];
b = [0;0];

%% linear transform
linear_transform = @(x)[A*x];
N=100000;
[mu_aff, Sigma_aff] = affineGaussianTransform(mu_x,
Sigma_x, A, b);
[mu_appx, Sigma_appx, gauss_appx] =
approxGaussianTransform(mu_x, Sigma_x, linear_transform,
N);

xy1 = sigmaEllipse2D(mu_aff, Sigma_aff, 3);
xy2 = sigmaEllipse2D(mu_appx, Sigma_appx, 3);
figure;
plot(xy1(1,:), xy1(2,:),'color','red');
axis equal
hold on
plot(xy2(1,:), xy2(2,:),'color','green');
plot(mu_aff(1),mu_aff(2),'*','color','blue','LineWidth',3);
plot(mu_appx(1),mu_appx(2),'*','color','blue','LineWidth',3
);
plot(gauss_appx(1,:),gauss_appx(2,:),'.','color','yellow');
legend('sigma aff','sigma appx','mean aff','mean
appx','location','best');
hold off

%% non linear transform
non_linear_transform = @(x)[sqrt(x(1,:).^2 +
x(2,:).^2);atan2(x(2,:),x(1,:))];
N= 100000;
[mu_appx_nl, Sigma_appx_nl, gauss_appx_nl] =
approxGaussianTransform(mu_x, Sigma_x,
non_linear_transform, N);
```

```
xy_nl = sigmaEllipse2D(mu_appx_nl, Sigma_appx_nl, 3);

figure;
plot(xy_nl(1,:), xy_nl(2,:),'color','green');
axis equal
hold on
plot(mu_appx_nl(1),mu_appx_nl(2),'*','color','blue','LineWi
dth',3);
plot(gauss_appx_nl(1,:),gauss_appx_nl(2,:),'.','color','cya
n');
legend('Sigma appx','mean appx','location','best')
hold off

%% c for linear
Sigma_x_new = [0.3,0;0,0.1];
linear_transform = @(x)[A*x];

[mu_aff, Sigma_aff] = affineGaussianTransform(mu_x,
Sigma_x_new, A, b);
[mu_appx, Sigma_appx, gauss_appx] =
approxGaussianTransform(mu_x, Sigma_x_new,
linear_transform, 5000);

xy1 = sigmaEllipse2D(mu_aff, Sigma_aff, 3);
xy2 = sigmaEllipse2D(mu_appx, Sigma_appx, 3);
figure;
plot(xy1(1,:), xy1(2,:),'color','red');
axis equal
hold on
plot(xy2(1,:), xy2(2,:),'color','green');
plot(mu_aff(1),mu_aff(2),'*','color','blue','LineWidth',3);
plot(mu_appx(1),mu_appx(2),'*','color','blue','LineWidth',3
);
plot(gauss_appx(1,:),gauss_appx(2,:),'.','color','yellow');
legend('sigma aff','sigma appx','mean aff','mean
appx','location','best');
hold off

% non linear
non_linear_transform = @(x)[sqrt(x(1,:).^2 +
x(2,:).^2);atan2(x(2,:),x(1,:))];
```

```
[mu_appx_nl, Sigma_appx_nl, gauss_appx_nl] =
approxGaussianTransform(mu_x, Sigma_x_new,
non_linear_transform, 5000);

xy_nl = sigmaEllipse2D(mu_appx_nl, Sigma_appx_nl, 3);

figure;
plot(xy_nl(1,:), xy_nl(2,:),'color','green');
axis equal
hold on
plot(mu_appx_nl(1),mu_appx_nl(2),'*','color','blue','LineWi
dth',3);
plot(gauss_appx_nl(1,:),gauss_appx_nl(2,:),'.','color','cya
n');
legend('Sigma appx','mean appx','location','best')
hold off

%% TASK 2 VACATION IN TURKEY
mu_antalya =15;
mu_izmir =13;
Sigma_Snow =2.5^2;

y_antalya = 15;
y_izmir = 13;

mu_Anna = 0;
mu_Oguz = 0;

Sigma_Anna = 1.7^2;
Sigma_Oguz = 5.2^2;

[mean_antalya, sd_antalya] = jointGaussian(mu_antalya,
Sigma_Snow, Sigma_Anna);
[mean_izmir, sd_izmir] = jointGaussian(mu_izmir,
Sigma_Snow, Sigma_Oguz);

xy_antalya = sigmaEllipse2D(mean_antalya, sd_antalya, 3);
xy_izmir = sigmaEllipse2D(mean_izmir, sd_izmir, 3);

figure;
plot(xy_antalya(1,:), xy_antalya(2,:));
hold on
plot(mean_antalya(1), mean_antalya(2), 'x', 'color',
'blue')
```

