TTK4250 Sensor Fusion
Solution to Assignment 2

Task 1: *Bayesian estimation of an existence variable*

We are back at tracking the number of boats in a region. However, you now know that there is at most one boat in the region, and that it is there with probability $r_k \in (0, 1)$ at time step k. The boat will stay in the region with probability P_{S} and leave with probability $1 - P_{\mathrm{S}}$ if it is there. Otherwise, it may enter with probability P_E and not enter with probability $1 - P_E$ if it is not in the region. You have an unreliable measurement coming from a radar that detects a present boat with probability P_{D}, and may not detect a present boat with probability $1 - P_{\mathrm{D}}$. If it did not detect a present boat, it may declare that there is a boat present anyway, termed a false alarm, with probability P_{FA}. The task is to apply a Bayesian filter to this problem.

Hint: It might ease proper book keeping if you start by denoting the events with variables and use their (possibly conditional) PMFs before you insert the given probabilities.

(a) Apply the Bayes prediction step to get the predicted probability $r_{k+1|k}$ in terms of r_k, P_{S} and P_E.

Solution: Let us introduce the needed events for a particular time step k. The events are "in region" R_k, "staying" S_k and "entering" E_k, and we use $\neg$ in front of an event for the complementary event that it did not occur. We can see S_k and E_k as conditional events for R_{k+1} where $S_k = R_{k+1}|R_k$ and $E_k = R_{k+1}|\neg R_k$. With this we state the transition PMFs in terms of the given probabilities; $\Pr(R_k) = r_k$, $\Pr(\neg R_k) = 1 - r_k$, $\Pr(R_{k+1}|R_k) = P_{\mathrm{S}}$, $\Pr(\neg R_{k+1}|R_k) = 1 - P_{\mathrm{S}}$, $\Pr(R_{k+1}|\neg R_k) = P_E$, $\Pr(\neg R_{k+1}|\neg R_k) = 1 - P_E$. Thus using the prediction step of recursive Bayesian state estimation, which is simply an application of the total probability theorem, we get

$$\begin{aligned} r_{k+1|k} &= \Pr(R_{k+1}) = \Pr(R_{k+1}|R_k)\Pr(R_k) + \Pr(R_{k+1}|\neg R_k)\Pr(\neg R_k) \\ &= P_{\mathrm{S}} r_k + P_E(1 - r_k). \end{aligned}$$

We of course also have

$$\begin{aligned} 1 - r_{k+1|k} &= \Pr(\neg R_{k+1}) = \Pr(\neg R_{k+1}|R_k)\Pr(R_k) + \Pr(\neg R_{k+1}|\neg R_k)\Pr(\neg R_k) \\ &= (1 - P_{\mathrm{S}})r_k + (1 - P_E)(1 - r_k) \ = 1 - P_{\mathrm{S}} r_k - P_E(1 - r_k). \end{aligned}$$

(b) Apply the Bayes update step to get posterior probability for the boat being in the region, r_{k+1}, in terms of $r_{k+1|k}$, P_{D} and P_{FA}. That is, condition the probability on the measurement. There are two cases that needs to be considered; receiving a detection and not receiving a detection.

Solution: We still use the event R_k, but also introduce the new events D_k denoting the event of detecting a present boat, and F_k the event of getting a false alarm. The probabilities for the event are then described through $\Pr(D_k|R_k) = P_{\mathrm{D}}$, $\Pr(\neg D_k|R_k) = 1 - P_{\mathrm{D}}$, $\Pr(D_k|\neg R_k) = 0$, $\Pr(\neg D_k|\neg R_k) = 1$, $\Pr(F_k|D_k) = 0$, $\Pr(\neg F_k|D_k) = 1$, $\Pr(F_k|\neg D_k) = P_{FA}$ and $\Pr(\neg F_k|\neg D_k) =$

$1 - P_{FA}$.

