732A96: Labs
Advanced Machine Learning

Carles Sans Fuentes

October 16, 2017

```
1
2  trans_probs <- diag(1/2, 10) +
3    diag(1/2, 10)[, c(10, 1:9)]
4  emission_probs <-
5    diag(1/5, 10)[, c(3:10, 1:2)] +
6    diag(1/5, 10)[, c(2:10, 1)] +
7    diag(1/5, 10) +
8    diag(1/5, 10)[, c(10, 1:9)] +
9    diag(1/5, 10)[, c(9:10, 1:8)]
10
11 emission_density <- function(x, z) {
12   return(emission_probs[z, x])
13 }
14
15 transition_density <- function(z, previous_z) {
16   return(trans_probs[previous_z, z])
17 }
18
19 transition_density2 <- function(z,previous_z){
20   if( z == zt){
21
22     return(0.5)
23
24   } else if( (z + 1) == zt){
25     return(0.5)
26   } else return(0)
27
28 }
29
30 get_alpha_scalar <- function(zt, xt, previous_alpha, previous_z) {
31   # Args:
32   #   zt Scalar, hidden state at which to compute alpha.
33   #   xt Scalar, observed state.
34   #   previous_alpha Vector, alpha for all z_{t-1}.
35   #   previous_z     Vector, all z_{t-1}.
36
37   summation_term <- 0
38   for (i in 1:length(previous_z)) {
39     summation_term <- summation_term +
40       previous_alpha[i] * transition_density(zt, previous_z[i])
41   }
42
43   alpha <- emission_density(xt, zt) * sum(summation_term)
44   return(alpha)
45 }
46
47 get_alpha <- function(Zt, xt, previous_alpha, previous_z) {
48   # Args:
49   #   Zt Vector, hidden states at which to compute alpha.
50   #   xt Scalar, observed state.
51   #   previous_alpha Vector, alpha for all z_{t-1}.
52   #   previous_z     Vector, all z_{t-1}.
53
54   alpha <- sapply(Zt, function(zt) {
55     get_alpha_scalar(zt, xt, previous_alpha, previous_z)
56   })
57
58   return(alpha)
59 }
60
61 get_beta_scalar <- function(zt, next_x, next_beta, next_z) {
62   # Args:
63   #   zt        Scalar, hidden state at which to compute alpha.
64   #   next_x    Scalar, observed next state.
65   #   next_beta Vector, alpha for all z_{t+1}.
66   #   next_z    Vector, all z_{t+1}.
```

```
67
68    summation_term <- 0
69    for (i in 1:length(next_z)) {
70      summation_term <- summation_term +
71        next_beta[i] * emission_density(next_x, next_z[i]) * transition_density(next_z[i], zt)
72    }
73
74    # P(z_(t+1) | z_t) =
75    # 0.5 if z_t = z_(t+1)
76    # 0.5 if z_t = z_t + 1
77    # 0 otherwise
78
79
80    return(summation_term)
81  }
82
83  get_beta <- function(Zt, next_x, next_beta, next_z) {
84    # Args:
85    #   Zt        Vector, hidden states at which to compute alpha.
86    #   next_x    Scalar, observed next state.
87    #   next_beta Vector, alpha for all z_{t+1}.
88    #   next_z    Vector, all z_{t+1}.
89
90    beta <- sapply(Zt, function(zt) {
91      get_beta_scalar(zt, next_x, next_beta, next_z)
92    })
93
94    return(beta)
95  }
96
97  fb_algorithm <- function(
98    observations,
99    emission_density,
100   transition_density,
101   possible_states,
102   initial_density) {
103
104   t_total <- length(observations)
105   cardinality <- length(possible_states)
106
107   # Alpha
108   alpha <- matrix(NA, ncol=cardinality, nrow=t_total)
109
110   for (i in 1:cardinality) {
111     alpha[1, i] <-
112       emission_density(observations[1], possible_states[i]) * initial_density[i]
113   }
114
115   for (t in 2:t_total) {
116     alpha[t, ] <- get_alpha(possible_states, observations[t], alpha[t - 1, ], possible_states)
117   }
118
119
120   # Beta
121   beta <- matrix(NA, ncol=cardinality, nrow=t_total)
122
123   beta[t_total, ] <- 1
124
125   for (t in (t_total - 1):1) {
126     beta[t, ] <- get_beta(possible_states, observations[t + 1], beta[t + 1, ], possible_states)
127   }
128
129   return(list(alpha = alpha, beta = beta))
130 }
131
132 filtering <- function(alpha) {
133   alpha / rowSums(alpha)
134 }
135
136 smoothing <- function(alpha, beta) {
137   alpha * beta / rowSums(alpha * beta)
138 }
139
140
141
142
143
144
145
146 robotHmm <- HMM::initHMM(
147   States = 1:10,
148   Symbols = 1:10,
149   transProbs = trans_probs,
150   emissionProbs = emission_probs
151 )
152
153 # Create a wrapper for simHMM to assign class to the output
```

```r
simHMM <- function(hmm, length) {
  simulation <- HMM::simHMM(hmm, length)
  return(structure(simulation, class="HmmSimulation"))
}

# Simulate
nSim <- 100
robotSimultation <- simHMM(hmm=robotHmm, length=nSim)

#debugonce(fb_algorithm)

alphabeta <- fb_algorithm(observations = robotSimultation$observation,
                          emission_density = emission_density,
                          transition_density = transition_density,
                          possible_states = 1:10,
                          initial_density = rep(0.1, 10))

filtering(alphabeta$alpha)
smoothing(alphabeta$alpha, alphabeta$beta)


plot(apply(filtering(alphabeta$alpha), 1, which.max), type = "l")
plot(apply(smoothing(alphabeta$alpha, alphabeta$beta), 1, which.max), type = "l")
lines(x = 1:100,robotSimultation$states, type = "l", col ="green")




# # # Test
# zt <- 5
# xt <- 6
# previous_alpha <- rep(0.1, 10)
# previous_z <- 1:10
# transition_density(zt, previous_z[5])
# get_alpha_scalar(zt, xt, previous_alpha, previous_z)


# # Test
# zt <- 1:10
# xt <- 6
# previous_alpha <- rep(0.1, 10)
# previous_z <- 1:10
# transition_density(zt, previous_z[5])
# get_alpha(zt, xt, previous_alpha, previous_z)
#


# # Test
# Zt <- 1:10
# next_x <- 6
# next_beta <- rep(0.1, 10)
# next_z <- 1:10
# transition_density(zt, previous_z[5])
# get_beta_scalar(5, next_x, next_beta, next_z)
# get_beta(zt, next_x, next_beta, next_z)


# Define the transition, emission and initialization probabilities --------

emission_probs <- matrix(c(.2, .2, .2, 0, 0, 0, 0, 0, .2, .2,
                           .2, .2, .2, .2, 0, 0, 0, 0, 0, .2,
                           .2, .2, .2, .2, .2, 0, 0, 0, 0, 0,
                           0, .2, .2, .2, .2, .2, 0, 0, 0, 0,
                           0, 0, .2, .2, .2, .2, .2, 0, 0, 0,
                           0, 0, 0, .2, .2, .2, .2, .2, 0, 0,
                           0, 0, 0, 0, .2, .2, .2, .2, .2, 0,
                           0, 0, 0, 0, 0, .2, .2, .2, .2, .2,
                           .2, 0, 0, 0, 0, 0, .2, .2, .2, .2,
                           .2, .2, 0, 0, 0, 0, 0, .2, .2, .2), byrow=TRUE, nrow=10)

transition_probs <- matrix(c(.5, .5, 0, 0, 0, 0, 0, 0, 0, 0,
                             0, .5, .5, 0, 0, 0, 0, 0, 0, 0,
                             0, 0, .5, .5, 0, 0, 0, 0, 0, 0,
                             0, 0, 0, .5, .5, 0, 0, 0, 0, 0,
                             0, 0, 0, 0, .5, .5, 0, 0, 0, 0,
                             0, 0, 0, 0, 0, .5, .5, 0, 0, 0,
                             0, 0, 0, 0, 0, 0, .5, .5, 0, 0,
                             0, 0, 0, 0, 0, 0, 0, .5, .5, 0,
                             0, 0, 0, 0, 0, 0, 0, 0, .5, .5,
                             .5, 0, 0, 0, 0, 0, 0, 0, 0, .5), byrow=TRUE, nrow=10)

tProbDensity <- function(zt, zt_1) {
  return(transition_probs[zt_1, zt])
}

eProbDensity <- function(xt, zt) {
  return(emission_probs[zt, xt])
```

```
}

initProbDensity <- function(z0) {
  return(dunif(z0, min=1, max=10))
}


# Simulate data ------------------------------------------------------------

library(HMM)

robotHmm <- HMM::initHMM(
  States = 1:10,
  Symbols = 1:10,
  transProbs = transition_probs,
  emissionProbs = emission_probs
)

simHMM <- function(hmm, length) {
  simulation <- HMM::simHMM(hmm, length)
  return(structure(simulation, class="HmmSimulation"))
}

nSim <- 100
robotSimultation <- simHMM(hmm=robotHmm, length=nSim)

X <- robotSimultation$observation
Z <- robotSimultation$states

# Implement Viterbi --------------------------------------------------------

possibleStates <- 1:10
get_omega <- function(Z, Omega, Z_next, x_next) {
  sapply(Z_next, function(z_next) {
    term1 <- log(eProbDensity(x_next, z_next))

    term2 <- sapply(Z, function(z) {
      log(tProbDensity(z_next, z))
    }) + Omega

    return(term1+ max(term2))
  })
}

get_phi <- function(Z, Z_next, Omega) {
  sapply(Z_next, function(z_next) {
    term <- sapply(Z, function(z) {
      log(tProbDensity(z_next, z))
    }) + Omega
    return(Z[which.max(term)])
  })
}

viterbi <- function(observations, possibleStates) {
  cardinality <- length(possibleStates)
  t_total <- length(observations)

  omega_0 <- vector("numeric", length = cardinality)
  for (i in 1:cardinality) {
    omega_0[i] <- log(initProbDensity(possibleStates[i])) + log(eProbDensity(observations[1],
        possibleStates[i]))
  }


  omega <- matrix(NA, nrow=t_total, ncol=cardinality)
  phi <- matrix(NA, nrow=t_total, ncol=cardinality)
  omega[1, ] <- omega_0

  for (i in 1:(t_total-1)) {
    omega[i+1, ] <- get_omega(possibleStates, omega[i, ], possibleStates, observations[i+1])
    phi[i+1, ] <- get_phi(possibleStates, possibleStates, omega[i, ])
  }

  mpp <- rep(NA, t_total)
  mpp[t_total] <- possibleStates[which.max(omega[t_total, ])]
  for (t in (t_total - 1):1) {
    mpp[t] <- phi[t + 1, possibleStates[mpp[t + 1]] == possibleStates]
  }

  return(list(path = mpp, omega = omega, phi = phi))

}

results <- viterbi(X, possibleStates)
results$path

results_HMM <- HMM::viterbi(robotHmm, X)
```

```
327
328 cbind(results$path, results_HMM)
```

732A96: Lab 1
Advanced Machine Learning

Carles Sans Fuentes

October 9, 2017

Assignment

Question 1

In this question we are asked to show that multiple runs of the hill-climbing algorithm can return non-equivalent graphs. The hill-climbing algorithm is is a local greedy search optimization technique which iteratively starts with an arbitrary solution to a problem, and then it tries to find a better solution by incrementally changing a single element of the solution, such that if the change produces a better solution, then this new one is taken for the next incremental change until no further improvements are found. Intuitively, it can be understood that different ending outcomes can be found from different initial points if the function is complex enough. In order to prove this, the asia dataset from the bnlearn package in R has been taken. Two different starts has been tried, specifying in one an initial connection to exist and in the second one just totally random. It can be seen in the information and graphs below (table and 1 that the graphs are not equivalent.

```
> mygraph
  Bayesian network learned via Score-based methods
  model:
    [partially directed graph]
  nodes:                                8
  arcs:                                 7
    undirected arcs:                    2
    directed arcs:                      5
  average markov blanket size:          2.25
  average neighbourhood size:           1.75
  average branching factor:             0.62
  learning algorithm:                   Hill-Climbing
  score:                                BIC (disc.)
  penalization coefficient:             4.258597
  tests used in the learning procedure: 77
  optimized:                            TRUE
> cmygraph
  Bayesian network learned via Score-based methods model:
    [partially directed graph]
  nodes:                                8
  arcs:                                 8
    undirected arcs:                    6
    directed arcs:                      2
  average markov blanket size:          2.25
  average neighbourhood size:           2.00
  average branching factor:             0.25
  learning algorithm:                   Hill-Climbing
  score:                                BIC (disc.)
  penalization coefficient:             4.258597
  tests used in the learning procedure: 82
  optimized:                            TRUE
> all.equal(mygraph, cmygraph)
[1] "Different number of directed/undirected arcs"
```

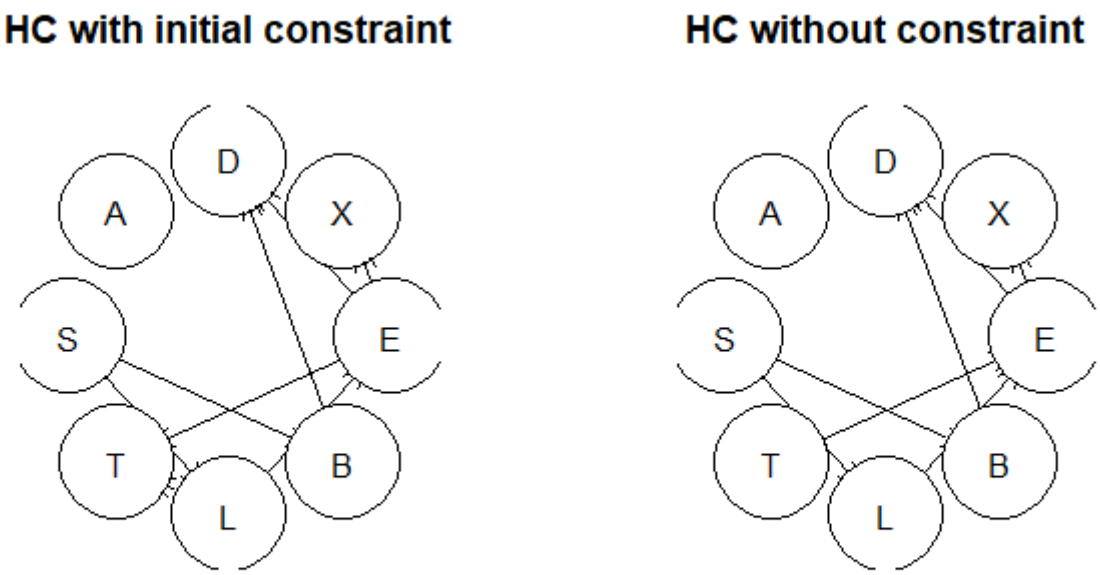

Figure 1: Graphical illustration of HC optimization for two different set ups

ALTERNATIVE SHOWING:

In order to show it in another way, I am going to use the data set alarm. I will be simulating from hill climbing as many times until it creates different graphs. And I will show this last ones graphically. This is what it can be seen in the following graph:

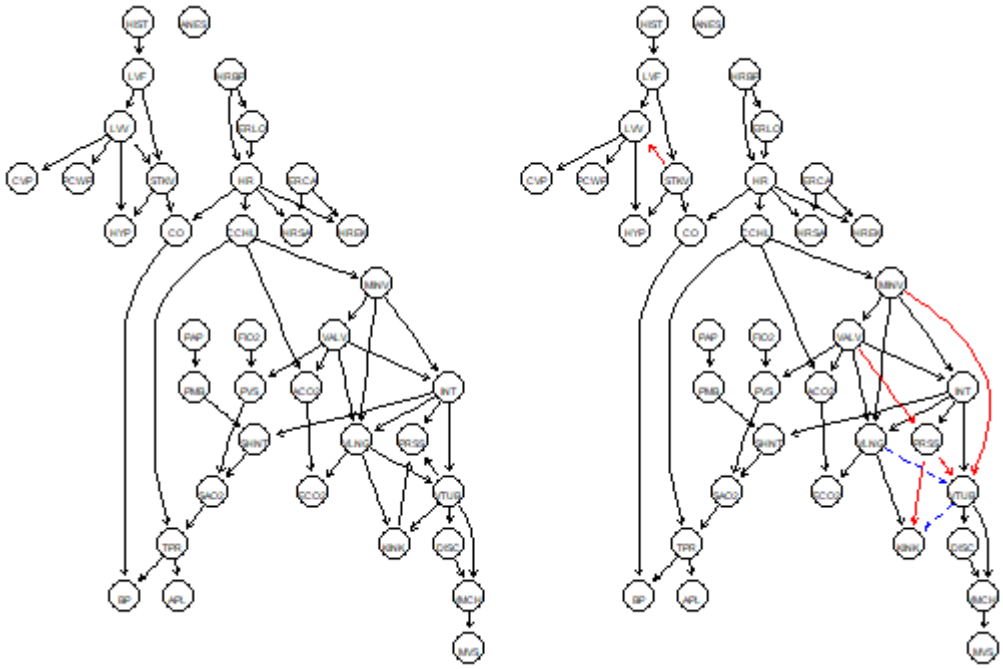

Figure 2: Graphical representation of the alarm data with different hc ending points

As explained previously, the hc() is a greedy algorithm which ends up in a local optima, so it tries to change arcs in order to get a better optima.

