Viktor Gustafsson vikgu708

Lab 2 TDDE15

Assignment 1 – Implement Robot Model

I implemented the network using the provided information given. The transition probabilities were given as 50% chance of moving to the next state in a circle and 50% of staying. The emission probabilities were given as 1/5 to display the current state or +/- 2 states. This was translated to transition and emission matrixes were the rows corresponds to the current state and the column refers to the next state/emission for the state and emission matrix respectively.

I choose to name the circular states as integers from 1 to 10 for ease of programming. The emission state labels were chosen as strings, “C1” to “C10”.

Code

```
library("HMM")

transProbs=t(matrix(c(0.5,0.5,0,0,0,0,0,0,0,0,
                      0,0.5,0.5,0,0,0,0,0,0,0,
                      0,0,0.5,0.5,0,0,0,0,0,0,
                      0,0,0,0.5,0.5,0,0,0,0,0,
                      0,0,0,0,0.5,0.5,0,0,0,0,
                      0,0,0,0,0,0.5,0.5,0,0,0,
                      0,0,0,0,0,0,0.5,0.5,0,0,
                      0,0,0,0,0,0,0,0.5,0.5,0,
                      0,0,0,0,0,0,0,0,0.5,0.5,
                      0.5,0,0,0,0,0,0,0,0,0.5),nrow=10,ncol=10))

emissionProbs = t(matrix(c(0.2,0.2,0.2,0,0,0,0,0,0.2,0.2,
                           0.2,0.2,0.2,0.2,0,0,0,0,0,0.2,
                           0.2,0.2,0.2,0.2,0.2,0,0,0,0,0,
                           0,0.2,0.2,0.2,0.2,0.2,0,0,0,0,
                           0,0,0.2,0.2,0.2,0.2,0.2,0,0,0,
                           0,0,0,0.2,0.2,0.2,0.2,0.2,0,0,
                           0,0,0,0,0.2,0.2,0.2,0.2,0.2,0,
                           0,0,0,0,0,0.2,0.2,0.2,0.2,0.2,
                           0.2,0,0,0,0,0,0.2,0.2,0.2,0.2,
                           0.2,0.2,0,0,0,0,0,0.2,0.2,0.2),nrow=10,ncol=10))

ringStates = seq(1,10,1)
sensorStates = c("C1","C2","C3","C4","C5","C6","C7","C8","C9","C10")
model = initHMM(States=ringStates, Symbols = sensorStates, transProbs= transPr
obs, emissionProbs = emissionProbs)
```

Viktor Gustafsson vikgu708

Question 2 – Run Simulationas

Simulations were run with the simHMM function.

Code

```
## Assignment 2
nSims = 100
sims = simHMM(model, nSims)
obs = sims$observation
```