In addition, we use the event $M_k = D_k \cup F_k$ to denote the sensor declaring a present boat, which has conditional probabilities

$$\begin{aligned}
\Pr(M_k|R_k) &= \Pr(D_k|R_k) + \Pr(F_k|\neg D_k, R_k)\Pr(\neg D_k|R_k) = P_{\mathrm{D}} + P_{FA}(1 - P_{\mathrm{D}}),\\
\Pr(M_k|\neg R_k) &= \Pr(D_k|\neg R_k) + \Pr(F_k|\neg D_k, \neg R_k)\Pr(\neg D_k|\neg R_k) = 0 + P_{FA} \cdot 1,\\
\Pr(\neg M_k|R_k) &= \Pr(\neg D_k \cap \neg F_k|R_k) = (1 - P_{FA})(1 - P_{\mathrm{D}}) = 1 - P_{\mathrm{D}} - P_{FA}(1 - P_{\mathrm{D}}),\\
\Pr(\neg M_k|\neg R_k) &= \Pr(\neg D_k \cap \neg F_k|\neg R_k) = (1 - P_{FA}).
\end{aligned}$$

We also need the marginal probabilities for M_k, which are

$$\begin{aligned}
\Pr(M_k) &= \Pr(M_k|R_k)\Pr(R_k) + \Pr(M_k|\neg R_k)\Pr(\neg R_k)\\
&= P_{\mathrm{D}} r_k + P_{FA}(1 - P_{\mathrm{D}})r_k + P_{FA}(1 - r_k) = P_{\mathrm{D}} r_k + P_{FA}(1 - P_{\mathrm{D}} r_k)\\
\Pr(\neg M_k) &= \Pr(\neg M_k|R_k)\Pr(R_k) + \Pr(\neg M_k|\neg R_k)\Pr(\neg R_k)\\
&= (1 - P_{FA})(1 - P_{\mathrm{D}})r_k + (1 - P_{FA})(1 - r_k)\\
&= (1 - P_{FA})(1 - P_{\mathrm{D}} r_k) = 1 - P_{\mathrm{D}} r_k - P_{FA}(1 - P_{\mathrm{D}} r_k)
\end{aligned}$$

Conditioning now becomes

$$\begin{aligned}
\hat{r}_{k+1}(M_{k+1}) &= \Pr(R_{k+1}|M_k) = \frac{\Pr(M_{k+1}, R_{k+1})}{\Pr(M_k)} = \frac{P_{\mathrm{D}} r_{k+1} + P_{FA}(1 - P_{\mathrm{D}})r_{k+1}}{P_{\mathrm{D}} r_{k+1} + P_{FA}(1 - P_{\mathrm{D}} r_{k+1})}\\
\hat{r}_{k+1}(\neg M_{k+1}) &= \Pr(R_{k+1}|\neg M_{k+1}) = \frac{\Pr(\neg M_{k+1}, R_{k+1})}{\Pr(\neg M_{k+1})} = \frac{(1 - P_{FA})(1 - P_{\mathrm{D}})r_{k+1}}{(1 - P_{FA})(1 - P_{\mathrm{D}} r_{k+1})}\\
&= \frac{(1 - P_{\mathrm{D}})r_{k+1}}{(1 - P_{\mathrm{D}} r_{k+1})}
\end{aligned}$$

Task 2: *KF initialization of CV model without prior knowledge*

The KF typically uses a prior for initializing the filter. However, in target tracking we often have no specific prior and would like to infere the initialization of the filter from the data. For the CV model with positional measurements, the position is observable with a single measurement, while the speed velocity needs two measurements to be observable (observable is here used in a statistical sense to mean that there is information about the state from the measurements). Since the KF is linear, we would like to use a linear initialization scheme that uses two measurements and the model parameters. That is

$$\hat{x}_1 = \begin{bmatrix} \hat{p}_1 \\ \hat{u}_1 \end{bmatrix} = \begin{bmatrix} K_{p_1} & K_{p_0} \\ K_{u_1} & K_{u_0} \end{bmatrix} \begin{bmatrix} z_1 \\ z_0 \end{bmatrix}, \tag{1}$$

where $z_k = \begin{bmatrix} I_2 & 0_2 \end{bmatrix} x_k + w_k = p_k + w_k$ with $x_k = \begin{bmatrix} p_k^T & u_k^T \end{bmatrix}^T$ and $w_k \sim \mathcal{N}(0, R)$. p_k is the position and u_k is the velocity at time step k.