Question 2

In figure 3 it can be found the graphical proof that increasing the sample size the BDeu decreases regularization.

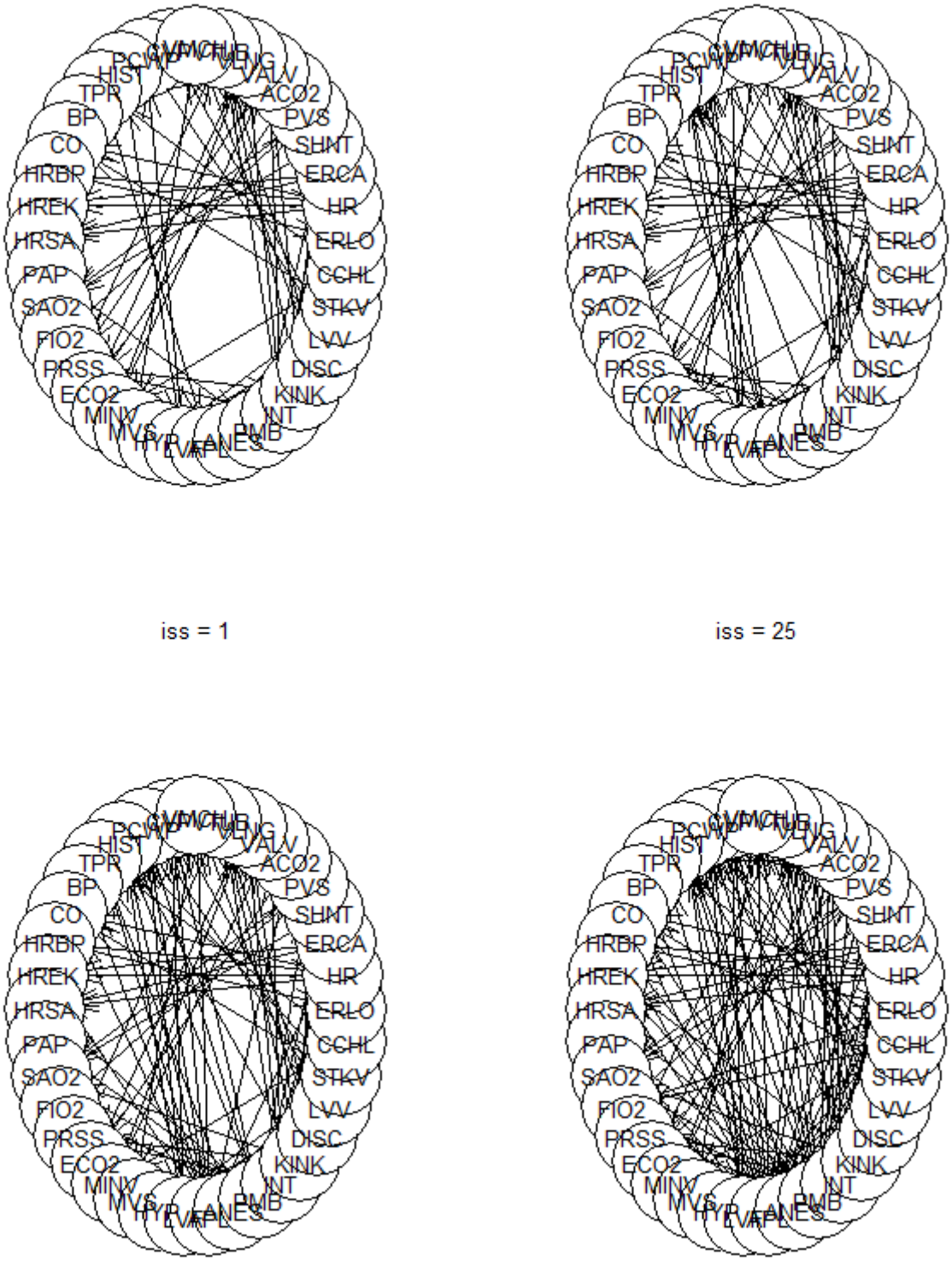

Figure 3: Graphical representation of the alarm data with different iss scores

The alpha.star results are the following ones:

```
[1] 6.779801 #for iss = 1
[1] 7.439901 #for iss = 25
[1] 10.34823 #for iss = 50
[1] 16.62416 #for iss = 100
```

The score got is lower because the more arc a graph has, the more likely to represent a graph from the data but the more complex it is so the more penalized the model is. Also, it is much probable that there will be a lot of parameters (e.g. arcs that do not exist in the real data) which will worsen the result.

Question 3

2nd rewritten EXPLANATION

Now we are asked to compare the answers given some queries by exact and approximate inference algorithms.

The approximate algorithm uses samples from the distribution while it discards the samples that are not consistent with the conditioning set. Now we will iterate 4 times on both algorithm in order to compare how well they perform. For that, I have used the dataset called data.learning and I have compared the LS and the approximate algorithm in order to find the conditional distribution of A, E given $B = b$ and $F = b$. and then further below it can be found the conditional distribution of A, E given $B = b$ and $F = b$, $D = b$ and $C = b$.

```
> res2
[[1]]
           A A_Approx.Freq          E E_Approx.Freq
a 0.07394366    0.06010929 0.3167939     0.3333333
b 0.64964789    0.65391621 0.3664122     0.3679417
c 0.27640845    0.28597450 0.3167939     0.2987250

[[2]]
           A A_Approx.Freq          E E_Approx.Freq
a 0.07394366    0.08791209 0.3167939     0.3150183
b 0.64964789    0.64285714 0.3664122     0.3791209
c 0.27640845    0.26923077 0.3167939     0.3058608

[[3]]
           A A_Approx.Freq          E E_Approx.Freq
a 0.07394366        0.0752 0.3167939        0.2992
b 0.64964789        0.6352 0.3664122        0.3840
c 0.27640845        0.2896 0.3167939        0.3168

[[4]]
           A A_Approx.Freq          E E_Approx.Freq
a 0.07394366    0.08448276 0.3167939     0.2948276
b 0.64964789    0.66724138 0.3664122     0.3862069
c 0.27640845    0.24827586 0.3167939     0.3189655
```

As previously mentioned, it can be found below the conditional distribution of A, E given $B = b$ and $F = b$, $D = b$ and $C = b$

```
> res5
[[1]]
           A A_Approx.Freq          E E_Approx.Freq
a 0.05115203    0.07142857 0.3167939     0.4285714
b 0.23753656    0.07142857 0.3664122     0.3214286
c 0.71131142    0.85714286 0.3167939     0.2500000

[[2]]
           A A_Approx.Freq          E E_Approx.Freq
a 0.05115203     0.0000000 0.3167939     0.3529412
b 0.23753656     0.1764706 0.3664122     0.2352941
c 0.71131142     0.8235294 0.3167939     0.4117647

[[3]]
           A A_Approx.Freq          E E_Approx.Freq
a 0.05115203    0.08333333 0.3167939     0.3333333
b 0.23753656    0.16666667 0.3664122     0.3333333
c 0.71131142    0.75000000 0.3167939     0.3333333

```

```
[[4]]
           A A_Approx.Freq          E E_Approx.Freq
a 0.05115203           0.05 0.3167939           0.40
b 0.23753656           0.15 0.3664122           0.25
c 0.71131142           0.80 0.3167939           0.35
```

Answer: As a result, the LS algorithm gives the same results every time (being exact) whereas the approximate one gives every time different result.

Comparing both algorithms, the approximate one performs worse when the conditioning set is large because it is based on first obtaining a sample from the distribution and then discarding the samples that are not consistent with the conditioning set (the less samples the higher variance). So, when the conditioning set is large, most of the samples are discarded leading to a poor approximation of the exact result.

In other words, the variance of the output prediction is greater the more nodes are observed because there is less data to calculate it, leading to a worse prediction of the approximate model compared to the exact one. Nevertheless, the goodness of the model is higher since the prediction will be more accurate.

Question 4

In this section We are asked to compute approximately the fraction of the 29281 DAGs that represent different independence models. In order to do so, I have used the random.graph function() sampling 10000 graphs. From these ones, I have check the unique ones and transformed them into DAGS. After that, I accounted for the the fraction of graphs resulted for different "burn.in" and "every" parameters getting the following results:

```
> result
            burn in = 1 burn in = 100 burn in = 10000 burn in = 1e+06
every = 2     0.6178384     0.6073096       0.6229086       0.6252186
every = 20    0.5471494     0.5539439       0.5519067       0.5567743
every = 100   0.5531273     0.5551819       0.5510988       0.5536594
every = 200   0.5454000     0.5534984       0.5536512       0.5552473
```

Results seem to show that the true proportion of essential graphs is approximately 0.55-0.6 of the DAGS. In light of the results, it is preferable to perform structure learning in the space of essential graphs for computation simplicity given that the result must be quite well accurate. The "Burn in" in this case does not change the result. These might mean that the process is directly stationary from the first observations. On the other side, the "every" parameter affects more to the output. By doing each observation less correlated with the previous ones (e.g. increasing the "every" parameter) the existent correlation through data tends to a certain level (e.g. in our case with 20 it already converges). leading to a more accurate fraction of the number of unique elements graphs.

Contributions

All results and comments presented have been developed and discussed together by the members of the group.

Appendix

Poisson regression-the MCMC way

```
1
2  ##Advanced Machine Learning lab 1
3
4
5  # source("http://bioconductor.org/biocLite.R")
6  # biocLite(c("graph", "RBGL", "Rgraphviz"))
7  #install.packages("gRain", dependencies=TRUE)
8  #install.packages("bnlearn")
9  #install.packages("gRbase")
10 #install.packages("gRain")
11 library(bnlearn)
12 library(gRain)
13 library(gRbase)
14
15 ###1
16 data(asia)
17
18 plot(asia)
19
20 par(mfrow=c(1,2))
21
22 wl = matrix(c("E", "T"), ncol = 2, byrow = TRUE,
23              dimnames = list(NULL, c("from", "to")))
24
25 plot(hc(asia, whitelist = wl), main ="HC with initial constraint")
26 plot(hc(asia, whitelist = NULL),main ="HC without constraint")
27
28 mygraph<-cpdag(hc(asia, whitelist = NULL, restart = 10))
29 cmygraph<-cpdag(hc(asia, whitelist = wl))
30 mygraph
31 cmygraph
32 all.equal(mygraph, cmygraph)
33
34 ##################Alternative
35 set.seed(12345)
36 countinue <- TRUE
37 while(countinue){
38   hc1 <- hc(alarm, restart = 10)
39   hc2 <- hc(alarm, restart = 10)
40   continue <- ifelse(all.equal(vstructs(hc1), vstructs(hc2)) == TRUE, TRUE, FALSE)
41   if (continue !=TRUE){
42     par(mfrow = c(1,2))
43     graphviz.compare(hc1, hc2)
44     par(mfrow = c(1,1))
45     break
46   }
47 }
48
49 ###2
50 data("alarm")
51 n<-4
52 iss<- c(1,25,50,100)
53 mydag<-list()
54 par(mfrow=c(2,2))
55 for(i in 1:n){
56   mydag[[i]]<- hc(alarm, restart = n,
57                   score = "bde", iss = iss[i])
58   plot(mydag[[i]], sub = paste0("iss = ", iss[i] ))
59   print(alpha.star(mydag[[i]], alarm, debug = FALSE))
60 }
61
62
63 par(mfrow=c(1,1))
64
65 ####3
66 data(learning.test)
67 pdag = iamb(learning.test)
68 pdag
69 plot(pdag)
70 dag = set.arc(pdag, from = "B", to = "A")
71 dag = pdag2dag(pdag, ordering = c("A", "B", "C", "D", "E", "F"))
72 plot(dag)
73 LS<- as.grain(fit)## it creates LS
74 plot(LS)
75 fit = bn.fit(dag, learning.test)##It creates all conditional tables from one edge with another
       with his possible outcomes
76
77 MM<- compile(LS)##it triangulates and moralizes
78 plot(MM)
79 basic<-querygrain(MM, nodes = c("A","E"))
80
```

```
####################
####################comparison of results for one change
## I want the conditional distribution of A, E given B = b

##Exact one
ChangeEvidence<-setEvidence(MM, c("B"), c("b"))
finalstate<-querygrain(ChangeEvidence, nodes = c("A","E"))

##Approximate algorithm
rsample<-cpdist(fit, nodes = c("A", "E"), evidence= (B=="b"), method = "ls")
condprob<-lapply(rsample, FUN = function(x){table(x)/length(x)})

res1<-cbind(A_LS= as.data.frame(finalstate)[1], A_Greedy= condprob$A,
      E_LS= as.data.frame(finalstate)[2], E_Greedy= condprob$E)
res1<-res1[,c(1,3,4,6)]
colnames(res1)<- c("A_LS", "A_Greedy","E_LS", "E_Greedy" )
res1

####################comparison of results for two changes
## I want the conditional distribution of A, E given B = b and F = b
ChangeEvidence2<-setEvidence(MM, c("B","F"), c("b", "b"))
finalstate2<-querygrain(ChangeEvidence2, nodes = c("A","E"))


res2<- list()
for(i in 1:4){
  ##Exact one
  ChangeEvidence2<-setEvidence(MM, c("B","F"), c("b", "b"))
  finalstate2<-querygrain(ChangeEvidence2, nodes = c("A","E"))
  ##Greedy algorithm
  rsample2<-cpdist(fit, nodes = c("A", "E"), evidence= (B=="b")&(F=="b"), method = "ls")
  condprob2<-lapply(rsample2, FUN = function(x){table(x)/length(x)})

  res2[[i]]<-cbind(A_LS= as.data.frame(finalstate2)[1], A_Approx= condprob2$A,
              E_LS= as.data.frame(finalstate2)[2], E_Approx= condprob2$E)
  res2[[i]]<-res2[[i]][,c(1,3,4,6)]

}
res2


###wITHOUT EVIDENCE
## I want the conditional distribution of A, E

##Exact one
finalstate3<-querygrain(MM, nodes = c("A","E"))


####################comparison of results for two changes
## I want the conditional distribution of A, E given B = b and F = b, D=d
ChangeEvidence4<-setEvidence(MM, c("B","F","D", "A"), c("b", "b","b", "b"))
finalstate4<-querygrain(ChangeEvidence4, nodes = c("A","E"))

i<-1
res5<- list()
for(i in 1:4){
  ##Exact one
  ChangeEvidence5<-setEvidence(MM, c("B","F","D", "C"), c("b", "b","b", "b"))
  finalstate5<-querygrain(ChangeEvidence5, nodes = c("A","E"))
  ##Greedy algorithm
  rsample5<-cpdist(fit, nodes = c("A", "E"), evidence= (B=="b")&(F=="b")&(D=="b")&(C=="b"),
       method = "ls")
  condprob5<-lapply(rsample5, FUN = function(x){table(x)/length(x)})

  res5[[i]]<-cbind(A_LS= data.frame(finalstate5)[1], A_Approx= data.frame(condprob5$A),
                   E_LS= data.frame(finalstate5)[2], E_Approx= data.frame(condprob5$E))
  res5[[i]]<-res5[[i]][,c(1,3,4,6)]

}
res5



###4
burn_in<- c(1,100, 10000, 1000000)
every<-c(2,20,100, 200)

nodes<- LETTERS[1:5]
num <- 1000


checkingDags<- function(burnin, every, num, nodes){
  mymat<- matrix(ncol =length(burnin), nrow=length(every))
  for(i in 1:length(burnin)){
    for(j in 1:length(every)){
      z<-random.graph(nodes = nodes, num = num,  method = "ic-dag",
                      every = every[j], burn.in =burnin[i])
```

```
167
168       m<-unique(z)
169       print(length(m))# This gives the proportion the number of repeated graphs out my
170
171       dagGraph<-lapply(m, FUN= function(x){cpdag(x)})
172       mymat[j,i]<- length(unique(dagGraph))/length(m)
173       }
174   }
175   return(mymat)
176 }
177
178 result<-checkingDags(burnin = burn_in, every = every, num = num, nodes = nodes)
179 colnames(result)<- paste0("burn in = " , burn_in)
180 rownames(result)<- paste0("every = " , every)
181 result
```

732A96: Lab 2
Advanced Machine Learning

Carles Sans Fuentes

September 21, 2017

Assignment

Main information about the lab : You do not have direct observation of the robot. However, the robot is equipped with a tracking device that you can access. The device is not very accurate though: If the robot is in the sector i, then the device will report that the robot is in the sectors $[i-2, i+2]$ with equal probability.