Question 3 – Compute smoothed and filtered distrubutions

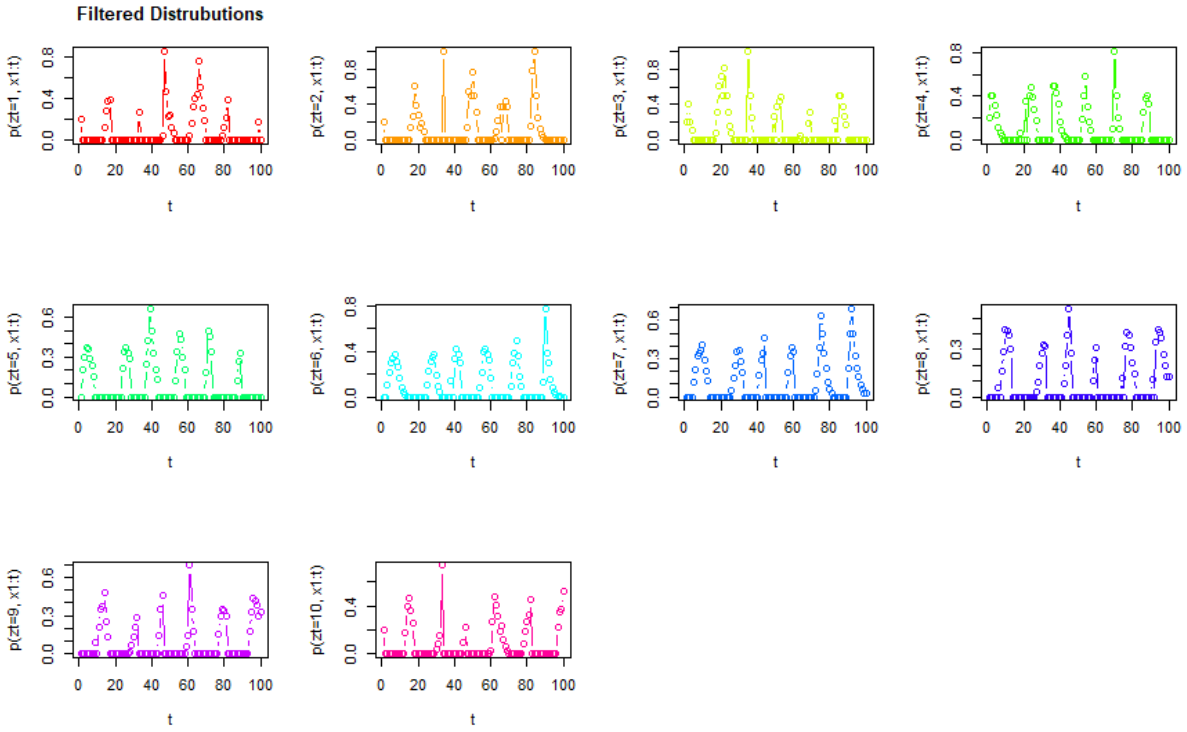
The smoothed and filtered distrubutions were computed by first using the forward and backwards functions to compute a(zt) and b(zt) for all time steps t, then normalizing a(zt) for filtered and multiplying a(zt)*b(zt) and then normalizing to get the smoothed distribution. The max probable path was gotten using the Viterbi function.