(a) Write z_1 and z_0 as a function of the noises, true position and speed, p_1 and u_1, using the CV model with positional measurements. Use v_k to denote the process distrurbance, and w_k to denote the measurement noise at time step k.

Hint: A discrete time transition matrix is always invertible, and it is easy to find for the CV model (remove the process noise and think what you are left with).

Solution: Simply using the model gives us

$$z_1 = Hx_1 + w_1 = p_1 + w_1 \tag{2}$$
$$z_0 = Hx_0 + w_0 = HF^{-1}(x_1 - v_0) + w_0 = p_1 - Tu_1 + w_0 - HF^{-1}v_0 \tag{3}$$

(b) Show that to get an unbiased initial estimate, the initialization gain matrix must satisfy $K_{p_1} = I_2$, $K_{p_0} = 0_2$, $K_{u_1} = \frac{1}{T}I_2$ and $K_{u_0} = -\frac{1}{T}I_2$, where T is the sampling time. That is, find the $K_\times$ so that $E[\hat{x}_1] = x_1 = \begin{bmatrix} p_1 & u_1 \end{bmatrix}^T$

Solution: We need ("$\stackrel{!}{=}$" denotes that we want to enforce it)

$$E[\hat{p}_1] = E[K_{p_1}(p_1 + w_1) + K_{p_0}(p_1 - Tu_1 - HF^{-1}v_0 + w_0)] = K_{p_1}p_1 + K_{p_0}(p_1 - Tu_1) \stackrel{!}{=} p_1$$

and

$$E[\hat{u}_1] = E[K_{u_1}(p_1 + w_1) + K_{u_0}(p_1 - Tu_1 - HF^{-1}v_0 + w_0)] = K_{u_1}p_1 + K_{u_0}(p_1 - Tu_1) \stackrel{!}{=} u_1.$$

K_{p_0} has to be zero to make the position estimate independent of the speed of the true state, which leaves $K_{p_1} = I_2$. In order for the estimated speed to be independent of the true position we need $K_{u_1} = -K_{u_0}$. At last we see that to get $E[\hat{u}_1] = u_1$ the only option is having $K_{u_0} = \frac{1}{T}I_2$.

(c) What is the covariance of this estimate?

Solution:

$$\begin{aligned}
\operatorname{cov}[\hat{x}_1] &= E[(\hat{x}_1 - x_1)^T(\hat{x}_1 - x_1)] \\
&= E\left[\begin{bmatrix} w_1 \\ \frac{1}{T}(w_1 - w_0 + HF^{-1}v_0) \end{bmatrix}\begin{bmatrix} w_1 & \frac{1}{T}(w_1 - w_0 + HF^{-1}v_0) \end{bmatrix}\right] \\
&= E\begin{bmatrix} w_1 w_1^T & \frac{w_1(w_1 - w_0 + HF^{-1}v_0)^T}{T} \\ \frac{(w_1 - w_0 + HF^{-1}v_0)w_1^T}{T} & \frac{(w_1 - w_0 + HF^{-1}v_0)(w_1 - w_0 + HF^{-1}v_0)^T}{T^2} \end{bmatrix} \\
&= E\begin{bmatrix} w_1 w_1^T & \frac{w_1 w_1^T}{T} \\ \frac{w_1 w_1^T}{T} & \frac{w_1 w_1^T + w_0 w_0^T + HF^{-1}v_0 v_0^T (F^{-1})^T H^T}{T^2} \end{bmatrix} + \underbrace{E[\cdots]}_{=0} \\
&= \begin{bmatrix} R & \frac{1}{T}R \\ \frac{1}{T}R & \frac{1}{T^2}(2R + HF^{-1}Q(F^{-1})^T H^T) \end{bmatrix}
\end{aligned}$$