Question 1

Build a HMM for the scenario described above. The HMM has been built according to the description above. THe information about my HMM is shown below:

```
> myhmm
$States
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"

$Symbols
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"

$startProbs
  1   2   3   4   5   6   7   8   9  10
0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1

$transProbs
    to
from   1   2   3   4   5   6   7   8   9  10
  1  0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  2  0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  3  0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0
  4  0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0
  5  0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0
  6  0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0
  7  0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0
  8  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0
  9  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5
  10 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5

$emissionProbs
       symbols
states   1   2   3   4   5   6   7   8   9  10
    1  0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2
    2  0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2
    3  0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0
    4  0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0
    5  0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0
    6  0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0
    7  0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0
    8  0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2
    9  0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2
    10 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2
```

Question 2 & 3

We have Simulated the HMM for 100 time steps and discarded the hidden states from the sample obtained above. We have used the remaining observations to compute the filtered and smoothed probability distributions for each of the 100 time points. Compute also the most probable path. Here below I show the result.

```
> marginalFilter
      1         2     3   4   5         6         7         8         9 10 11  12   13        14
1   0.0 0.0000000 0.000 0.0 0.0 0.0000000 0.0000000 0.0000000 0.0000000  0  0 0.0 0.00 0.0000000
2   0.2 0.1111111 0.125 0.2 0.1 0.0000000 0.0000000 0.0000000 0.0000000  0  0 0.0 0.00 0.0000000
3   0.2 0.2222222 0.375 0.8 0.5 0.3157895 0.1578947 0.1090909 0.1818182  0  0 0.0 0.00 0.0000000
4   0.2 0.2222222 0.500 0.0 0.4 0.4736842 0.3947368 0.3818182 0.8181818  0  0 0.0 0.00 0.0000000
5   0.2 0.2222222 0.000 0.0 0.0 0.2105263 0.3421053 0.5090909 0.0000000  1  0 0.0 0.00 0.0000000
6   0.2 0.2222222 0.000 0.0 0.0 0.0000000 0.1052632 0.0000000 0.0000000  0  1 0.5 0.25 0.1428571
7   0.0 0.0000000 0.000 0.0 0.0 0.0000000 0.0000000 0.0000000 0.0000000  0  0 0.5 0.50 0.4285714
8   0.0 0.0000000 0.000 0.0 0.0 0.0000000 0.0000000 0.0000000 0.0000000  0  0 0.0 0.25 0.4285714
9   0.0 0.0000000 0.000 0.0 0.0 0.0000000 0.0000000 0.0000000 0.0000000  0  0 0.0 0.00 0.0000000
10  0.0 0.0000000 0.000 0.0 0.0 0.0000000 0.0000000 0.0000000 0.0000000  0  0 0.0 0.00 0.0000000
           15 16  17    18   19    20          21          22          23          24          25
1  0.00000000  0 0.0 0.25 0.00 0.375 0.53846154 0.46153846 0.40000000 0.28089888 0.18539326
2  0.00000000  0 0.0 0.00 0.00 0.000 0.00000000 0.26923077 0.42222222 0.41573034 0.34831461
3  0.00000000  0 0.0 0.00 0.00 0.000 0.00000000 0.00000000 0.00000000 0.21348315 0.31460674
4  0.00000000  0 0.0 0.00 0.00 0.000 0.00000000 0.00000000 0.00000000 0.00000000 0.10674157
5  0.00000000  0 0.0 0.00 0.00 0.000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
6  0.09090909  0 0.0 0.00 0.00 0.000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
7  0.36363636  0 0.0 0.00 0.00 0.000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
8  0.54545455  0 0.0 0.00 0.00 0.000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
9  0.00000000  1 0.5 0.25 0.25 0.125 0.07692308 0.03846154 0.02222222 0.00000000 0.00000000
10 0.00000000  0 0.5 0.50 0.75 0.500 0.38461538 0.23076923 0.15555556 0.08988764 0.04494382
           26         27          28          29          30          31          32          33
                34
1  0.00000000 0.0000000 0.00000000 0.00000000 0.00000000 0.00000000 0.01790763 0.098020735
     0.0000000
2  0.30944625 0.0000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.008953817
     0.0000000
3  0.38436482 0.0000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.000000000
     0.0000000
4  0.24429967 0.6307190 0.31535948 0.00000000 0.00000000 0.00000000 0.00000000 0.000000000
     0.0000000
5  0.06188925 0.3071895 0.46895425 0.00000000 0.00000000 0.00000000 0.00000000 0.000000000
     0.0000000
6  0.00000000 0.0620915 0.18464052 0.72595281 0.36297641 0.00000000 0.00000000 0.000000000
     0.0000000
7  0.00000000 0.0000000 0.03104575 0.23956443 0.48275862 0.00000000 0.00000000 0.000000000
     0.0000000
8  0.00000000 0.0000000 0.00000000 0.03448276 0.13702359 0.78325688 0.00000000 0.000000000
     0.0000000
9  0.00000000 0.0000000 0.00000000 0.00000000 0.01724138 0.19495413 0.80395853 0.401979265
     0.3104076
10 0.00000000 0.0000000 0.00000000 0.00000000 0.00000000 0.02178899 0.17813384 0.491046183
     0.6895924
           35          36          37          38          39       40         41         42         43
1  0.3447962 0.42239811 0.37500000 0.33585485 0.00000000 0.000000 0.0000000 0.0000000 0.0000000
2  0.0000000 0.17239811 0.29739811 0.39097545 0.43675896 0.351519 0.1757595 0.0000000 0.0000000
3  0.0000000 0.00000000 0.08619905 0.22304803 0.36897231 0.648481 0.5000000 0.3704333 0.0000000
4  0.0000000 0.00000000 0.00000000 0.05012167 0.16415016 0.000000 0.3242405 0.4518267 0.0000000
5  0.0000000 0.00000000 0.00000000 0.00000000 0.03011857 0.000000 0.0000000 0.1777400 0.7798358
6  0.0000000 0.00000000 0.00000000 0.00000000 0.00000000 0.000000 0.0000000 0.0000000 0.2201642
7  0.0000000 0.00000000 0.00000000 0.00000000 0.00000000 0.000000 0.0000000 0.0000000 0.0000000
8  0.0000000 0.00000000 0.00000000 0.00000000 0.00000000 0.000000 0.0000000 0.0000000 0.0000000
9  0.1552038 0.07760189 0.03880095 0.00000000 0.00000000 0.000000 0.0000000 0.0000000 0.0000000
10 0.5000000 0.32760189 0.20260189 0.00000000 0.00000000 0.000000 0.0000000 0.0000000 0.0000000
           44          45          46          47          48          49          50          51          52
1  0.0000000 0.00000000 0.0000000 0.0000000 0.00000000 0.0000000 0.0000000 0.1114899 0.23495034
2  0.0000000 0.00000000 0.0000000 0.0000000 0.00000000 0.0000000 0.0000000 0.0000000 0.00000000
3  0.0000000 0.00000000 0.0000000 0.0000000 0.00000000 0.0000000 0.0000000 0.0000000 0.00000000
4  0.0000000 0.00000000 0.0000000 0.0000000 0.00000000 0.0000000 0.0000000 0.0000000 0.00000000
5  0.3899179 0.19495896 0.1002381 0.0000000 0.00000000 0.0000000 0.0000000 0.0000000 0.00000000
6  0.5000000 0.44495896 0.3290136 0.0000000 0.00000000 0.0000000 0.0000000 0.0000000 0.00000000
7  0.1100821 0.30504104 0.3856122 0.4859713 0.24298567 0.1502614 0.0751307 0.0000000 0.00000000
8  0.0000000 0.05504104 0.1851361 0.3881294 0.43705039 0.4205317 0.2853965 0.1872996 0.09917849
9  0.0000000 0.00000000 0.0000000 0.1258992 0.25701433 0.4292069 0.4248693 0.3689943 0.29456761
10 0.0000000 0.00000000 0.0000000 0.0000000 0.06294961 0.0000000 0.2146035 0.3322162 0.37130356
           53          54          55          56          57          58          59          60          61
1  0.3434770 0.3603652 0.36128948 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.00000000
2  0.0000000 0.1717385 0.29104355 0.6914874 0.4088044 0.2044022 0.0000000 0.0000000 0.00000000
3  0.0000000 0.0000000 0.00000000 0.3085126 0.5911956 0.5000000 0.3922940 0.0000000 0.00000000
4  0.0000000 0.0000000 0.00000000 0.0000000 0.0000000 0.2955978 0.4430824 0.5196077 0.00000000
5  0.0000000 0.0000000 0.00000000 0.0000000 0.0000000 0.0000000 0.1646236 0.3779957 0.60632805
6  0.0000000 0.0000000 0.00000000 0.0000000 0.0000000 0.0000000 0.0000000 0.1023966 0.32450340
7  0.0000000 0.0000000 0.00000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.06916855
8  0.0561902 0.0280951 0.01536711 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.00000000
9  0.2230793 0.1396348 0.09174282 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.00000000
10 0.3772535 0.3001664 0.24055704 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.00000000
```

```
            62         63          64          65          66          67          68          69
                        70
1   0.00000000 0.0000000 0.00000000 0.00000000 0.00000000 0.04777084 0.16397296 0.2903185
      0.7192699
2   0.00000000 0.0000000 0.00000000 0.00000000 0.00000000 0.00000000 0.03609363 0.0000000
      0.2807301
3   0.00000000 0.0000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.0000000
      0.0000000
4   0.00000000 0.0000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.0000000
      0.0000000
5   0.30316403 0.1542493 0.07712466 0.00000000 0.00000000 0.00000000 0.00000000 0.0000000
      0.0000000
6   0.46541572 0.3910520 0.27265066 0.18190223 0.09236510 0.00000000 0.00000000 0.0000000
      0.0000000
7   0.19683597 0.3369525 0.36400224 0.33109421 0.26048590 0.18496778 0.00000000 0.0000000
      0.0000000
8   0.03458428 0.1177462 0.22734934 0.30753506 0.32427889 0.30653916 0.00000000 0.0000000
      0.0000000
9   0.00000000 0.0000000 0.05887310 0.14885128 0.23174080 0.29147071 0.45183108 0.2561633
      0.0000000
10  0.00000000 0.0000000 0.00000000 0.03061722 0.09112931 0.16925152 0.34810232 0.4535182
      0.0000000
          71 72 73   74 75   76         77         78   79   80      81 82 83   84    85          86
                       87
1   0.3596349  0   0 0.0   0 0.0 0.0000000 0.0000000 0.2  0.5 0.5625   1   1 0.5 0.25 0.0000000
      0.0000000
2   0.5000000  0   0 0.0   0 0.0 0.0000000 0.0000000 0.0  0.1 0.3750   0   0 0.5 0.50 0.4285714
      0.2307692
3   0.1403651  0   0 0.0   0 0.0 0.0000000 0.0000000 0.0  0.0 0.0625   0   0 0.0 0.25 0.4285714
      0.4615385
4   0.0000000  1   0 0.0   0 0.0 0.0000000 0.0000000 0.0  0.0 0.0000   0   0 0.0 0.00 0.1428571
      0.3076923
5   0.0000000  0   1 0.5   0 0.0 0.0000000 0.0000000 0.0  0.0 0.0000   0   0 0.0 0.00 0.0000000
      0.0000000
6   0.0000000  0   0 0.5   0 0.0 0.0000000 0.0000000 0.0  0.0 0.0000   0   0 0.0 0.00 0.0000000
      0.0000000
7   0.0000000  0   0 0.0   1 0.5 0.0000000 0.0000000 0.0  0.0 0.0000   0   0 0.0 0.00 0.0000000
      0.0000000
8   0.0000000  0   0 0.0   0 0.5 0.6666667 0.3333333 0.0  0.0 0.0000   0   0 0.0 0.00 0.0000000
      0.0000000
9   0.0000000  0   0 0.0   0 0.0 0.3333333 0.5000000 0.0  0.0 0.0000   0   0 0.0 0.00 0.0000000
      0.0000000
10  0.0000000  0   0 0.0   0 0.0 0.0000000 0.1666667 0.8  0.4 0.0000   0   0 0.0 0.00 0.0000000
      0.0000000
            88         89          90          91          92          93          94          95
                  96
1   0.0000000 0.0000000 0.00000000 0.00000000 0.00000000 0.00000000 0.0000000 0.0000000
      0.00000000
2   0.1153846 0.0000000 0.00000000 0.00000000 0.00000000 0.00000000 0.0000000 0.0000000
      0.00000000
3   0.3461538 0.0000000 0.00000000 0.00000000 0.00000000 0.00000000 0.0000000 0.0000000
      0.00000000
4   0.3846154 0.5135135 0.25675676 0.00000000 0.00000000 0.00000000 0.0000000 0.0000000
      0.00000000
5   0.1538462 0.3783784 0.44594595 0.40310078 0.20155039 0.10156250 0.2007722 0.1003861
      0.05019305
6   0.0000000 0.1081081 0.24324324 0.39534884 0.39922481 0.30273438 0.7992278 0.5000000
      0.30019305
7   0.0000000 0.0000000 0.05405405 0.17054264 0.28294574 0.34375000 0.0000000 0.3996139
      0.44980695
8   0.0000000 0.0000000 0.00000000 0.03100775 0.10077519 0.19335937 0.0000000 0.0000000
      0.19980695
9   0.0000000 0.0000000 0.00000000 0.00000000 0.01550388 0.05859375 0.0000000 0.0000000
      0.00000000
10  0.0000000 0.0000000 0.00000000 0.00000000 0.00000000 0.00000000 0.0000000 0.0000000
      0.00000000
            97          98          99        100
1   0.0000000 0.00000000 0.05711921 0.1837807
2   0.0000000 0.00000000 0.00000000 0.0000000
3   0.0000000 0.00000000 0.00000000 0.0000000
4   0.0000000 0.00000000 0.00000000 0.0000000
5   0.0000000 0.00000000 0.00000000 0.0000000
6   0.1797030 0.00000000 0.00000000 0.0000000
7   0.3846535 0.00000000 0.00000000 0.0000000
8   0.3331683 0.57154119 0.00000000 0.0000000
9   0.1024752 0.34686638 0.64293598 0.3309189
10  0.0000000 0.08159243 0.29994481 0.4853004
> smoothing
       index
states          1          2          3          4          5          6          7          8 9 10 11
       12
     1  0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0  0  0
          0.0
     2  0.8148148 0.6296296 0.4444444 0.2592593 0.1111111 0.0000000 0.0000000 0.0000000 0  0  0
          0.0
     3  0.1851852 0.3703704 0.5555556 0.7407407 0.7407407 0.6666667 0.4444444 0.2222222 0  0  0
          0.0
```

```
118     4  0.0000000 0.0000000 0.0000000 0.0000000 0.1481481 0.3333333 0.5555556 0.7777778 1  0  0
             0.0
119     5  0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0  1  0
             0.0
120     6  0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0  0  1
             0.5
121     7  0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0  0  0
             0.5
122     8  0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0  0  0
             0.0
123     9  0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0  0  0
             0.0
124     10 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0  0  0
             0.0
125       index
126 states         13   14 15 16         17         18         19          20          21           22
127     1  0.0000000 0.0  0  0 0.0000000 0.000000 0.0000000 0.28611141 0.66759329 0.633197262
128     2  0.0000000 0.0  0  0 0.0000000 0.000000 0.0000000 0.00000000 0.00000000 0.298228218
129     3  0.0000000 0.0  0  0 0.0000000 0.000000 0.0000000 0.00000000 0.00000000 0.000000000
130     4  0.0000000 0.0  0  0 0.0000000 0.000000 0.0000000 0.00000000 0.00000000 0.000000000
131     5  0.0000000 0.0  0  0 0.0000000 0.000000 0.0000000 0.00000000 0.00000000 0.000000000
132     6  0.1666667 0.0  0  0 0.0000000 0.000000 0.0000000 0.00000000 0.00000000 0.000000000
133     7  0.6666667 0.5  0  0 0.0000000 0.000000 0.0000000 0.00000000 0.00000000 0.000000000
134     8  0.1666667 0.5  1  0 0.0000000 0.000000 0.0000000 0.00000000 0.00000000 0.000000000
135     9  0.0000000 0.0  0  1 0.7452445 0.490489 0.2357335 0.07634843 0.01233386 0.001085725
136     10 0.0000000 0.0  0  0 0.2547555 0.509511 0.7642665 0.63754016 0.32007285 0.067488795
137       index
138 states           23          24          25          26          27          28          29           30
139     1  0.182923331 0.02714313 0.00000000 0.0000000 0.0000000 0.0000000 0.00000000 0.00000000
140     2  0.809476593 0.33583724 0.06731496 0.0000000 0.0000000 0.0000000 0.00000000 0.00000000
141     3  0.000000000 0.63701963 0.44749358 0.1281156 0.0000000 0.0000000 0.00000000 0.00000000
142     4  0.000000000 0.00000000 0.48519145 0.5178924 0.2095450 0.0000000 0.00000000 0.00000000
143     5  0.000000000 0.00000000 0.00000000 0.3539921 0.5470336 0.3116031 0.00000000 0.00000000
144     6  0.000000000 0.00000000 0.00000000 0.0000000 0.2434214 0.5349173 0.43429008 0.00000000
145     7  0.000000000 0.00000000 0.00000000 0.0000000 0.0000000 0.1534796 0.48154340 0.57760581
146     8  0.000000000 0.00000000 0.00000000 0.0000000 0.0000000 0.0000000 0.08416652 0.38691196
147     9  0.000000000 0.00000000 0.00000000 0.0000000 0.0000000 0.0000000 0.00000000 0.03548224
148     10 0.007600076 0.00000000 0.00000000 0.0000000 0.0000000 0.0000000 0.00000000 0.00000000
149       index
150 states           31          32          33          34          35          36          37          38
151     1  0.000000000 0.0000000 0.0000000 0.0000000 0.36216976 0.54445735 0.50117641 0.2756079
152     2  0.000000000 0.0000000 0.0000000 0.0000000 0.00000000 0.13995424 0.36920910 0.5777995
153     3  0.000000000 0.0000000 0.0000000 0.0000000 0.00000000 0.00000000 0.03294120 0.1465927
154     4  0.000000000 0.0000000 0.0000000 0.0000000 0.00000000 0.00000000 0.00000000 0.0000000
155     5  0.000000000 0.0000000 0.0000000 0.0000000 0.00000000 0.00000000 0.00000000 0.0000000
156     6  0.000000000 0.0000000 0.0000000 0.0000000 0.00000000 0.00000000 0.00000000 0.0000000
157     7  0.000000000 0.0000000 0.0000000 0.0000000 0.00000000 0.00000000 0.00000000 0.0000000
158     8  0.741550314 0.0000000 0.0000000 0.0000000 0.00000000 0.00000000 0.00000000 0.0000000
159     9  0.251022958 0.9261236 0.5927026 0.2592816 0.08888475 0.01851422 0.00000000 0.0000000
160     10 0.007426728 0.0738764 0.4072974 0.7407184 0.54894549 0.29707419 0.09667329 0.0000000
161       index
162 states          39          40          41          42          43          44          45          46
              47
163     1  0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.00000000 0.00000000 0.0000000
             0.00000000
164     2  0.5964486 0.1187578 0.0000000 0.0000000 0.0000000 0.00000000 0.00000000 0.0000000
             0.00000000
165     3  0.4035514 0.8812422 0.3378417 0.0000000 0.0000000 0.00000000 0.00000000 0.0000000
             0.00000000
166     4  0.0000000 0.0000000 0.6621583 0.5569255 0.0000000 0.00000000 0.00000000 0.0000000
             0.00000000
167     5  0.0000000 0.0000000 0.0000000 0.4430745 0.7760094 0.30962274 0.08077435 0.0000000
             0.00000000
168     6  0.0000000 0.0000000 0.0000000 0.0000000 0.2239906 0.59805752 0.52230553 0.2651274
             0.00000000
169     7  0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.09231974 0.37166247 0.5696354
             0.57586333
170     8  0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.00000000 0.02525765 0.1652372
             0.38319954
171     9  0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.00000000 0.00000000 0.0000000
             0.04093713
172     10 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.00000000 0.00000000 0.0000000
             0.00000000
173       index
174 states          48          49          50          51          52          53          54          55
175     1  0.0000000 0.00000000 0.00000000 0.04715039 0.18764856 0.48419879 0.6221480 0.4184326
176     2  0.0000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.1877031 0.5815674
177     3  0.0000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.0000000 0.0000000
178     4  0.0000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.0000000 0.0000000
179     5  0.0000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.0000000 0.0000000
180     6  0.0000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.0000000 0.0000000
181     7  0.2540821 0.09031835 0.02123451 0.00000000 0.00000000 0.00000000 0.0000000 0.0000000
182     8  0.5787774 0.45832030 0.26242651 0.10189735 0.01779764 0.00000000 0.0000000 0.0000000
183     9  0.1671405 0.45136135 0.53542156 0.45235491 0.24978248 0.07065798 0.0000000 0.0000000
184     10 0.0000000 0.00000000 0.18091742 0.39859734 0.54477132 0.44514323 0.1901490 0.0000000
185       index
```

```
states        56         57          58         59         60          61          62          63
            64
     1  0.0000000 0.0000000 0.00000000 0.0000000 0.0000000 0.00000000 0.00000000 0.00000000
          0.0000000
     2  0.7555088 0.2075164 0.03479317 0.0000000 0.0000000 0.00000000 0.00000000 0.00000000
          0.0000000
     3  0.2444912 0.7924836 0.42250832 0.1199028 0.0000000 0.00000000 0.00000000 0.00000000
          0.0000000
     4  0.0000000 0.0000000 0.54269851 0.5368674 0.2553288 0.00000000 0.00000000 0.00000000
          0.0000000
     5  0.0000000 0.0000000 0.00000000 0.3432299 0.5505936 0.44107119 0.11026780 0.01575254
          0.0000000
     6  0.0000000 0.0000000 0.00000000 0.0000000 0.1940776 0.46368702 0.50784753 0.23961455
          0.0556883
     7  0.0000000 0.0000000 0.00000000 0.0000000 0.0000000 0.09524179 0.34774139 0.51616355
          0.3717333
     8  0.0000000 0.0000000 0.00000000 0.0000000 0.0000000 0.00000000 0.03414328 0.22846936
          0.4643561
     9  0.0000000 0.0000000 0.00000000 0.0000000 0.0000000 0.00000000 0.00000000 0.00000000
          0.1082223
     10 0.0000000 0.0000000 0.00000000 0.0000000 0.0000000 0.00000000 0.00000000 0.00000000
          0.0000000
       index
states         65         66          67         68 69 70 71 72 73 74 75         76         77
            78
     1  0.00000000 0.0000000 0.07048501 0.3202126  1  0  0  0  0  0  0 0.0000000 0.0000000
          0.0000000
     2  0.00000000 0.0000000 0.00000000 0.0000000  0  1  0  0  0  0  0 0.0000000 0.0000000
          0.0000000
     3  0.00000000 0.0000000 0.00000000 0.0000000  0  0  1  0  0  0  0 0.0000000 0.0000000
          0.0000000
     4  0.00000000 0.0000000 0.00000000 0.0000000  0  0  0  1  0  0  0 0.0000000 0.0000000
          0.0000000
     5  0.00000000 0.0000000 0.00000000 0.0000000  0  0  0  0  1  0  0 0.0000000 0.0000000
          0.0000000
     6  0.00000000 0.0000000 0.00000000 0.0000000  0  0  0  0  0  1  0 0.0000000 0.0000000
          0.0000000
     7  0.13003496 0.0000000 0.00000000 0.0000000  0  0  0  0  0  0  1 0.2222222 0.0000000
          0.0000000
     8  0.48312907 0.2508172 0.00000000 0.0000000  0  0  0  0  0  0  0 0.7777778 0.4444444
          0.0000000
     9  0.35076186 0.5377277 0.43005981 0.0000000  0  0  0  0  0  0  0 0.0000000 0.5555556
          0.6666667
     10 0.03607411 0.2114550 0.49945518 0.6797874  0  0  0  0  0  0  0 0.0000000 0.0000000
          0.3333333
       index
states        79         80 81 82 83         84         85          86         87         88
     89
     1  0.1111111 0.5555556  1  1  1 0.4310467 0.1264025 0.00000000 0.0000000 0.0000000
          0.00000000
     2  0.0000000 0.0000000  0  0  0 0.5689533 0.6092885 0.37920750 0.1026843 0.0000000
          0.00000000
     3  0.0000000 0.0000000  0  0  0 0.0000000 0.2643090 0.53472518 0.5530464 0.3080528
          0.00000000
     4  0.0000000 0.0000000  0  0  0 0.0000000 0.0000000 0.08606732 0.3442693 0.5794631
          0.65033376
     5  0.0000000 0.0000000  0  0  0 0.0000000 0.0000000 0.00000000 0.0000000 0.1124840
          0.33205511
     6  0.0000000 0.0000000  0  0  0 0.0000000 0.0000000 0.00000000 0.0000000 0.0000000
          0.01761113
     7  0.0000000 0.0000000  0  0  0 0.0000000 0.0000000 0.00000000 0.0000000 0.0000000
          0.00000000
     8  0.0000000 0.0000000  0  0  0 0.0000000 0.0000000 0.00000000 0.0000000 0.0000000
          0.00000000
     9  0.0000000 0.0000000  0  0  0 0.0000000 0.0000000 0.00000000 0.0000000 0.0000000
          0.00000000
     10 0.8888889 0.4444444  0  0  0 0.0000000 0.0000000 0.00000000 0.0000000 0.0000000
          0.00000000
       index
states         90        91         92         93          94          95          96         97
            98
     1  0.00000000 0.000000 0.0000000 0.0000000 0.00000000 0.00000000 0.00000000 0.0000000
          0.0000000
     2  0.00000000 0.000000 0.0000000 0.0000000 0.00000000 0.00000000 0.00000000 0.0000000
          0.0000000
     3  0.00000000 0.000000 0.0000000 0.0000000 0.00000000 0.00000000 0.00000000 0.0000000
          0.0000000
     4  0.28334043 0.000000 0.0000000 0.0000000 0.00000000 0.00000000 0.00000000 0.0000000
          0.0000000
     5  0.63740946 0.775458 0.5465133 0.3175685 0.08862378 0.01477063 0.00000000 0.0000000
          0.0000000
     6  0.07925011 0.224542 0.4534867 0.6824315 0.91137622 0.36784548 0.08833972 0.0000000
          0.0000000
     7  0.00000000 0.000000 0.0000000 0.0000000 0.00000000 0.61738389 0.52947025 0.2207073
          0.0000000
     8  0.00000000 0.000000 0.0000000 0.0000000 0.00000000 0.00000000 0.38219003 0.5734981
          0.4118733
```

```
    9  0.00000000 0.000000 0.0000000 0.0000000 0.00000000 0.00000000 0.00000000 0.2057946
      0.4999290
    10 0.00000000 0.000000 0.0000000 0.0000000 0.00000000 0.00000000 0.00000000 0.0000000
      0.0881977
      index
states          99       100
    1  0.02939923 0.1837807
    2  0.00000000 0.0000000
    3  0.00000000 0.0000000
    4  0.00000000 0.0000000
    5  0.00000000 0.0000000
    6  0.00000000 0.0000000
    7  0.00000000 0.0000000
    8  0.00000000 0.0000000
    9  0.66183781 0.3309189
    10 0.30876296 0.4853004
> Viterbi
  [1] "2"  "2"  "2"  "2"  "2"  "3"  "3"  "3"  "4"  "5"  "6"  "6"  "6"  "7"  "8"  "9"  "9"  "9"
      "10"
 [20] "1"  "1"  "1"  "1"  "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "9"  "10" "1"  "1"  "1"
      "1"
 [39] "2"  "2"  "3"  "4"  "5"  "5"  "5"  "6"  "7"  "8"  "9"  "10" "1"  "1"  "1"  "1"  "1"  "2"
      "2"
 [58] "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "1"  "1"  "1"  "2"  "3"  "4"  "5"  "6"  "7"
      "8"
 [77] "9"  "10" "1"  "1"  "1"  "1"  "1"  "1"  "1"  "2"  "2"  "3"  "4"  "4"  "5"  "5"  "5"  "6"
      "7"
 [96] "8"  "9"  "10" "1"  "1"
```