Code

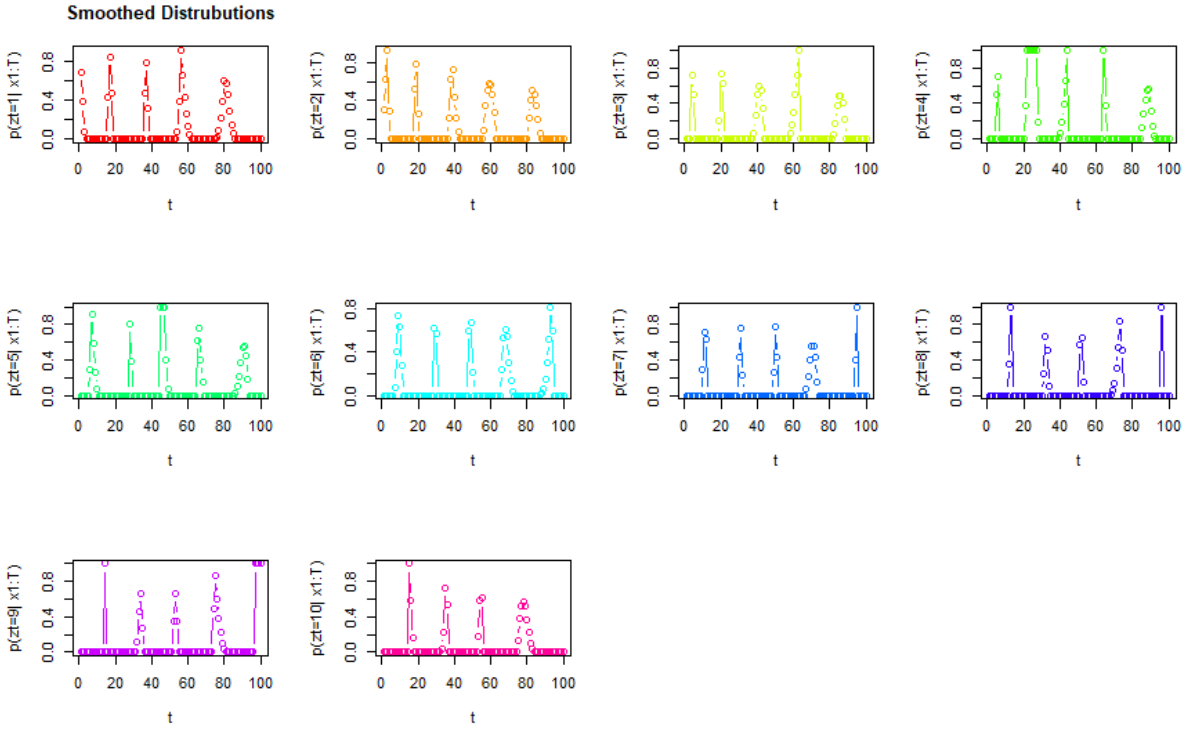
```
# Assignment 3
# Calculate filtered p(zt|x0:t) = a(zt)/sum(a(zt))
#           and smoothed p(zt|x0:T) = a(zt)*b(zt)/sum(a(zt)*b(zt))
#
# a(zt) = p(x0:t, zt) = p(xt|zt)Sum(a(zt-1)*p(zt|zt-1))   /sum over zt-1
# b(zt) = p(xt+1:T|zt) = sum(b(zt+1)*p(xt+1|zt+1)*p(zt+1|z))

alphas = exp(forward(model,obs))
betas = exp(backward(model,obs))

probs.filtered = sweep(alphas, 2, apply(alphas,2,sum), '/')
probs.smoothed =sweep(alphas*betas, 2, apply(alphas*betas,2,sum), '/')
maxProbPath = viterbi(model, obs)


#plots FILTERED
graphics.off()
par(mfrow=c(3,4))
for(i in ringStates ){
  plot(probs.filtered[i,], type="b", col = rainbow(10)[i], xlab="t", ylab=spri
ntf("p(zt=%i, x1:t)",i), main=ifelse(i==1,"Filtered Distrubutions",""))
}

#plots SMOOTHED
graphics.off()
par(mfrow=c(3,4))
for(i in ringStates ){
  plot(probs.smoothed[i,], type="b", col = rainbow(10)[i], xlab="t", ylab=spri
ntf("p(zt=%i| x1:T)",i), main=ifelse(i==1,"Smoothed Distrubutions",""))
}

#plots MOST PROBABLE PATH
```