The hidden terms in the underbraced expectation is zero due to independence. Further we have $HF^{-1} = \begin{bmatrix} I_2 & TI_2 \end{bmatrix}$, which for the CV model lets us write

$$HF^{-1}Q(F^{-1})^T H^T = \begin{bmatrix} I_2 & TI_2 \end{bmatrix} \sigma_a^2 \begin{bmatrix} \frac{T^3}{3}I_2 & \frac{T^2}{2}I_2 \\ \frac{T^2}{2}I_2 & TI_2 \end{bmatrix} \begin{bmatrix} I_2 \\ TI_2 \end{bmatrix} = \sigma_a^2 \left(\frac{T^3}{3} - \frac{T^3}{2} - \frac{T^3}{2} + T^3 \right) I_2 = \sigma_a^2 \frac{T^3}{3} I_2.$$

This finally gives

$$\operatorname{cov}[\hat{x}] = \begin{bmatrix} R & \frac{1}{T}R \\ \frac{1}{T}R & \frac{2}{T^2}R + \sigma_a^2 \frac{T}{3} I_2 \end{bmatrix}.$$

(d) You have used this initialization scheme for your estimator and found a mean and covariance. What distribution does the true state have after this initialization? What are its parameters.

Hint: From equations you have already used, you can write x_1 in terms of $\hat{x}$ and disturbances and noises. You should be able to see the result as a linear transformation of some random variables. Note that $\hat{x}$ is given since the measurements are given and thus can be treated as a constant.

Solution: We follow the hint.

$$x_1 = \hat{x}_1 - \begin{bmatrix} w_1 \\ \frac{w_1 - w_0 + HF^{-1}v_0}{T}, \end{bmatrix}$$

and see that the RHS is clearly a linear combination of Gaussian random variables and a constant since $\hat{x}$ is given. We also see that $E[x_1|\hat{x}_1] = \hat{x}_1$ and $\text{cov}[x_1|\hat{x}_1] = \text{cov}[\hat{x}_1]$. Then, using the last two results, this makes $x \sim \mathcal{N}(\hat{x}, \text{cov}(\hat{x}))$.

(e) In theory, would you say that this initialization scheme is optimal or suboptimal, given that the model and two measurements is all we have? What would you say about its optimality in practice?

Solution: In theory, and according to our model, it is optimal. In practice, our model and its parameters are just approximations of the truth; the model is usually a simplification of reality, the parameters can be very good but almost certainly not perfect, and very few things are really truly perfectly stationary and Gaussian. In addition we will most likely have some information, however little, so that it is likely that we should be using a prior of some kind. These considerations tell us that it is likely not an initialization scheme that is as optimal as the theory suggests. However, in many cases this is a very simple, robust and good technique to use. Also, specifying another more complex – better, but probably still not optimal – model will not necessarily give much better results.

Task 3: *Implement EKF i MATLAB*

In MATLAB, Finish the following functions in the EKF skeleton found on blackboard.

You are free to change the prewritten code (for instance to optimize), but the function definitions must be the same for evaluation purposes. obj has functions as members: `.f(x, Ts), .F(x, Ts), .h(x), .H(x), .Q(x, Ts), .R`

(a) `[xp, Pp] = EKF.predict(obj, x, P)`

Solution:
```
xp = obj.f(x, Ts);
Fk = obj.F(x, Ts);
Pp = Fk * P * Fk' + obj.Q(x, Ts);
```

(b) `[vk, Sk]=EKF.innovation(obj, z, x, P)`

Solution:
```
vk = z - obj.h(x);
Hk = obj.H(x);
Sk = Hk * P * Hk' + obj.R;
```

(c) `[xupd, Pupd] = EKF.update(obj, z, x, P)`

Solution:
```
[vk, Sk] = obj.innovation(z, x, P);
Hk = obj.H(x);
Wk = P * Hk'/Sk;
xupd = x + Wk * vk;
I = eye(size(P));
Pupd = (I - Wk * Hk) * P * (I - Wk * Hk)' + Wk * obj.R * Wk';
```

(d) `NIS = EKF.NIS(obj, z, x, P)` (for parameter evaluation)

Solution:
```
[vk, Sk] = obj.innovation(z, x, P);
NIS = vk' * (Sk \ vk);
```

(e) `ll = EKF.loglikelihood(obj, z, x, P)` (for future use)