Question 4 & 5

Compute the accuracy of the filtered and smoothed probability distributions, and of the most probable path. That is, compute the percentage of the true hidden states that are guessed by each method with different samples.

As the hint was telling us,the forward function in the HMM package returns probabilities in log scale so we needed to use the functions exp and prop.table in order to obtain a normalized probability distribution. Then, we also have used the functions apply and which.max to find out the most probable states. I have repeated the procedure 5 times for a simulation of length 100 leading to the following results:

```
> Simulations$PercentageSIm
[[1]]
      smoothingresult Filteringresult Viterbiresult
FALSE            0.32             0.54          0.39
TRUE             0.68             0.46          0.61

[[2]]
      smoothingresult Filteringresult Viterbiresult
FALSE            0.23             0.51          0.35
TRUE             0.77             0.49          0.65

[[3]]
      smoothingresult Filteringresult Viterbiresult
FALSE            0.42             0.51          0.44
TRUE             0.58             0.49          0.56

[[4]]
      smoothingresult Filteringresult Viterbiresult
FALSE             0.2              0.4          0.35
TRUE              0.8              0.6          0.65

[[5]]
      smoothingresult Filteringresult Viterbiresult
FALSE            0.36             0.46          0.61
TRUE             0.64             0.54          0.39
```

In general,the smoothed algorithm is more accurate than the filtered distributions because it is using future values to predict previous values as well, giving to the predicted one a better prediction from the future. Moreover, the smoothed distribution is also more accurate than the most probable path because it has more constraints that needs to be taken into account. Those constraints might be wrong at some point because you are never sure of where your real robot is (e.g. it can be

between $[i-2, i+2]$), so marking a probable path might make you choose not the most optimal place in general but according to the constraint.

Question 6

In order to answer whether it is true that the more observations you have the better you know where the robot is, it is necessary to understand what entropy in statistics is. Entropy is a measure of uncertainty which goes up when the uncertainty is high and goes down when the uncertainty goes down. By plotting the entropy for each point of the observations, if that statement was correct, we should see a decreasing trend of uncertainty in our model that should go asymptotically until 0. Nevertheless that is not the case, and it can be seen in the figure below. This makes sense since the model is randomly transitioned and no matter the more observations you have, you still have the same probability transition for it and the same information about the model for a particular point.

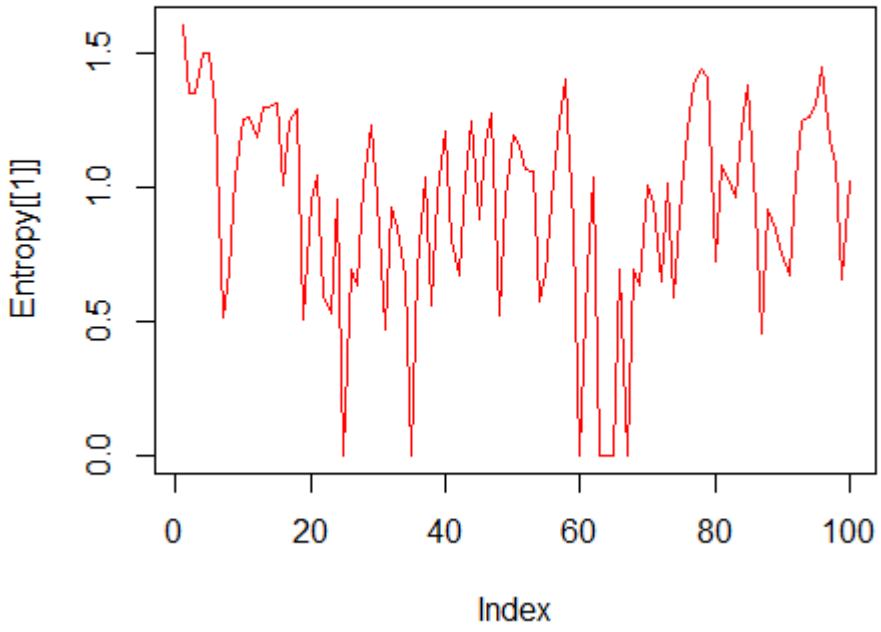

Figure 1: Graphical representation of the entropy for one of the simulations

Question 7

Consider any of the samples above of length 100. Compute the probabilities of the hidden states for the time step 101.

In order to do so, we just need as likelihood the posterior of our data and take the last distribution of the value and then multiplied by our prior which is our transition distribution, giving the following possible distribution:

```
> result
        [,1]
[1,] 0.1875
[2,] 0.5000
[3,] 0.3125
[4,] 0.0000
[5,] 0.0000
[6,] 0.0000
```

```
 [7,] 0.0000
 [8,] 0.0000
 [9,] 0.0000
[10,] 0.0000
```

Contributions

All results and comments presented have been developed and discussed together by the members of the group.