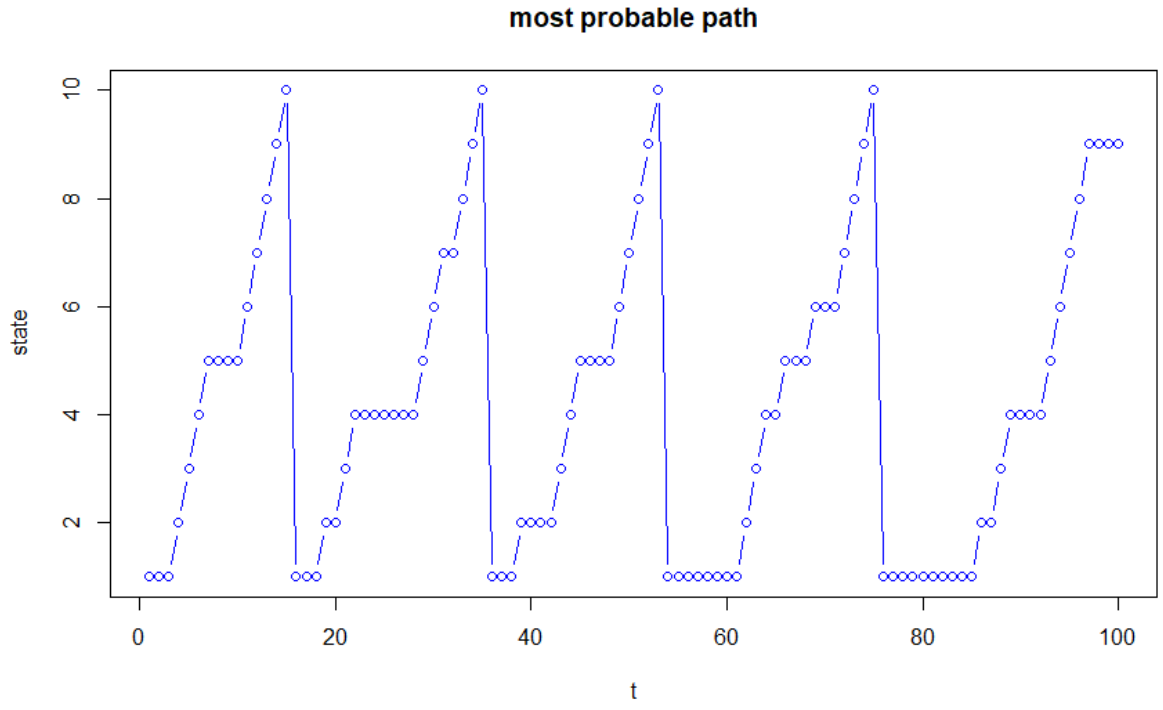
```
graphics.off()
par(mfrow=c(1,1))
plot(maxProbPath,type="b", xlab="t", ylab="state", main="most probable path",
col="blue")
```

Results



Viktor Gustafsson vikgu708





Viktor Gustafsson vikgu708

Question 4 – Compare accuaracies for smoothed, filtered, and most probable path

The accuracy for the each distribution was calculated thusly:

1. Find the most probable state for each timestep using which.max
2. Produce confusion matrix by comparing the predicted state with the real state using table
3. Compute accuracy by summing the diagonal of the confusion matrix and dividing by the sum of the entire confusion matrix.

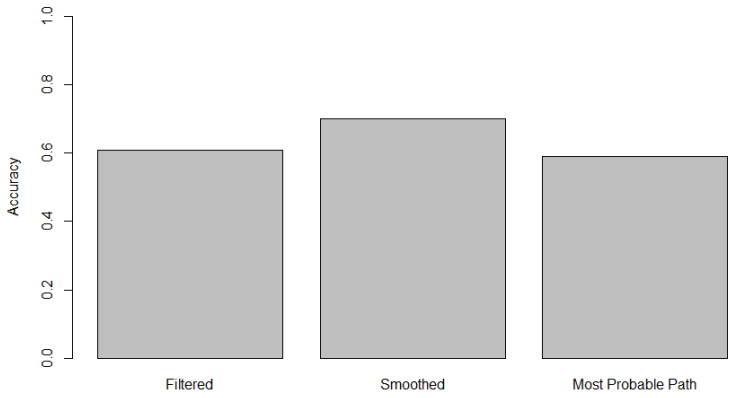
The accuracy for the most likely path is computed using only step 2 and 3.

Code

```
# Assignment 4

predStates.filtered = apply(probs.filtered,2,which.max)
predStates.smoothed = apply(probs.smoothed,2,which.max)

conf.maxProbPath = table(maxProbPath, sims$states)
conf.filtered = table(predStates.filtered,sims$states)
conf.smoothed = table(predStates.smoothed, sims$states)


acc.filtered = sum(diag(conf.filtered))/sum(conf.filtered)
acc.smoothed = sum(diag(conf.smoothed))/sum(conf.smoothed)
acc.maxProbPath = sum(diag(conf.maxProbPath))/sum(conf.maxProbPath)

print(paste("Accuracy filtered: ", acc.filtered, "Accuracy smoothed: ", acc.sm
oothed, "Accuracy max probable path: ", acc.maxProbPath))
barplot(c(acc.filtered, acc.smoothed, acc.maxProbPath), names.arg=c("Filtered"
, "Smoothed", "Most Probable Path"), ylab="Accuracy", ylim=c(0,1))
```