Solution: This is equivalent to implement the logarithm of the distribution of the innovation, $\log(\mathcal{N}(v_k; 0, HPH^T + R))$;
```
[vk, Sk] = obj.innovation(z, x, P);
NIS = obj.NIS(z, x, P);
ll = -0.5 * (NIS + log(det(2 * pi * Sk)))
```

Task 4: *Make CV model to use with the EKF class*

In MATLAB: Make a model structure that implements an EKF for the CV model to use with the EKF class you made in the last task. You can use the skeleton makeCVmodel function found on Blackboard. You have the standard model for a KF

$$x_k = Fx_{k-1} + v_{k-1}, \qquad v_{k-1} \sim \mathcal{N}(0; Q) \tag{4}$$
$$z_k = Hx_k + w_k, \qquad w_k \sim \mathcal{N}(0, R) \tag{5}$$

(a) Implement the prediction of the mean, `model.f(x, Ts)`.

Solution: `model.f = @(x, Ts) (eye(4) + Ts*diag([1; 1], 2))x;`

(b) Implement the Jacobian of f with respect to x, `model.F(x, Ts)`.

Solution: `model.F = @(x, Ts) (eye(4) + Ts*diag([1; 1], 2));`

(c) Implement the measurement prediction, `model.h(x)`.

Solution: `model.h = @(x) x(1:2);`

(d) Implement the Jacobian of h with respect to x, `model.H(x)`.

Solution: `model.H = @(x, Ts) [eye(2), zeros(2,2)];`

(e) Implement the process noise covariance matrix as a function of x, `model.Q(x, Ts)`.

Solution:
```
model.Q = @(x, Ts) q * ...
[Ts^3/3, 0, Ts^2/2, 0; 0, Ts^3/3, 0, Ts^2/2; Ts^2/2, 0, Ts, 0; 0, Ts^2/2, 0, Ts];
```

(f) Implement the measurement noise covariance matrix, `model.R(x, z)`.

Solution: `model.R = @(x, z) r * eye(2);`

Task 5: *Tuning of KF*

Tune your CV KF implementation to fit the given data. The data is created by simulating a CT model with $\sigma_a = 0.25^2 = 0.0625$ and $\sigma_\omega = \left(\frac{\pi}{15}\right) = 0.0439$. You can also generate new data if you want to test more.

The CV continuous time acceleration covariance will be denoted by q, and the measurement noise covariance by r in this task and code.

Hint: Section 4.6 - 4.8 will probably be helpfull.

(a) Simply tune q and r until you are satisfied

Solution: $r = 10$ and $q = 4$ seems to be giving good enough and smooth results. $r = 10$ and $q = 13$ gives the lowest RMSE but naturaly a more jagged estimate. The needed code for evaluationx can be written as

```
% set parameters
q = 4;
r = 10;

% create the model and estimator object
model = discreteCVmodel(q, r);
ekf = EKF(model);

% initialize
xbar(:, 1) = [0, 0, 1, 1]';
Pbar(:, : , 1) = diag([50, 50, 10, 10].^2);

% estimate
for k = 1:K
    [xhat(:, k), Phat(:, :, k)] = ...
        ekf.update(Z(:, k), xbar(:, k), Pbar(:, :, k));
    if k < K
        [xbar(:, k + 1), Pbar(:,:, k + 1)] = ...
            ekf.predict(xhat(:, k), Phat(:, :, k), Ts);
    end
end

% calculate a performance metric
```

```
posRMSE = sqrt(mean(sum((xhat(1:2, :) - Xgt(1:2, :)).^2, 1))); %
    position RMSE
velRMSE = sqrt(mean(sum((xhat(3:4, :) - Xgt(3:4, :)).^2, 1))); %
    velocity RMSE
```

(b) Make a grid for q and r and plot NIS using `surf`. Use the distribution of NIS to find confidence intervals and plot the contour lines of your evaluated NIS for these intervals using `contour`. Would you do some corrections to your tuning from A? Do you expect NIS to be good for tuning a CV model to a CT trajectory?