Appendix

Poisson regression-the MCMC way

```
1
2
3  ####################Lab1
4
5  set.seed(12345)
6  #install.packages("HMM")
7  library(HMM)
8
9  ##QUestion1
10 # Initialise HMM
11 states           <- as.character(1:10)
12 symbols          <- as.character(1:10)
13 startProbs       <- rep(0.1,10)
14 transProbs       <- diag(x = 0.5, 10)
15 transProbs[10,1]<- 0.5 #writing the ones left
16 emissionProbs    <- diag(x = 0.2, 10)
17 emissionProbs[1,9:10]<-rep(0.2,2)
18 emissionProbs[2,10]<-rep(0.2,1)
19 emissionProbs[9,1]<-rep(0.2,1)
20 emissionProbs[10,1:2]<-rep(0.2,1)
21 for(i in 1:ncol(transProbs)){
22   for(j in 1:nrow(transProbs)){
23     if(j == i+1){
24       transProbs[i,j]<-0.5
25       emissionProbs[i,j]<-0.2}
26     if(j == i+2){ emissionProbs[i,j]<-0.2}
27     if(j == i-1){ emissionProbs[i,j]<-0.2}
28     if(j == i-2){ emissionProbs[i,j]<-0.2}
29 }}
30
31
32 myhmm = initHMM(States = states,
33                 Symbols = symbols,
34                 startProbs = startProbs,
35                 transProbs= transProbs,
36                 emissionProbs=emissionProbs)
37
38 ##2
39 length<-100
40
41 my100sim<-simHMM(myhmm, length)
42
43
44 #3
45
46 myobs<-my100sim$observation # Taking just the real observations (Z)
47 ###Filtering
48 filtering<-exp(forward(myhmm, observation=myobs))#A matrix containing the forward probabilities
       given on a logarithmic scale (natural logarithm).
49 marginalFilter<-apply(as.data.frame(filtering),2,  FUN = function(x){prop.table(x)})##prop.
     table already calculated the % on 1.
50
51
52 ###Smoothing
53 smoothing<-posterior(myhmm, observation=myobs)
54
55 ##Most probable path
56
57 Viterbi<-viterbi(myhmm, observation=myobs)
58
59 #4
60 mystates<-my100sim$states
61
62
63 smoothingMostProb   <- sapply(as.data.frame(smoothing), which.max)
64 FilteringMostProb   <- sapply(as.data.frame(marginalFilter),which.max)
65 Viterbi
66
67 smoothingresult <-table(mystates == smoothingMostProb)
68 Filteringresult <-table(mystates == FilteringMostProb)
69 Viterbiresult   <-table(mystates == Viterbi)
70
71 ResultTable     <-cbind(smoothingresult, Filteringresult, Viterbiresult)
72
73 ###5
74
75
76 Comparingsimuations<-function(HMM, length){
77   ResultTable<-list()
78   FilterEntropy<-list()
79   library(entropy)
```

```
  for(i in 1:length(length)){
    my100sim              <-simHMM(myhmm, length[i])
    myobs                 <-my100sim$observation # Taking just the real observations (Z)
    ###Filtering
    filtering             <-exp(forward(myhmm, observation=myobs))#A matrix containing the
        forward probabilities given on a logarithmic scale (natural logarithm).
    marginalFilter        <-apply(as.data.frame(filtering),2,  FUN = function(x){prop.table(x)
        })##prop.table already calculated the % on 1.
    ###Smoothing
    smoothing             <-posterior(myhmm, observation=myobs)


    #Getting just the Z states for different models
    mystates              <-my100sim$states
    smoothingMostProb     <- sapply(as.data.frame(smoothing), which.max)
    FilteringMostProb     <- sapply(as.data.frame(marginalFilter),which.max)

    smoothingresult       <-table(mystates == smoothingMostProb)
    Filteringresult       <-table(mystates == FilteringMostProb)
    Viterbi               <-viterbi(myhmm, observation=myobs)##Most probable path

    Viterbiresult         <-table(mystates == Viterbi)
    ResultTable[[i]]      <-cbind(smoothingresult, Filteringresult, Viterbiresult)/length[i]


    FilterEntropy[[i]]    <- apply(marginalFilter,2,entropy.empirical)

  }
 return(list(PercentageSIm=ResultTable, Entropy=FilterEntropy))
}


length  <-rep(100,5)
myhmm = initHMM(States = states,
                Symbols = symbols,
                startProbs = startProbs,
                transProbs= transProbs,
                emissionProbs=emissionProbs)

Simulations<-Comparingsimuations(myhmm, length = length)

Simulations$PercentageSIm

#6
#install.packages("entropy")

Entropy<-Simulations$Entropy

plot(Entropy[[1]], col = "red", type = "l")
lines(Entropy[[2]], col = "blue")
lines(Entropy[[3]], col = "green")
lines(Entropy[[4]], col = "purple")
lines(Entropy[[5]], col = "black")

##7
likelihood<-posterior(myhmm, myobs)
prior<-transProbs
result<-prior%*%likelihood[,100]
```

732A96: Lab 3
Advanced Machine Learning

Carles Sans Fuentes

October 7, 2017

Assignment

Question 1

Implementing Gaussian process regression from scratch. This first exercise will have you writing your own code for the Gaussian process regression model:

$$y = f(x) + \epsilon, \epsilon \sim N(0, \sigma_n^2), with$$

$$f(x) \sim GP[0, k(x, x')]$$

. When it comes to the posterior distribution for f, I strongly suggest that you implement Algorithm 2.1 on page 19 of Rasmussen and Williams (RW) book. That algorithm uses the Cholesky decomposition (chol() in R) to attain numerical stability. Here is what you need to do:

a , b

Question a: Write your own code for simulating from the posterior distribution of f(x) using the squared exponential kernel. The function (name it posteriorGP) should return vectors with the posterior mean and variance of f, both evaluated at a set of x-values (x*). You can assume that the prior mean of f is zero for all x.

Question b: Now let the prior hyperparameters be $\sigma_f = 1$,$l = 0.3$. Update this prior with a single observation: $(x, y) = (0.4, 0.719)$. Assume that the noise standard deviation is known to be $\sigma_n = 0.1$. Plot the posterior mean of f over the interval$[-1,1]$. Plot also 95% probability (pointwise) bands for f.

Answer:

The function GaussianKernel() is the the squared exponential kernel. The SimGP() is the function related to the Gaussian Process. The prior has been updated as well as the other parameters. A plot with the mean and the confidence bands is shown below:

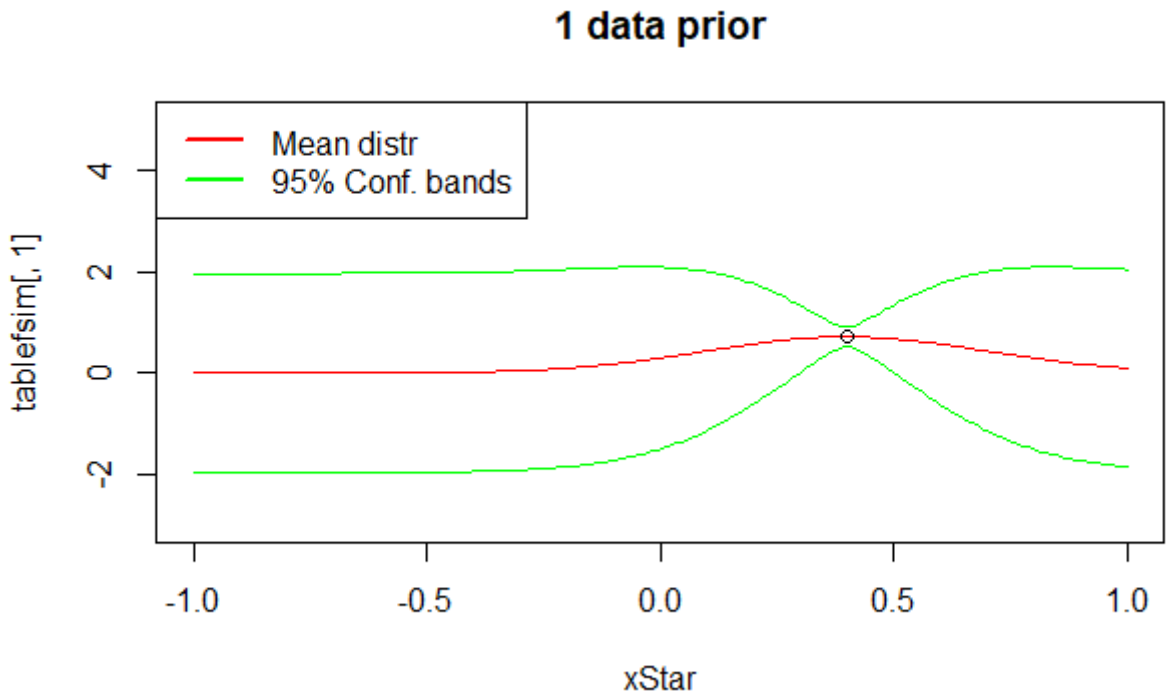

Figure 1: Posterior mean of f over the interval[−1,1]with 95% probability bands for f

c

Question: Update your posterior from 1b) with another observation: $(x, y) = (-0.6, -0.044)$. Plot the posterior mean of f over the interval $[-1, 1]$. Plot also 95% probability bands for f. [Hint: updating the posterior after one observation with a new observation gives the same result as updating the prior directly with the two observations. Bayes is beautiful!]

Answer: A plot with the mean and the confidence bands is shown below:

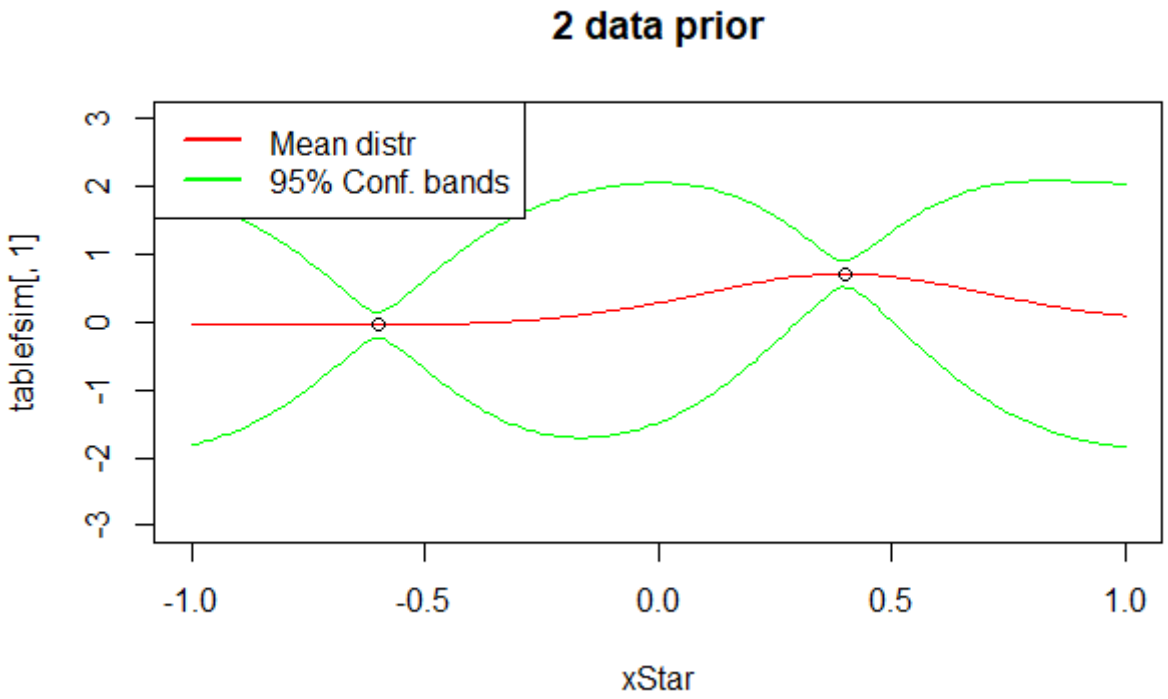

Figure 2: Posterior mean of f over the interval[−1,1]with 95% probability bands for f

d

Question: Compute the posterior distribution of f using all 5 data points in Table 1 below (note that the two previous observations are included in the table). Plot the posterior mean of f over the interval [−1,1]. Plot also 95% probability intervals for f.

Answer: A plot with the mean and the confidence bands is shown below:

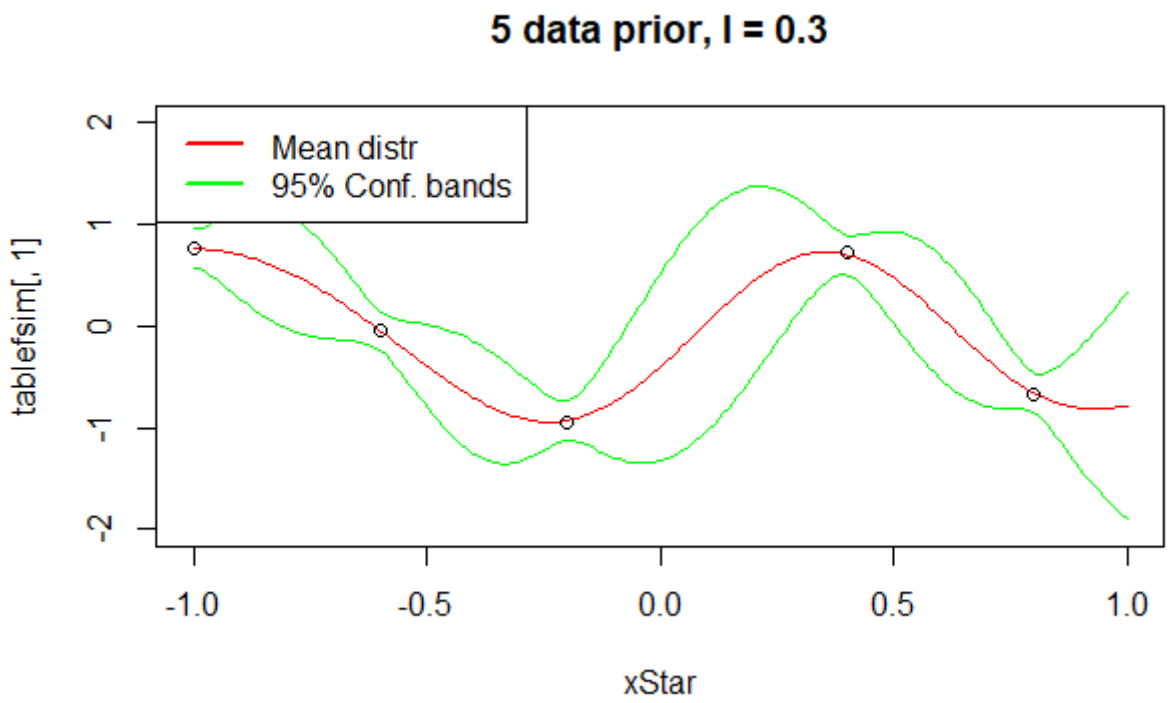

Figure 3: Posterior mean of f over the interval[−1,1]with 95% probability bands for f

e

Question: Repeat 1d), this time with the hyperparameters $\sigma_f = 1, l = 1$ Compare the results.

Answer: A plot with the mean and the confidence bands is shown below:

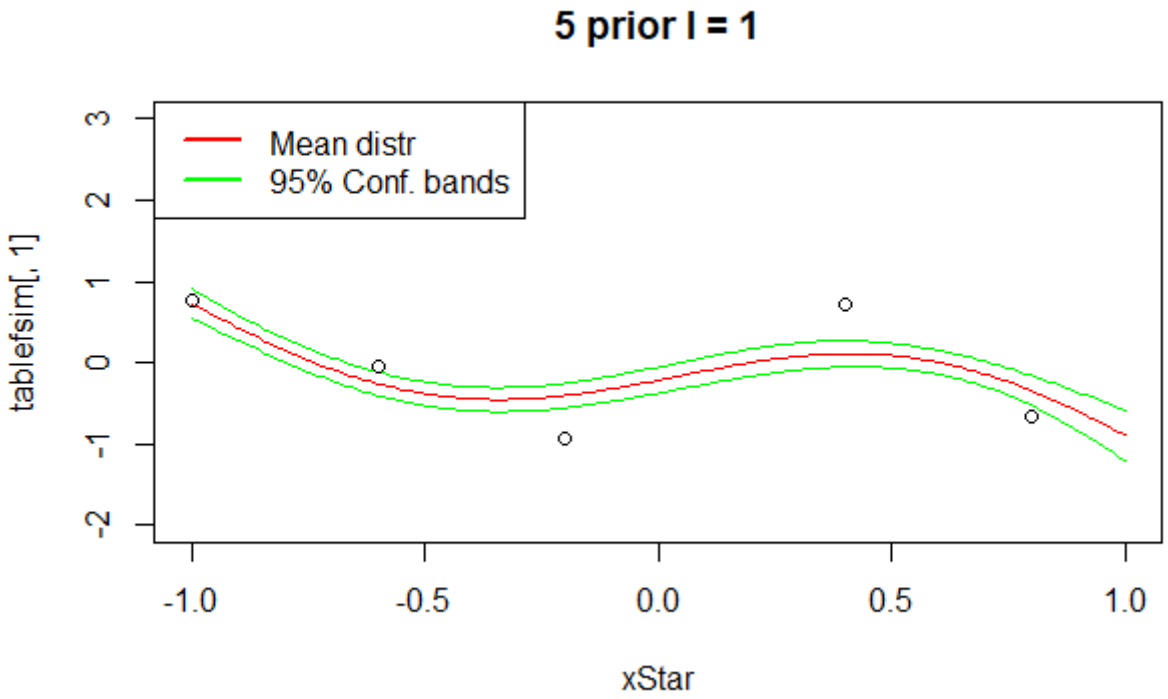

Figure 4: Posterior mean of f over the interval[−1,1]with 95% probability bands for f

It can be seen that the larger the l the more smoother our prediction will be, and the smaller our confidence bands get. This makes sense since the l parameter is a smoother hyper-parameter that the large it is the higher the covariance it is for a same σ_f, giving in this way more importance to σ_f. We are assuming by increasing l that we are more sure about our next distance training point. However, the choice of this l in our model is worse than having l equal to 3.

Question 2

Gaussian process regression on real data using the kernlab package. This exercise lets you explore the kernlab package on a data set of daily mean temperature in Stockholm (Tullinge) during the period January 1, 2010 - December 31, 2015. I have removed the leap year day February 29, 2012 to make your life simpler. A small version of it can be used in order to decrease computation time:

a

Question: Familiarize yourself with the following functions in kernlab, in particular the gausspr and kernelMatrix function. Do ?gausspr and read the input arguments and the output. Also,go through myKernLabDemo.R carefully; you will need to understand it. Now,define your own square exponential kernel function (with parameters l (ell) and σ_f (sigmaf)), evaluate it in the point $x = 1, x' = 2$, and use the kernelMatrix function to compute the covariance matrix $K(x, x_\star)$ for the input vectors $x = (1, 3, 4)^T and x_\star = (2, 3, 4)^T$

Answer: The results of the KernelMatrix and of my own square exponential kernel function called GaussianKernel2 (equal as in the previous activity but in a closure mode for convenience) has been displayed below.

```
> cov_matrix <- kernelMatrix(kernel =  GaussianKernel2, x=X, y = XStar)
> cov_matrix
An object of class "kernelMatrix"
          [,1]      [,2]       [,3]
[1,] 2.4261226 0.5413411 0.04443599
```

```
[2,] 2.4261226 4.0000000 2.42612264
[3,] 0.5413411 2.4261226 4.00000000

K<-GaussianKernel2(x=x1,y= x2)
> K
          [,1]
[1,] 2.426123
```

b

Question: Consider first the model:

$$temp = f(time) + \epsilon; \epsilon \sim N(0, \sigma_n^2)$$

$$f \sim GP(0, , k(time, time')$$

. Let σ_n^2 be the residual variance from a simple quadratic regression fit (using the lm() function in R). Estimate the above Gaussian process regression model using the squared exponential function from 2a) with $\sigma_f = 20$ and $l = 0.2$. Use the predict function to compute the posterior mean at every data point in the training datasets. Make a scatter plot of the data and superimpose the posterior mean of f as a curve (use $type =$ "l" in the plot function). Play around with different values on σ_f and l(no need to write this in the report though).