```
axis equal
hold off
figure;
plot(xy_izmir(1,:), xy_izmir(2,:));
hold on
plot(mean_izmir(1), mean_izmir(2), 'x', 'color', 'red')
legend('Sigma Antalya', 'Sigma Izmir','mean antalya', 'mean
izmir')
hold off

[mu_antalya_Anna, Sigma_antalya_Anna] =
posteriorGaussian(mu_antalya, Sigma_Snow, y_antalya,
Sigma_Anna);
[mu_izmir_Oguz, Sigma_izmir_Oguz] =
posteriorGaussian(mu_izmir, Sigma_Snow, y_izmir,
Sigma_Oguz);

x = 5:0.05:30;
Antalya_Anna = normpdf(x, mu_antalya_Anna,
sqrt(Sigma_antalya_Anna));
Izmir_Oguz = normpdf(x, mu_izmir_Oguz,
sqrt(Sigma_izmir_Oguz));

figure;
plot(x, Antalya_Anna);
hold on
plot(x, Izmir_Oguz);
legend('p(x=Antalya|y=Anna)', 'p(x=Izmir|y=Oguz)')
hold off

%% TASK 3 MMSE AND MAP ESTIMATES
x = -18:0.05:18;

w_a = [0.2, 0.8];
mu_a = [1, 1];
Sigma_a = [0.5, 9];
PDF_a = w_a(1)*normpdf(x, mu_a(1), sqrt(Sigma_a(1))) +
w_a(2)*normpdf(x, mu_a(2), sqrt(Sigma_a(2)));
MMSE_a = gaussMixMMSEEst(w_a, mu_a, Sigma_a);
[max_a, index_a] = max(PDF_a);
MAP_a = x(index_a);
figure;
plot(x, PDF_a);
hold on
```

```
plot(MMSE_a, 0, 'o')
plot(MAP_a, 0, 'x')
legend('Posterior', 'MMSE', 'MAP')
hold off

w_b = [0.49, 0.51];
mu_b = [6, -6];
Sigma_b = [2, 2];

PDF_b = w_b(1)*normpdf(x, mu_b(1), sqrt(Sigma_b(1))) +
w_b(2)*normpdf(x, mu_b(2), sqrt(Sigma_b(2)));
MMSE_b = gaussMixMMSEEst(w_b, mu_b, Sigma_b);
[max_b, index_b] = max(PDF_b);
MAP_b = x(index_b);
figure;
plot(x, PDF_b);
hold on
plot(MMSE_b, 0, 'o')
plot(MAP_b, 0, 'x')
legend('Posterior', 'MMSE', 'MAP')
hold off

w_c = [0.4, 0.6];
mu_c = [1.5, 3];
Sigma_c = [2.5, 1.4];

PDF_c = w_c(1)*normpdf(x, mu_c(1), sqrt(Sigma_c(1))) +
w_c(2)*normpdf(x, mu_c(2), sqrt(Sigma_c(2)));
MMSE_c = gaussMixMMSEEst(w_c, mu_c, Sigma_c);
[max_c, index_c] = max(PDF_c);
MAP_c = x(index_c);
figure;
plot(x, PDF_c);
hold on
plot(MMSE_c, 0, 'o')
plot(MAP_c, 0, 'x')
legend('Posterior', 'MMSE', 'MAP')
hold off
```

SIGMA ELLIPSE 2D

```
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

%Your code here
xy = []
    for phi = linspace(0,2*pi,npoints)
        xy(:,end+1) = mu + level * sqrtm(Sigma) * [cos(phi)
sin(phi)]'
    end
```

```
end
```

AFFINE GAUSSIAN TRANSFORM:

```
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
mu_y = A * mu_x + b
Sigma_y = A * Sigma_x * A'

end
```

APPROX GAUSSIAN TRANSFORM:

```
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
x_s = mvnrnd(mu_x, Sigma_x, N)'
y_s = f(x_s)
mu_y = mean(y_s,2)
Sigma_y = 1/(N-1) * (y_s - mu_y) * (y_s - mu_y)'
end
```

POSTERIOR GAUSSIAN

```
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
mu=(mu_x*sigma2_r + sigma2_x*y)/(sigma2_r + sigma2_x)
sigma2=(1/sigma2_r + 1/sigma2_x)^-1
end
```

JOINT GAUSSIAN

```
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
A1=[1 0; 1 1];
b1=[0;0];
mu=A1*[mu_x; 0]+b1
Sigma=A1*blkdiag(sigma2_x,sigma2_r)*A1'


end
```

GAUSSIAN MMSE:

```
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
xHat = w(:)'*mu(:)
end
```