Results

```
"Accuracy filtered:  0.61 Accuracy smoothed:  0.7 Accuracy max probable pat
h:  0.59"
```



Question 5 – Compare accuracies for methods over simulation length

The code in the previous steps were reformatted into a function that returns the accuracies for the different methods for a given number of samples. The function was called several times, varying the amount of simulations from 10 to 400 with an increment of 20. The resulting accuracies are plotted.

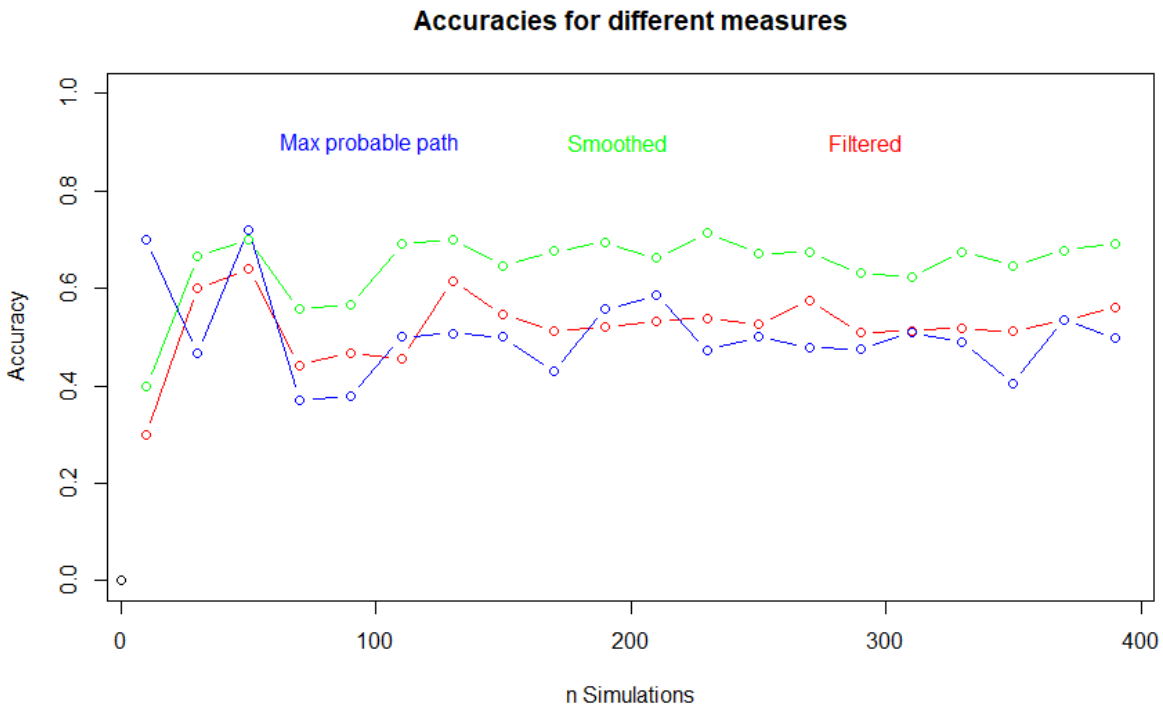
Code

```
library("HMM")
library("circlize")


transProbs=t(matrix(c(0.5,0.5,0,0,0,0,0,0,0,0,
                      0,0.5,0.5,0,0,0,0,0,0,0,
                      0,0,0.5,0.5,0,0,0,0,0,0,
                      0,0,0,0.5,0.5,0,0,0,0,0,
                      0,0,0,0,0.5,0.5,0,0,0,0,
                      0,0,0,0,0,0.5,0.5,0,0,0,
                      0,0,0,0,0,0,0.5,0.5,0,0,
                      0,0,0,0,0,0,0,0.5,0.5,0,
                      0,0,0,0,0,0,0,0,0.5,0.5,
                      0.5,0,0,0,0,0,0,0,0,0.5),10))

emissionProbs = t(matrix(c(0.2,0.2,0.2,0,0,0,0,0,0.2,0.2,
                           0.2,0.2,0.2,0.2,0,0,0,0,0,0.2,
                           0.2,0.2,0.2,0.2,0.2,0,0,0,0,0,
                           0,0.2,0.2,0.2,0.2,0.2,0,0,0,0,
                           0,0,0.2,0.2,0.2,0.2,0.2,0,0,0,
                           0,0,0,0.2,0.2,0.2,0.2,0.2,0,0,
                           0,0,0,0,0.2,0.2,0.2,0.2,0.2,0,
                           0,0,0,0,0,0.2,0.2,0.2,0.2,0.2,
                           0.2,0,0,0,0,0,0.2,0.2,0.2,0.2,
                           0.2,0.2,0,0,0,0,0,0.2,0.2,0.2),10))

ringStates = seq(1,10,1)
sensorStates = c("C1","C2","C3","C4","C5","C6","C7","C8","C9","C10")
model = initHMM(States=ringStates, Symbols = sensorStates, transProbs= transPr
obs, emissionProbs = emissionProbs)

getAccuracies = function(nSims, model){


  sims = simHMM(model, nSims)