Answer: The plot is shown below:

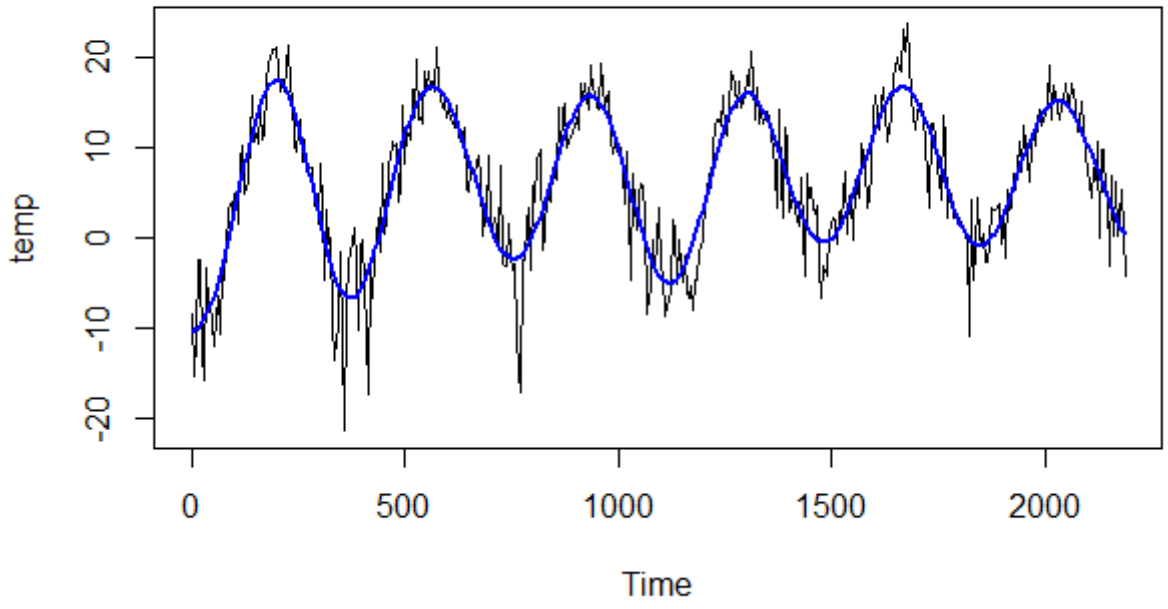

Figure 5: scatter plot of the data superimposed by the posterior mean of f as a curve

c

Question: Kernlab can compute the posterior variance of f, but I suspect a bug in the code (I get weird results). Do you own computations for the posterior variance of f (hint: Algorithm 2.1 in RW), and plot 95% (pointwise) posterior probability bands for f. Use $\sigma_f = 20$ and $l = 0.2$. Superimpose those bands on the figure with the posterior mean in 2b).

Answer: The plot is shown below:

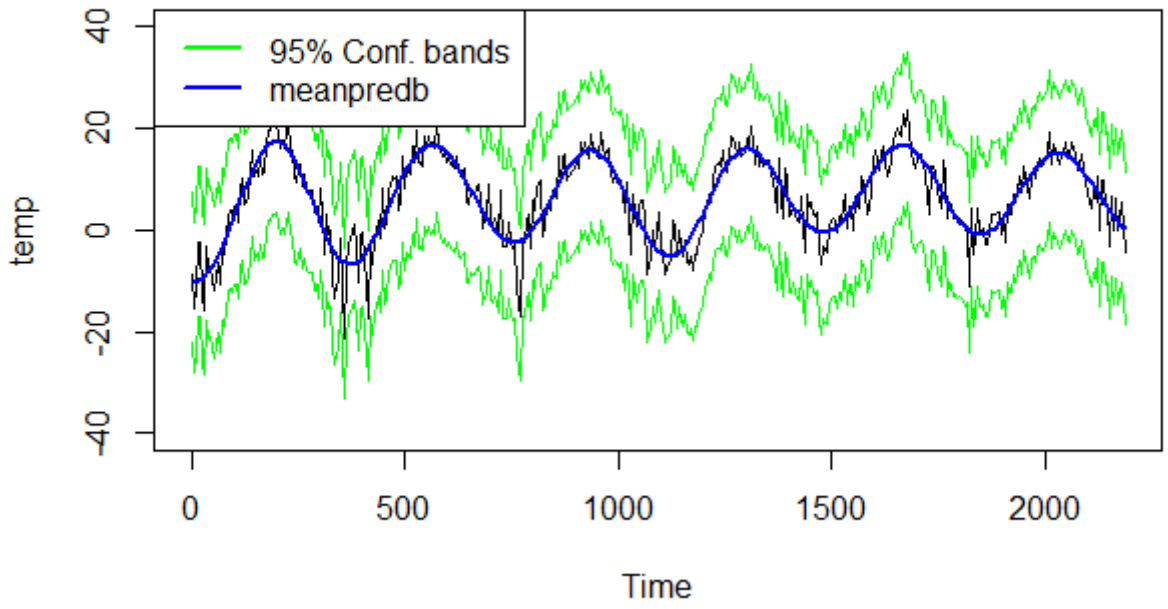

Figure 6: scatter plot of the data superimposed by the posterior mean of f as a curve with 95% probable confidence bands

d

Question:

$$temp = f(day) + \epsilon; \epsilon \sim N(0, \sigma_n^2)$$

$$f \sim GP(0, , k(day, day')$$

. Estimate the model using the squared exponential function from 2a) with $\sigma_f = 20$ and $l = 6 * 0.2 = 1.2$. (I multiplied ‘ by 6 compared to when you used time as input variable since kernlab automatically standardizes the data which makes the distance between points larger for day compared to time). Superimpose the posterior mean from this model on the fit (posterior mean) from the model with time using $\sigma_f = 20$,$l = 0.2$. Note that this plot should also have the time variable on the horizontal axis.

Answer: The plot is shown below:

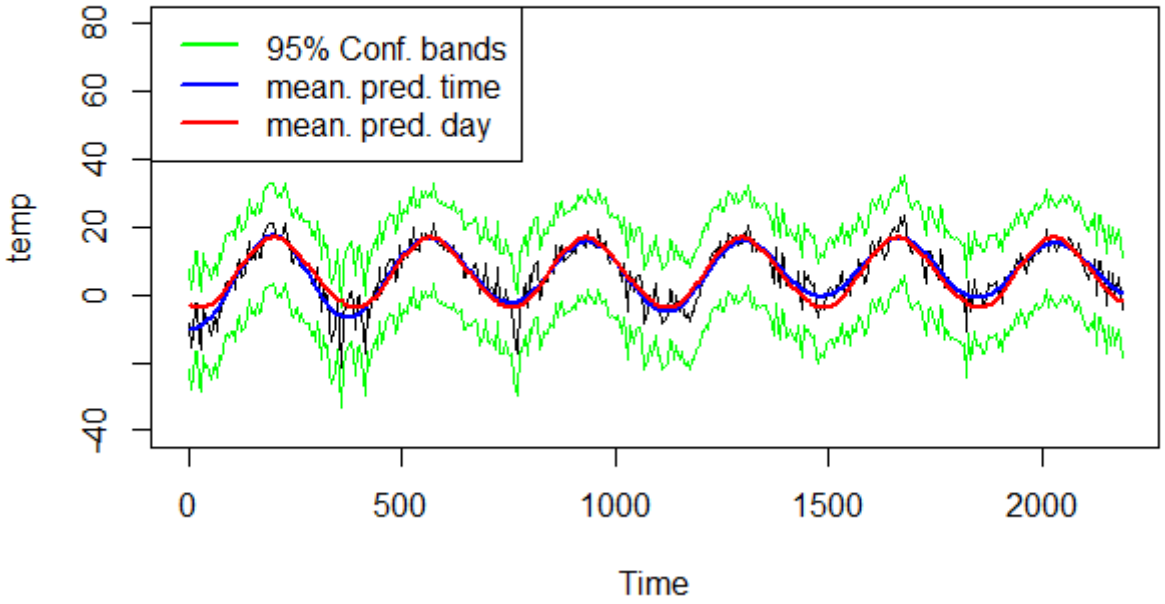

Figure 7: scatter plot of the data superimposed by the posterior mean of f as a curve with 95% probable confidence bands and new mean estimated by day model

The mean prediction for time (blue line) seems to take into account more larger variance in the data, whereas in the case of the mean of the prediction day (red line), it seems to be more stable even for larger residuals. These makes sense in the day model we are imposing to predict the same result year after year averaging all our data, whereas in the other observations are taken separately as they are linear and not cyclical over time. The pros of the and cons of the model are a possible prediction problem: overfitting vs having a stable good model.

e

Question: Now implement a generalization of the periodic kernel given in my slides from Lecture 2 of the GP topic (Slide 6) Compare the fit to the previous two models (with $\sigma_f = 20$,$l = 0.2$). Discuss the results.

Answer: The plot is shown below:

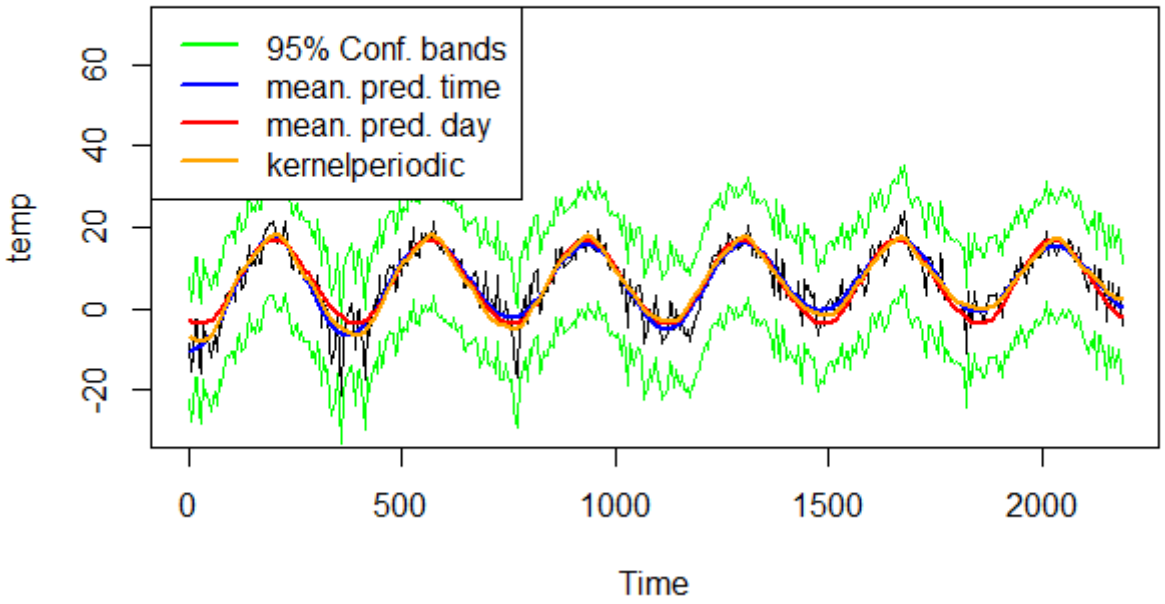

Figure 8: scatter plot of the data superimposed by the posterior mean of f as a curve with 95% probable confidence bands, new mean estimated by day model and new mean by the last periodic kernel

Having two l components enables to get a controlled stable mean model since it weights the seasonal and the every year different component. All in all, I think the (orange line) kernel periodic is better model since it combines both of the previous models in order to get a more stable prediction controlling just the parameters.

Question 3

Download the banknote fraud data:

a

Question: Use kernlab to fit a Gaussian process classification model for fraud on the training data, using kernlab. Use kernlab's the default kernel and hyperparameters. Start with using only the first two covariates varWave and skewWave in the model. (1) Plot contours of the prediction probabilities over a suitable grid of values for varWave and skewWave. Overlay the training data for $fraud = 1$ (as blue points) and $fraud = 0$ (as red points). You can take a lot of code for this from my KernLabDemo.R. (2)Compute the confusion matrix for the classifier and its accuracy.

Answer: The Confusion Matrix and the accuracy can be seen below

```
> predictionfunction(data = data, rows = SelectTraining)
     true
pred   0   1
   0 512  24
   1  44 420
[1] 0.932
```

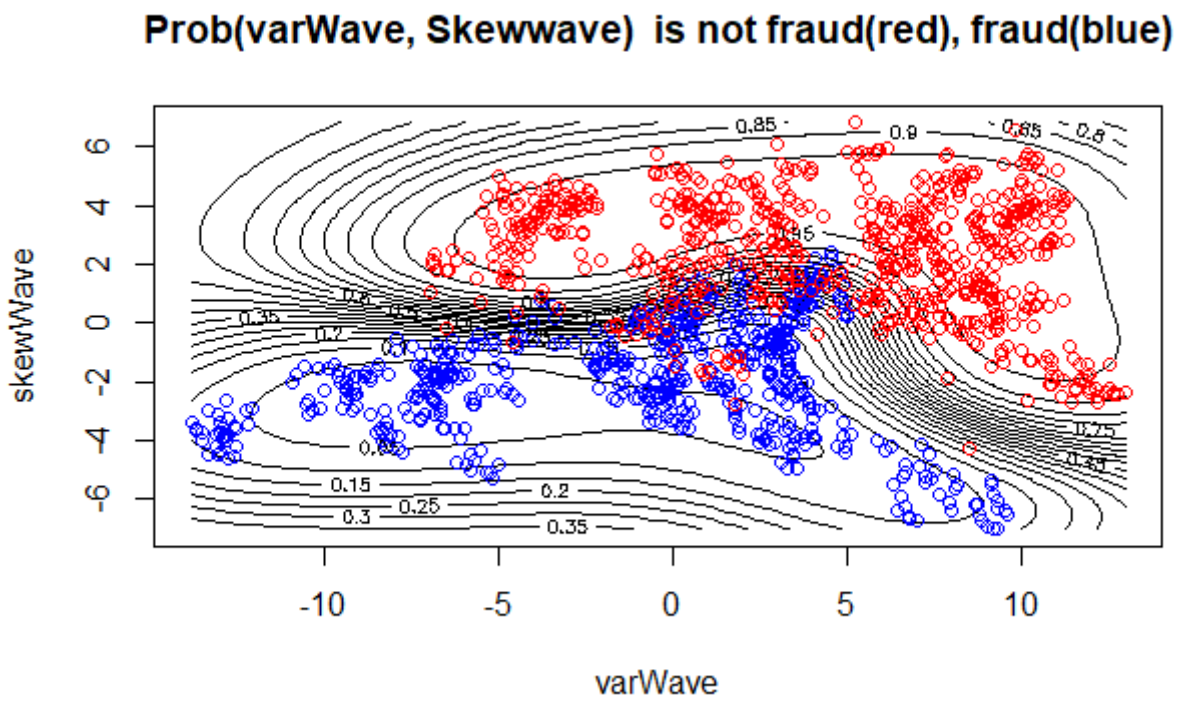

Figure 9: Plot contours of the prediction probabilities over a suitable grid of values for varWave and skewWave

b

Question: Using the estimated model from 3a), make predictions for the testset. Compute the accuracy

Answer: The Confusion Matrix and the accuracy can be seen below

```
> probPreds<-predictionfunction(data = data, rows = test)
     true
pred   0   1
   0 191   9
   1  15 157
[1] 0.9354839
```

c

Train a model using all four covariates. Make predictions on the test and compare the accuracy to the model with only two covariates.

Answer: The Confusion Matrix and the accuracy can be seen below

```
> predictionfunction(data = data, rows = SelectTraining)
     true
pred   0   1
   0 552   0
   1   4 444
[1] 0.996
> predictionfunction(data = data, rows = test)
     true
pred   0   1
   0 205   0
   1   1 166
[1] 0.9973118
```

With 4 covariates the prediction improves. This makes sense since we have more data to improve the prediction. Also these means that both covariates are important and significant to perform classification analysis.

Contributions

All results and comments presented have been developed and discussed together by the members of the group.