Hint: Write help "the function" in MATLAB to get information on it. chi2inv might come in handy

Solution: We make a loggrid of 20 values in each parameter with the intervals $q \in [0.05, 25]$ and $r \in [0.1, 50]$. ANIS did not give a definitive answer but the contour plot of the confidence interval for $\alpha \in \{0.05, 0.95\}$ suggests for a narrow band of parameter values such that $r \approx a\frac{1}{q} + 8$ for some $a > 0$. $q = 6$ and $r = 10$ is in this confidence region and also in accordance with what was found in (a), altough other parameters are also valid.

Since we are not using the correct dynamic model, the ANIS will not be perfectly chi squared as we are not able to produce the correct innovation covariance, but should nevertheless be close enough since we at least have the correct assumption of additive Gaussian measurements noise with position measurements.

The missing code can be written as

```
% parameters for the parameter grid
Nvals = 20;
qlow = 0.05;
qhigh = 10;
rlow = 0.1;
rhigh = 100;

%[...]

% other values of interest that can be stored
innovs = zeros(Nvals, Nvals, 2, K);
innovCorr = zeros(Nvals, Nvals, 11, 4);
NormInnov = zeros(Nvals, Nvals, 2, K);
Ss = zeros(Nvals, Nvals, 2 ,2, K);

% initialize (the same for all parameters can be used)
xbar(:, 1) = [0, 0, 1, 1]';
Pbar(:, : , 1) = diag([50, 50, 10, 10].^2);

% run through the grid and estimate
for qi = 1:Nvals
    for ri = 1:Nvals
        model = discreteCVmodel(qs(qi), rs(ri));
        ekf = EKF(model);
        for k = 1:K
            NIS(qi, ri, k) = ...
```

```
            ekf.NIS(Z(:, k), xbar(:, k), Pbar(:, :, k));
        [xhat(:, k), Phat(:, :, k)] = ...
            ekf.update(Z(:, k), xbar(:, k), Pbar(:, :, k));
        NEES(qi, ri, k) = (Xgt(1:4, k) - xhat(:,k))' * ...
            (Phat(:, :, k) \ (Xgt(1:4, k) - xhat(:, k)));
        if k < K
            [xbar(:, k + 1), Pbar(:, :, k + 1)] = ...
                ekf.predict(xhat(:, k), Phat(:, :, k), Ts);
        end
    end

    % some other quantities of interest
    [innovs(qi, ri, :, k), Ss(qi, ri, :, :, k)] = ...
        ekf.innovation(Z(:,k), xbar(:, k), Pbar(:,:,k));
    NormInnov(qi, ri, :, k) = ...
        chol(squeeze(Ss(qi, ri, :, :, k)))' \ ...
            squeeze(innovs(qi, ri, :, k));
    innovCorr(qi, ri, :, :) = ...
        xcorr(squeeze(NormInnov(qi, ri, :, :))', 5);
  end
end

% calculate averages
ANEES = mean(NEES, 3);
ANIS = mean(NIS, 3);
Ainnov = mean(innovs, 4);

%[...]

alphas = [0.05, 0.95];
CINIS = chi2inv(alphas, 2*K)/K;

%[...]

surfc(qs,rs, ANIS'); hold on; % note transpose
contour3(qs, rs, ANIS', CINIS); % note transpose
```

(c) Do the same for NEES as you did with NIS. Would you tune differently now? Do you expect NEES to be good for tuning a CV model to a CT trajectory?

Solution: The ANEES is calculated over the same grid as ANIS. The confidence intervall contour plot also gives a similar shaped band of suggested values. The confidence interval contours of ANEES cross the confidence interval contours of ANIS. Wanting both to be consistent leaves a small region for us to choose from with $q \in [6.5, 7]$ and $r \in [9.5, 10.5]$ approximately. Therefore, for instance, a pick of $q = 6.7$ and $r = 10$ seem like a good option. Note that this might not always be the case, and we might have to choose between consistent ANIS and ANEES in real life applications.

For $|\omega| > 0$, the prediction is biased in both speed and position. The predicted covariance will be wrong since the CT model makes the positional covariances elongated in the across track

direction. The CV model will have to compensate for this by having a greater covariance to handle the maneuvers, but will then create a to large along track covariance. These effects will probably be more severe for NEES, as NIS is based on a correct measurement model that includes some noise that "vashes" out some of the discrepancies in the motion model. As such we expect NIS to be a more accurate tool in terms of closer to chi square. However, NEES gives a better view of the full state errors, so if ground truth is available it should at least be considered.