  obs = sims$observation

  alphas = exp(forward(model,obs))
  betas = exp(backward(model,obs))

  probs.filtered = sweep(alphas, 2, apply(alphas,2,sum), '/')
  probs.smoothed = sweep(alphas*betas, 2, apply(alphas*betas,2,sum), '/')
```

```
  predStates.filtered = apply(probs.filtered,2,which.max)
  predStates.smoothed = apply(probs.smoothed,2,which.max)
  maxProbPath = viterbi(model, obs)

  conf.maxProbPath = table(maxProbPath, sims$states)
  conf.filtered = table(predStates.filtered, sims$states)
  conf.smoothed = table(predStates.smoothed, sims$states)


  acc.filtered = sum(diag(conf.filtered))/sum(conf.filtered)
  acc.smoothed = sum(diag(conf.smoothed))/sum(conf.smoothed)
  acc.maxProbPath = sum(diag(conf.maxProbPath))/sum(conf.maxProbPath)

  return(t(c(acc.filtered, acc.smoothed, acc.maxProbPath)))
  }

sims = seq(10,400,20)
maxSims = sims[length(sims)]
minSims = sims[1]

graphics.off()
par(mfrow=c(3,3))
accuracies = matrix(0,nrow=3, ncol=(length(sims)))
labels = c("Filtered", "Smoothed", "Max probable path")

for (i in 1:length(sims)){
  accuracies[,i] = getAccuracies(sims[i],model)
  print(sims[i]/maxSims)
  flush.console()
}

graphics.off()
plot(0,0, xlab="n Simulations", ylab="Accuracy", main="Accuracies for differen
t measures",xlim=c(sims[1],sims[length(sims)]),ylim=c(0,1))
for(i in 1:3 ){
  lines(x=sims, y=accuracies[i,], type="b", col = rainbow(3)[i])
  text(maxSims-maxSims/4*i,0.9,labels[i],col= rainbow(3)[i])

}
```

Viktor Gustafsson vikgu708

Results



Reasoning

Smoothed has a noticeably higher accuracy for all simulation sizes larger than 100.

The reason why smoothed is always larger than filtered is likely because smoothed has more information, as smoothed will calculate the probability for each state given all the emissions, whilst filtered will only calculate the probability for each state given the emissions thus far.