Appendix

Poisson regression-the MCMC way

```
1
2
3  ##Advanced ML lab 3
4  library("mvtnorm")
5
6  ##1a
7  x <-  c(0.4)
8  y <- c(0.719)
9  xStar <- seq(-1,1,length=200)
10 sigma_f<-1
11 l<-0.3
12 hyperParam<- c(sigma_f, l)
13 sigmaNoise<- 0.1
14 nSim <- 100
15
16 ###Calculating K from y and x, that are two vector inputs
17 GaussianKernel<- function(x1,x2, sigma_f =sigma_f, l=l){
18   n1<- length(x1)
19   n2<- length(x2)
20   K<- matrix(NA, n1, n2)
21   for(i in 1:n2){
22     K[,i]<-  sigma_f^2*exp(-0.5*( (x1-x2[i])/l)^2 )
23   }
24   return(K)
25 }
26
27 GaussianKernel(c(1,4), c(3,1), sigma = 2, l=1)
28 GaussianKernel(c(1,4), c(3,1), sigma = 2, l=2)
29 GaussianKernel(c(1,4), c(3,1), sigma = 2, l=3)
30 GaussianKernel(c(1,4), c(3,1), sigma = 2, l=4)
31
32
33
34 SimGP <- function(K,x,y, xStar, nSim, sigmaNoise, ...){
35   # Simulates nSim realizations (function) form a Gaussian process with mean m(x) and
       covariance K(x,x')
36   # over a grid of inputs (x)
37   n <- length(xStar)
38   #if (is.numeric(m)) meanVector <- rep(0,n) else meanVector <- m(xStar)
39
40
41   covMat    <- K(x, x,...)
42   covMatdiff2<- K(x, xStar,...)
43   covMatStar<- K(xStar, xStar,...)
44   L<-t(chol(covMat+ (sigmaNoise^2)*diag(dim(covMat)[2])))
45   alpha<- solve(t(L),solve(L,y))
46   fStar<- t(covMatdiff2)%*%alpha
47   v<-solve(L,covMatdiff2)
48   covfStar <- covMatStar-t(v)%*%v
49   return(list(mean=fStar, covfStar= diag(covfStar)))
50 }
51
52 ##b
53 fSim <- SimGP(y=y, xStar = xStar, x= x, K=GaussianKernel, nSim = nSim, sigma_f =sigma_f, l=l,
       sigmaNoise = sigmaNoise )
54
55 plot(fSim$mean)
56 plot(fSim$covfStar)
57 tablefsim<-cbind(mean = fSim$mean, low= fSim$mean-1.96*sqrt(fSim$covfStar),high= fSim$mean+1.96
       *sqrt(fSim$covfStar))
58 plot(x= xStar,y = tablefsim[,1], col = "red", type = "l", ylim = c(-3,5), main = "1 data prior"
       )
59 lines(x= xStar, y =tablefsim[,2], col ="green")
60 lines(x= xStar, y = tablefsim[,3], col = "green")
61 points(x = x, y = y)
62 legend("topleft", # places a legend at the appropriate place
63        c("Mean distr ", "95% Conf. bands"), # puts text in the legend
64        lty=c(1,1), # gives the legend appropriate symbols (lines)
65        lwd=c(2.5,2.5),col=c("Red","Green")) # gives the legend lines the correct color and
            width
66
67 ##c
68
69 x<-c(x,-0.6)
70 y<-c(y, -0.044)
71
72 fSim <- SimGP(y=y, xStar = xStar, x= x, K=GaussianKernel, nSim = nSim, sigma_f =sigma_f, l=l,
       sigmaNoise = sigmaNoise )
73
74 tablefsim<-cbind(mean = fSim$mean, low= fSim$mean-1.96*sqrt(fSim$covfStar),high= fSim$mean+1.96
       *sqrt(fSim$covfStar))
```

```
plot(x= xStar,y = tablefsim[,1], col = "red", type = "l", ylim = c(-3,3), main = "2 data prior")
lines(x= xStar, y =tablefsim[,2], col ="green")
lines(x= xStar, y = tablefsim[,3], col = "green")
points(x = x, y = y)
legend("topleft", # places a legend at the appropriate place
       c("Mean distr ", "95% Conf. bands"), # puts text in the legend
       lty=c(1,1), # gives the legend appropriate symbols (lines)
       lwd=c(2.5,2.5),col=c("Red","Green")) # gives the legend lines the correct color and width

###d
x<- c(-1.0,-0.6,-0.2,0.4,0.8)
y<- c(0.768,-0.044,-0.940,0.719, -0.664)

fSim <- SimGP(y=y, xStar = xStar, x= x, K=GaussianKernel, nSim = nSim, sigma_f =sigma_f, l=l, sigmaNoise = sigmaNoise )

tablefsim<-cbind(mean = fSim$mean, low= fSim$mean-1.96*sqrt(fSim$covfStar),high= fSim$mean+1.96*sqrt(fSim$covfStar))
plot(x= xStar,y = tablefsim[,1], col = "red", type = "l", main = "5 data prior, l = 0.3", ylim = c(-2,2))
lines(x= xStar, y =tablefsim[,2], col ="green")
points(x = x, y = y)
lines(x= xStar, y = tablefsim[,3], col = "green")
legend("topleft", # places a legend at the appropriate place
       c("Mean distr ", "95% Conf. bands"), # puts text in the legend
       lty=c(1,1), # gives the legend appropriate symbols (lines)
       lwd=c(2.5,2.5),col=c("Red","Green")) # gives the legend lines the correct color and width

###e

sigma_f<-1
l<-1
hyperParam<- c(sigma_f, l)

fSim <- SimGP(y=y, xStar = xStar, x= x, K=GaussianKernel, nSim = nSim, sigma_f =sigma_f, l=l, sigmaNoise = sigmaNoise )

tablefsim<-cbind(mean = fSim$mean, low= fSim$mean-1.96*sqrt(fSim$covfStar),high= fSim$mean+1.96*sqrt(fSim$covfStar))
plot(x= xStar,y = tablefsim[,1], col = "red", type = "l", ylim = c(-2,3), main = "5 prior l = 1")
lines(x= xStar, y =tablefsim[,2], col ="green")
lines(x= xStar, y = tablefsim[,3], col = "green")
points(x = x, y = y)
legend("topleft", # places a legend at the appropriate place
       c("Mean distr ", "95% Conf. bands"), # puts text in the legend
       lty=c(1,1), # gives the legend appropriate symbols (lines)
       lwd=c(2.5,2.5),col=c("Red","Green")) # gives the legend lines the correct color and width

###################################2
#
#
#
# plot(xStar, fSim[1,], type="l", ylim = c(-3,3))
# for (i in 2:dim(fSim)[1]) {
#   lines(xStar, fSim[i,], type="l")
# }
# lines(xStar,MeanFunc(xStar), col = "red", lwd = 3)
#
#
# # Plotting using manipulate package
# #install.packages("manipulate")
# library(manipulate)
#
# plotGPPrior <- function(sigma_f, l, nSim=20){
#   fSim <- SimGP(y=y, xStar = xStar, x= x, K=GaussianKernel, nSim = nSim, sigma_f =sigma_f, l=l, sigmaNoise = sigmaNoise )
#   plot(xStar, fSim[1,], type="l", ylim = c(-3,3), ylab="f(x)", xlab="x")
#   for (i in 2:nSim) {
#     lines(xStar, fSim[i,], type="l")
#   }
#   title(paste('length scale =',l,', sigmaf =',sigma_f))
# }
#
# manipulate(
#   plotGPPrior(sigma_f, l, nSim = 20),
#   sigma_f = slider(0, 2, step=0.1, initial = 1, label = "SigmaF"),
#   l = slider(0, 2, step=0.1, initial = 1, label = "Length scale, l")
# )
#

###2

```

```r
library(kernlab)
Data<- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/
    TempTullinge.csv", header=TRUE,
                sep = ";")

TData<-data.frame(time = 1:2190, date = Data$date,day =rep(1:365,6), temp = Data$temp)
GaussKernel<- function(sigmaF , ell){
  rval <- function(x,y, sigma_f = sigmaF, l = ell){
    n1 <- length(x)
    n2 <- length(y)
    K <- matrix(NA,n1,n2)
    for (i in 1:n2){
      K[,i] <- sigma_f^2*exp(-0.5*( (x-y[i])/l)^2 )
    }
    return(K)
  }

  class(rval) <- "kernel"
  return(rval)
}

SmallData<-  TData[seq(1, nrow(TData), 5), ]
sigma_f<-2
l<-1
X<-matrix(c(1,3,4), ncol = 1)
XStar<-matrix(c(2,3,4), ncol = 1)

GaussianKernel2 <- GaussKernel(sigmaF =  sigma_f, ell = l)
cov_matrix <- kernelMatrix(kernel =  GaussianKernel2, x=X, y = XStar)

x1<-1
x2<-2

K<-GaussianKernel2(x=x1,y= x2)
##b
sigma_f<-20
l <-0.2

x<-SmallData$time
x_prime<- seq(1,365,1)
GaussianKernel3<-GaussKernel(sigmaF = sigma_f, ell = l)
K<-kernelMatrix(GaussianKernel3, x,x_prime)

mylm<-lm(temp~time+ I(time^2), SmallData)
sigma_2_n<-var(mylm$residuals)


GPfit <- gausspr(temp ~ time, data = SmallData ,
                 kernel = GaussianKernel3,
                 kpar = list(sigma = sigma_f, ell = l), var = sigma_2_n)

myrange<-range(SmallData$time)
Xgrid<- myrange[1]:myrange[2]
meanPred<-predict(GPfit, SmallData)
length(meanPred)
plot(x = SmallData$time, SmallData$temp, type = "l",
     xlab = "Time", ylab = "temp")
lines(SmallData$time, meanPred, col="blue", lwd = 2)


##c

fSim <- SimGP(y=SmallData$temp, xStar = SmallData$time, x= SmallData$time,
              K=GaussianKernel3, nSim = nSim,
              sigma_f =sigma_f, l=l,
              sigmaNoise = sqrt(sigma_2_n ))


tablefsim<-cbind(mean = fSim$mean, low= fSim$mean-1.96*sqrt(fSim$covfStar),high= fSim$mean+1.96
    *sqrt(fSim$covfStar))
plot(x = SmallData$time, SmallData$temp, type = "l",
     xlab = "Time", ylab = "temp", ylim= c(-40,40))
lines(SmallData$time, meanPred, col="blue", lwd = 2)
lines(x= SmallData$time, y =tablefsim[,2], col ="green")
lines(x= SmallData$time, y = tablefsim[,3], col = "green")
legend("topleft", # places a legend at the appropriate place
       c( "95% Conf. bands", "meanpredb"), # puts text in the legend
       lty=c(1,1), # gives the legend appropriate symbols (lines)
       lwd=c(2.5,2.5),col=c("Green", "blue")) # gives the legend lines the correct color and
            width

####d


mylm<-lm(temp~day+ I(day^2), SmallData)
sigma_2_n<-var(mylm$residuals)
sigma_f<-20
```

```
l <- 1.2


GaussianKernel4<-GaussKernel(sigmaF = sigma_f, ell = l)


GPfit <- gausspr(temp ~ day, data = SmallData ,
                 kernel = GaussianKernel4,
                 kpar = list(sigma = sigma_f, ell = l), var = sigma_2_n)


meanPred2<-predict(GPfit, SmallData)

plot(x = SmallData$time, SmallData$temp, type = "l",
     xlab = "Time", ylab = "temp", ylim= c(-40,80))
lines(SmallData$time, meanPred, col="blue", lwd = 2)
lines(x= SmallData$time, y =tablefsim[,2], col ="green")
lines(x= SmallData$time, y = tablefsim[,3], col = "green")
lines(SmallData$time, meanPred2, col="red", lwd = 2)

legend("topleft", # places a legend at the appropriate place
       c( "95% Conf. bands", "mean. pred. time", "mean. pred. day"), # puts text in the legend
       lty=c(1,1), # gives the legend appropriate symbols (lines)
       lwd=c(2.5,2.5),col=c("Green", "blue", "red")) # gives the legend lines the correct color
            and width



##e

periodicKernel<- function(sigmaF, ell1, ell2, di){
  rval<-function(x1,x2, sigma_f =sigmaF, l_1=ell1, l_2=ell2, d=di){

      K<-  sigma_f^2*exp((-2*sin(pi*abs(x1-x2)/d)**2)/l_1^2)*exp((-1/2*abs(x1-x2)**2)/l_2^2)

  }
  class(rval) <- "kernel"
  return(rval)
}
sigma_f<-20
l_1<- 1
l_2<-10
d <- 365/sd(SmallData$time)
periodicKernel5<-periodicKernel(sigmaF=sigma_f, ell1=l_1, ell2=l_2, di=d)

GPfit <- gausspr(temp ~ time, data = SmallData ,
                 kernel = periodicKernel5,
                 kpar = list(sigma = sigma_f, ell = l), var = sigma_2_n)

meanPred3<-predict(GPfit, SmallData)

plot(x = SmallData$time, SmallData$temp, type = "l",
     xlab = "Time", ylab = "temp", ylim= c(-30,70))
lines(SmallData$time, meanPred, col="blue", lwd = 2)
lines(x= SmallData$time, y =tablefsim[,2], col ="green")
lines(x= SmallData$time, y = tablefsim[,3], col = "green")
lines(SmallData$time, meanPred2, col="red", lwd = 2)
lines(SmallData$time, meanPred3, col="orange", lwd = 2)
legend("topleft", # places a legend at the appropriate place
       c( "95% Conf. bands", "mean. pred. time", "mean. pred. day", "kernelperiodic"), # puts
           text in the legend
       lty=c(1,1), # gives the legend appropriate symbols (lines)
       lwd=c(2.5,2.5),col=c("Green", "blue", "red", "orange")) # gives the legend lines the
           correct color and width



##3


data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/
    banknoteFraud.csv", header=FALSE, sep=",")
names(data) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")


data[,5] <- as.factor(data[,5])
set.seed(111)
SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
model<-gausspr(fraud~varWave+skewWave, data = data[SelectTraining,])
##Part III
data<- data
predictionfunction<- function(data, rows){
  prediction<-predict(model,data[rows,])
  confusionMat<-table(pred=prediction, true = data[rows,5]) # confusion matrix
  Accuracy<-sum(diag(confusionMat))/sum(confusionMat)
  print(confusionMat)
  print(as.numeric(Accuracy))
```

```
318
319 }
320
321 predictionfunction(data = data, rows = SelectTraining)
322
323 ##contour
324 x1 <- seq(min(data[,"varWave"]),max(data["varWave"]),length=100)
325 x2 <- seq(min(data["skewWave"]),max(data["skewWave"]),length=100)
326 gridPoints <- meshgrid(x1, x2)
327 gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))
328
329 gridPoints <- data.frame(gridPoints)
330 names(gridPoints) <- c("varWave", "skewWave")
331 ######
332 probPreds <- predict(model, gridPoints, type="probabilities")
333 # Plotting for Prob(setosa)
334 contour(x2,x1,matrix(probPreds[,1],100), 20, xlab = "skewWave", ylab = "varWave", main = "Prob(
      varWave, Skewwave)  is not fraud(red), fraud(blue)")
335 points(data[data[,5]==1,"skewWave"],data[data[,5]==1,"varWave"],col="blue")
336 points(data[data[,5]==0,"skewWave"],data[data[,5]==0,"varWave"],col="red")
337
338 ###b
339
340 test<-setdiff( 1:dim(data)[1], SelectTraining)
341 probPreds<-predictionfunction(data = data, rows = test)
342
343
344
345
346
347 #c
348
349 model<-gausspr(fraud~., data = data[SelectTraining,])
350 #train
351 predictionfunction(data = data, rows = SelectTraining)
352 #test
353 predictionfunction(data = data, rows = test)
```

732A96: Lab 4
Advanced Machine Learning

Carles Sans Fuentes

October 9, 2017

Assignment

Question 1

Question The purpose of the lab is to put in practice some of the concepts covered in the lectures. To do so, you are asked to implement the particle filter for robot localization. For the particle filter algorithm, please check Section 13.3.4 of Bishop's book and/or the slides for the last lecture on SSMs.

Run it for $T = 100$ time steps to obtain $z_{1:100}$ (i.e. states) and $x_{1:100}$ (i.e. observations). Use the observations (i.e. sensor readings) to identify the state (i.e. robot location) via particle filtering. Use 100 particles. For each time step, show the particles, the expected location and the true location. Repeat the exercise after replacing the standard deviation of the emission model with 5 and then with 50. Comment on how this affects the results. Finally, show and explain what happens when the weights in the particle filter are always equal to 1, i.e. there is no correction.