The code needed for calculating NEES is given above, while the code for ploting is as follows

```
alphas = [0.05, 0.95];
CINEES = chi2inv(alphas, 4*K)/K;
%[...]
surfc(qs ,rs, ANEES'); hold on; % note transpose
contour3(qs, rs, ANEES', CINEES); % note transpose
```

(d) Run your tuned filter on some new data. Are the results as expected? Why/why not?

Solution: It seems to be mostly doing a good job. The RMSE values are fluctuating quite a bit, and sometimes also the estimate. This is likely due to the discrepancy between the estimation and simulation model. One might have tuned differently if different sample data were considered.

Task 6: *Implement a SIR particle filter for a pendulum*

We want to estimate the position of a pendulum influenced by some random disturbance, with a suboptimal measuring device.

The pendulum dynamic equation is given by $\ddot{\theta} = \frac{g}{l}\sin(\theta) + F$, where g is the gravitational acceleration and l is the lenght from the center of rotation to the pendulum. The pendulum is assumed to be a point mass attached to a massless rod. F represents other forces that acts on the pendulum, here modeled as a stochastic angular acceleration process. θ is 0 when the rod is at the bottom.

A measuring device is set up L_d below the rotation point, and L_l to the left so that $z_k = h(\theta) = \sqrt{(L_d - \cos(theta))^2 + (\sin(\theta) - L_l)^2)} + w_k$, where $w_k \sim \text{triangle}(E[w_k], a)$, a symmetric triangle distribution with width $2a$, height $\frac{1}{a}$ and peak at $E[w_k]$.

A skeleton that creates data and provides some plot aid and so on can be found on Blackboard.

(a) Finish the particle filter. To finish it you need to implement particle prediction, weight update and resampling (algorithm 3). How does the filter perform?

Solution: It seems to perform OK with around 300 particles. The estimate is naturaly bi modal. More particles seems slightly better, less seems to (erroneously) loose the multimodality and by chance pick one of them. At the bottom the measurements are not very sensitive to the position, so the estimate becomes wide.

```
% number of particles to use
N = 300;

% initialize particles, pretend you do not know where the
   pendulum starts
px = [2*pi*(rand(1, N) - 0.5); randn(1, N)*pi/4]; % uniform,
   Gaussian
```

```
% initial weights
w = ones(N, 1)/N;

%[...]

for k = 1:K
   % weight update
   for n = 1:N
      w(n) = pdf(hpdf, Z(k) - h(px(:, n))); % write help pdf
   end
   w = w/sum(w);

   % resample
   cumw = cumsum(w);
   i = 1;
   for n  = 1:N
      u = (n - 1)/N;
      while u > cumw(i)
         i = i + 1;
      end
      pxn(:, n) = px(:, i);
   end

   % trajecory sample prediction
   for n = 1:N
      px(:, n) = discPendulum(pxn(:,n), random(PFfpdf), Ts);
   end

   %[...]
end
```

(b) Do not resample the simulated pendulum, but vary L_l and generate new measurements to see how it changes the estimate. Does it remove any problems in the estimation you found above?

Solution: Already at $L_l = 0.1$ the estimate seems to find the right mode after a little while, so this setup seems much better. This becomes more prominent for higher values of L_l.

(c) When would you choose to try a PF instead of a EKF and why?

Solution: If the model is highly nonlinear with high noise, multimodality and/or highly skewed we can assume that an EKF will struggle and a PF might do better. High noise in a highly nonlinear model means large covariances and large probability for the linarization at the mean to give poor description of the truth. A mean and covariance is not a very good descriptor of something that is multi modal. High skewness along with nonlinearities is an issue due to the EKF being symetric in the sense that it only estimates mean and covariance.

In this case we only had a minor nonlinearity but the estimation problem was inherently bi modal so we can assume that an EKF would have struggled, clinging onto one of the modes at

random.