Smoothed has a higher accuracy than the most probable path because the most probable path has the restriction that each state must be legally accessible from the next, constraining the solution for this method. In comparison, the resulting sequence of maximum likelihood for the path of smoothed could be illegal, and thus smoothed can output a path that is of higher likelihood but is not possible in practice.

The most probable path can be more accurate than the smoothed distribution for low samples since the real path has lower variance, i.e., the actual path has less opportunities to diverge from the state transitions and emissions of highest likelihood and is therefore more likely to follow the most probable path.

Viktor Gustafsson vikgu708

Assignment 6

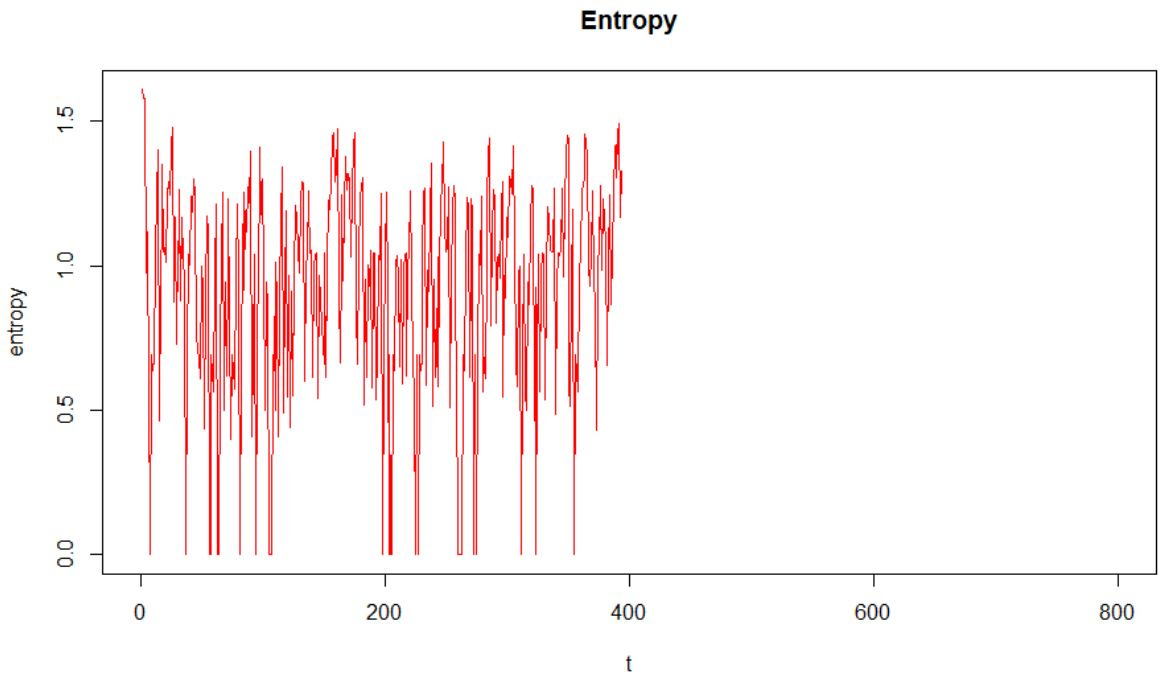
To figure out if more observations give us a better understanding of where the robot is, we simply sample many points from the network and calculate the filtered distribution of the robot, since the filtered distribution will equate the distribution we receive when we try to pinpoint the robots current location given all the previous emissions (measurements) for each time step t. We can then calculate the entropy for each timestep t and see if there is any pattern of the entropy increasing or decreasing. An increase in entropy of the filtered distribution equates to knowing less about the robots location, as a higher entropy means the distribution is more alike a uniform distribution (a uniform distribution contains no information of where the robot is located).