Answer:

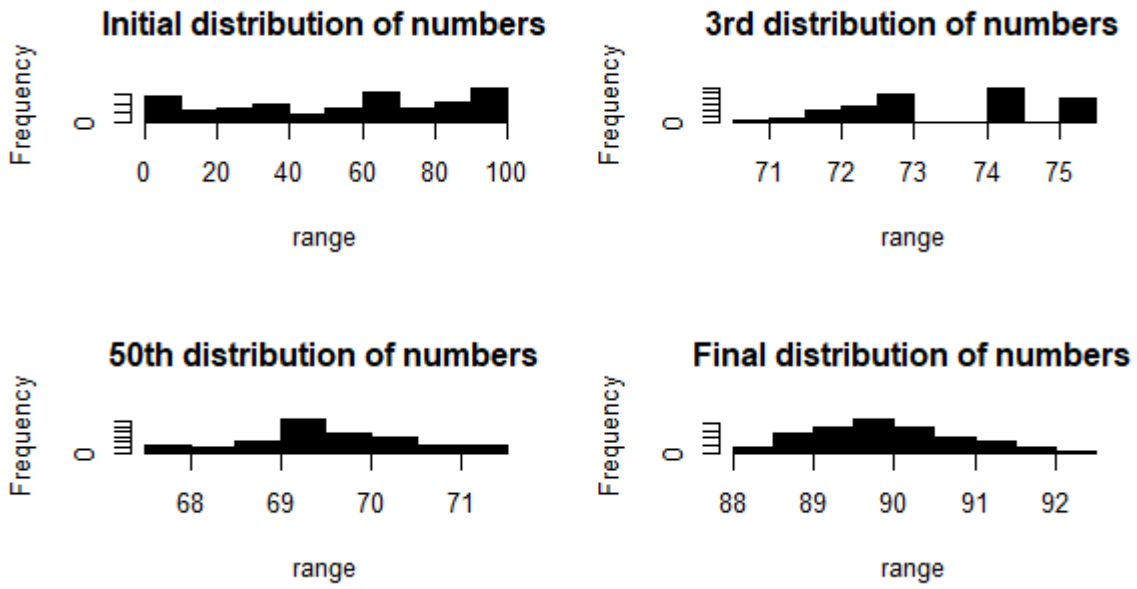

Figure 1: Histogram of the distribution of parameters with sd equal to 1

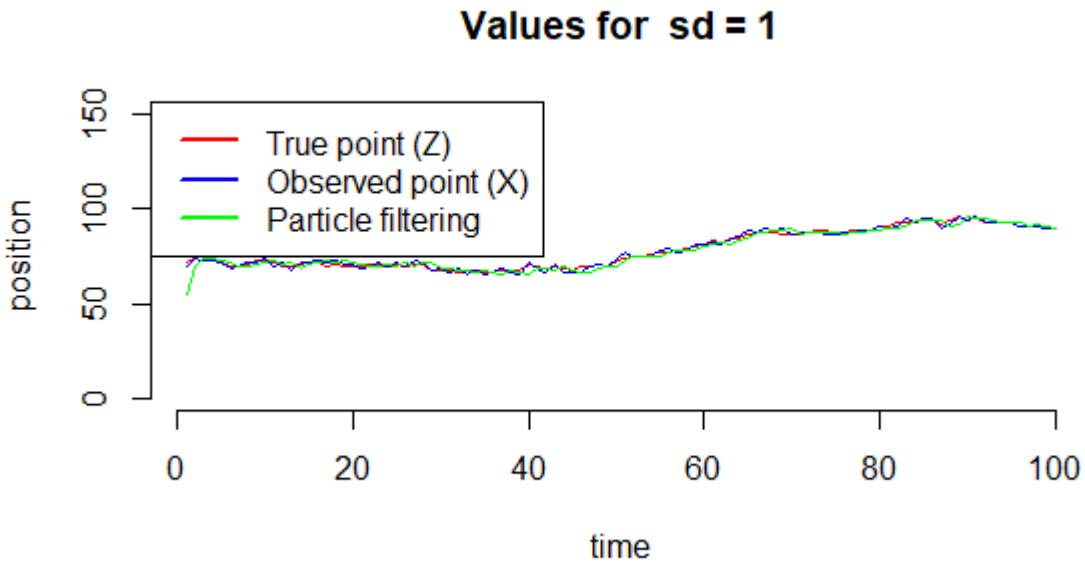

Figure 2: Plot X, Z and the particle filtering for 100 simulations with sd equal to 1

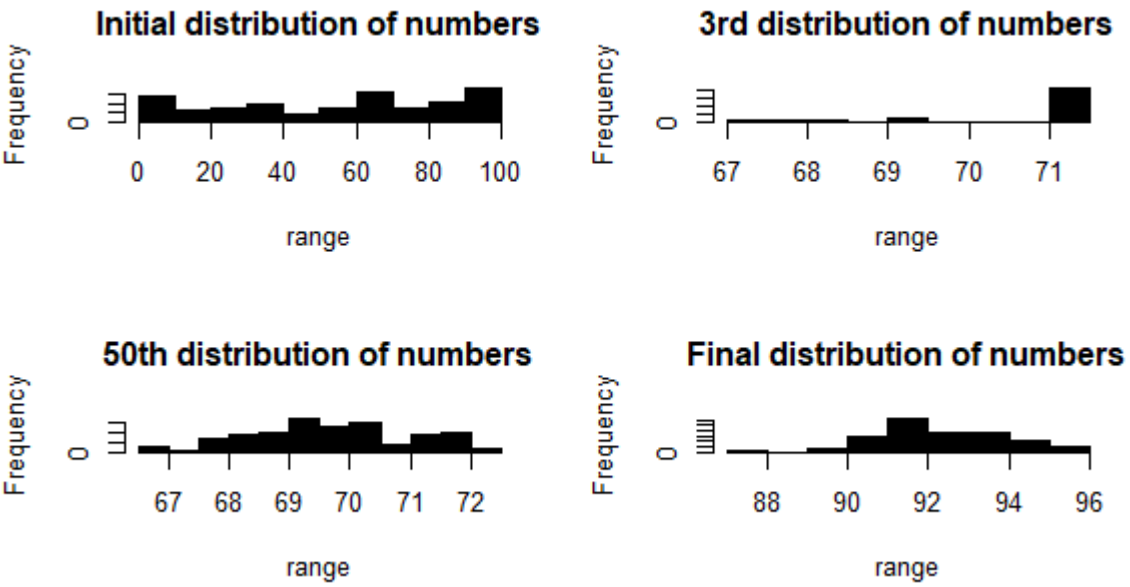

Figure 3: Histogram of the distribution of parameters with sd equal to 5

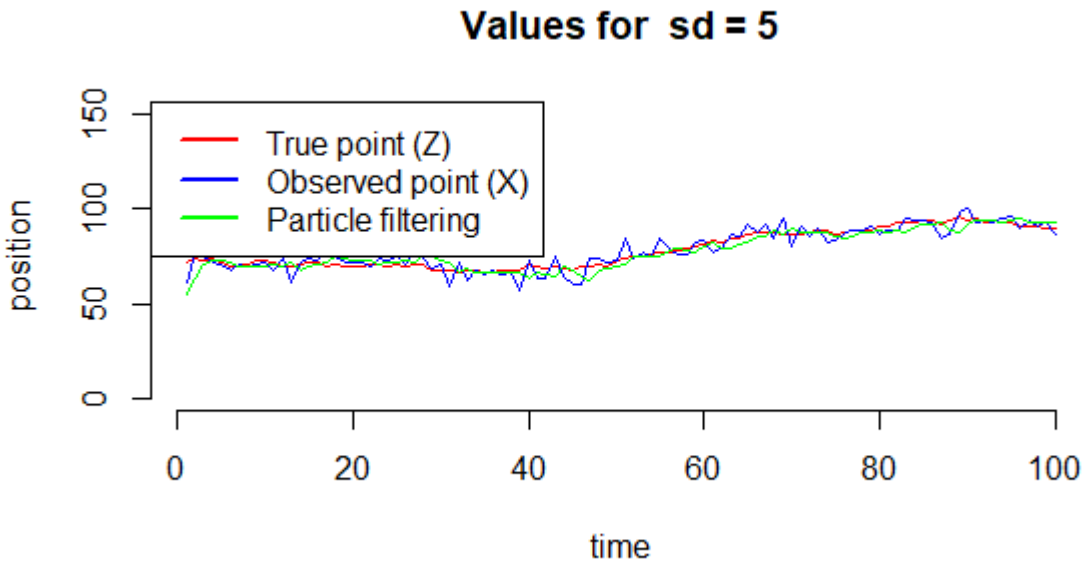

Figure 4: Plot X, Z and the particle filtering for 100 simulations with sd equal to 5

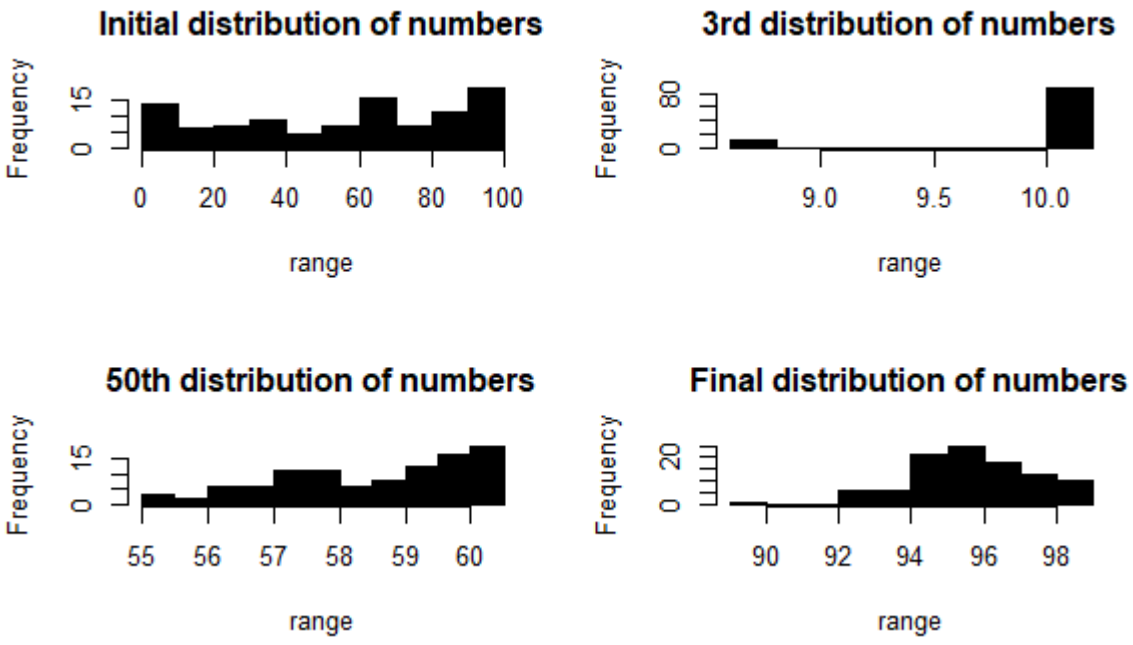

Figure 5: Histogram of the distribution of parameters with sd equal to 50

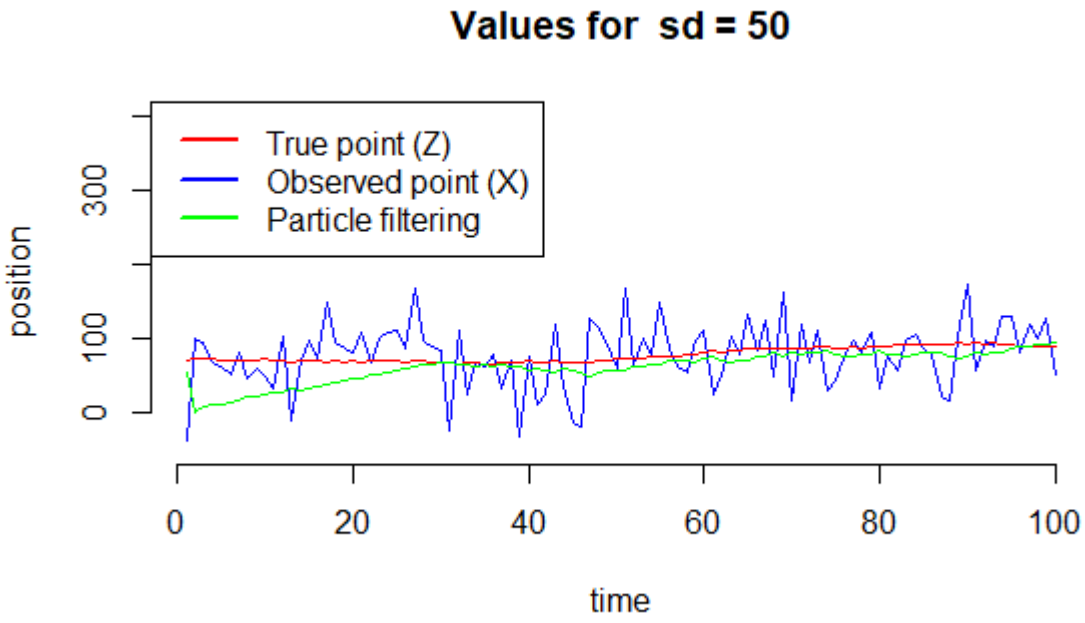

Figure 6: Plot X, Z and the particle filtering for 100 simulations with sd equal to 50

When increasing the standard deviation of the emission function, we are both less certain about the location of the robot givebn that our sensor is worse. Intuitively, if the sensor is worse (e.g. higher variance), our prediction will also be worse. For that , we can see that the higher the sd, the worse and more bumped our X is (blue line) and the more the particle filtering lasts to converge to the real position. Nevertheless, it ends up giving a good prediction.

If we set every time the weight to be 1 in all cases (e.g. it means all numbers have the same weights), our prediction will be much worse since we are not taking into account the conditional probabilities of the previous points.

Contributions

All results and comments presented have been developed and discussed together by the members of the group.

Appendix

Poisson regression-the MCMC way

```
1
2  ##########
3  #####1
4  set.seed(12345)
5  #######Generating model
6  #initial model
7  p1<-runif(1,0,100)
8
9  # Transition
10
11 sd_T<-1
12 sd_E<-50
13
14 Transition<- function(n,p0, sd_2){
15   z<- integer(n)
16   z[1]<-p0
17   for(i in 2:n){
18     p<- sample(c(z[i-1]-1,z[i-1],z[i-1]+1),1)
19     z[i]<-rnorm(1,p,sqrt(sd_2))
20     }
21   return(z)
22 }
23 myZ<-Transition(100, p0= p1, sd_2= sd_T**2)
24 ##Emission model
25
26
27 Emission<- function(vectorz, sd_2){
28   n<- length(vectorz)
29   x<- integer(n)
30   for(i in 1:n){
31     p<- sample(c(vectorz[i]-1,vectorz[i],vectorz[i]+1),1)
32     x[i]<-rnorm(1,p,sqrt(sd_2))
33   }
34   return(x)
35 }
36 x_t<- Emission(myZ, sd_2 = sd_E**2)
37
38 ######################
39 weights<- rep(0.01,100)
40 X<-runif(100,0,100)
41 calculationprob<- function(simulations, init_weights, grid, sd_E, sd_T, x_t= x_t){
42   n<- length(grid)
43   parameters<- matrix( ncol = n,nrow = simulations)
44   newweigths<- matrix(ncol = n,nrow = simulations)
45
46   newweigths[1,]<-init_weights/sum(init_weights)
47   parameters[1,]<-sample(x =grid,size = n, replace=TRUE, prob = newweigths[1,])
48  ### sample(dnorm(parameters[1,],grid) Left that
49   xt<- integer(100)
50   for(i in 2:simulations){
51
52     for(j in 1:simulations){
53       xt[j]<-Transition(2,parameters[i-1,j],sd_T)[2]
54     }
55     newweigths[i,]<-dnorm(x_t[i-1], xt, sqrt(sd_E))/sum(dnorm(x_t[i-1], xt,sqrt(sd_E)))
56     parameters[i,]<-sample(xt, size =n, replace=TRUE, prob = newweigths[i,])
57
58   }
59   result<- list(parameters = parameters, weights=newweigths)
60   return(result)
61 }
62 sd_2T<-1
63 sd_2E<-1
64
65 trial<-calculationprob(simulations =100, init_weights = weights, grid= X,
66                        sd_T = sd_T, sd_E = sd_E,x_t= x_t)
67 parameter_est<-trial$parameters
68 parameter_est
69
70 plotting<- function(Z=myZ,  X=x_t , particle= trial$parameters){
71   n<-1:length(Z)
72   for(i in n){
73     plot(0, xlim= c(1,100), ylim = c(0,150 ),bty='n',pch='',ylab='sep representation for
           clearness',xlab='position')
74     points(y = 20, x = Z[i], col ="red")
75     points(y = 30, x = X[i], col ="blue")
76     points(y =rep(40, 100), x = particle[i,], col = "green")
77     legend("topleft", # places a legend at the appropriate place
78            c("True point (Z) ", "Estimated point (X)", "Particle filtering"), # puts text in
                 the legend
79            lty=c(1,1), # gives the legend appropriate symbols (lines)
```

```
            lwd=c(2.5,2.5),col=c("Red"," blue", "Green")) # gives the legend lines the correct
                color and width
             Sys.sleep(0.2)
  }
}
par(mfrow=c(1,1))
plotting()


par(mfrow=c(1,1))
plot(0, xlim= c(1,100), ylim = c(-50,400),
     bty='n',pch='',ylab='position',xlab='time', main = paste0("Values for  sd = ", sd_E))
lines(x = 1:100, y = myZ, col ="red")
lines(x = 1:100, y = x_t, col ="blue")
lines(x =1:100, y = apply(trial$parameters,1,mean), col = "green")
legend("topleft", # places a legend at the appropriate place
       c("True point (Z) ", "Observed point (X)", "Particle filtering"), # puts text in the
           legend
       lty=c(1,1), # gives the legend appropriate symbols (lines)
       lwd=c(2.5,2.5),col=c("Red"," blue", "Green")) # gives the legend lines the correct color
            and width

par(mfrow=c(2,2))
hist(trial$parameters[1,], main= "Initial distribution of numbers", xlab = "range", col = "
    black")
hist(trial$parameters[3,], main= "3rd distribution of numbers", xlab = "range", col = "black")
hist(trial$parameters[50,], main= "50th distribution of numbers", xlab = "range", col = "black"
    )
hist(trial$parameters[100,], main= "Final distribution of numbers", xlab = "range", col = "
    black")
```