Code

```
library("HMM")
library("circlize")


transProbs=t(matrix(c(0.5,0.5,0,0,0,0,0,0,0,0,
                      0,0.5,0.5,0,0,0,0,0,0,0,
                      0,0,0.5,0.5,0,0,0,0,0,0,
                      0,0,0,0.5,0.5,0,0,0,0,0,
                      0,0,0,0,0.5,0.5,0,0,0,0,
                      0,0,0,0,0,0.5,0.5,0,0,0,
                      0,0,0,0,0,0,0.5,0.5,0,0,
                      0,0,0,0,0,0,0,0.5,0.5,0,
                      0,0,0,0,0,0,0,0,0.5,0.5,
                      0.5,0,0,0,0,0,0,0,0,0.5),10))

emissionProbs = t(matrix(c(0.2,0.2,0.2,0,0,0,0,0,0.2,0.2,
                           0.2,0.2,0.2,0.2,0,0,0,0,0,0.2,
                           0.2,0.2,0.2,0.2,0.2,0,0,0,0,0,
                           0,0.2,0.2,0.2,0.2,0.2,0,0,0,0,
                           0,0,0.2,0.2,0.2,0.2,0.2,0,0,0,
                           0,0,0,0.2,0.2,0.2,0.2,0.2,0,0,
                           0,0,0,0,0.2,0.2,0.2,0.2,0.2,0,
                           0,0,0,0,0,0.2,0.2,0.2,0.2,0.2,
                           0.2,0,0,0,0,0,0.2,0.2,0.2,0.2,
                           0.2,0.2,0,0,0,0,0,0.2,0.2,0.2),10))

ringStates = seq(1,10,1)
sensorStates = seq(1,10,1)
model = initHMM(States=ringStates, Symbols = sensorStates, transProbs= transPr
obs, emissionProbs = emissionProbs)
nSims = 800

sims = simHMM(model, nSims)
obs = sims$observation

alphas = exp(forward(model,obs))
betas = exp(backward(model,obs))
```

```
probs.filtered = sweep(alphas, 2, apply(alphas,2,sum), '/')
predStates.filtered = apply(probs.filtered,2,which.max)



entropies = rep(0,nSims)
for (i in 1:nSims){
  entropies[i] = entropy.empirical(probs.filtered[,i])
}
plot(x=1:nSims, y=entropies, type="l", col="red", xlab="t", ylab="entropy", ma
in="Entropy")
```

Result



Reasoning

The entropy oscillates between 0 and 1.5, with the mean around 1. This means that there is no pattern of knowing more or less about the robots position for more measurements. This makes intuitive sense, as both the emission function and the transition function contains a lot of inherent entropy with no evident convergence of states. Also, uncertainty of the robots position is the same regardless of the timestep as it is mostly connected to the uncertainty in the measurements which remains constant.

The entropy cannot be measured because of alpha values rounding to zeros for late values of t since alpha decreases for each timestep.

Viktor Gustafsson vikgu708

Assignment 7

Code is reused from assignment 3. We simply create the same HMM, sample 100, compute the filter distribution as per assignment 3. We then matrix multiply the transpose transition probability matrix with the transition vector. We then normalize this vector. The resulting vector is the probability for the next state.

Code

```
library("HMM")

transProbs = t(matrix(c(0.5,0.5,0,0,0,0,0,0,0,0,
                        0,0.5,0.5,0,0,0,0,0,0,0,
                        0,0,0.5,0.5,0,0,0,0,0,0,
                        0,0,0,0.5,0.5,0,0,0,0,0,
                        0,0,0,0,0.5,0.5,0,0,0,0,
                        0,0,0,0,0,0.5,0.5,0,0,0,
                        0,0,0,0,0,0,0.5,0.5,0,0,
                        0,0,0,0,0,0,0,0.5,0.5,0,
                        0,0,0,0,0,0,0,0,0.5,0.5,
                        0.5,0,0,0,0,0,0,0,0,0.5),10))

emissionProbs = t(matrix(c(0.2,0.2,0.2,0,0,0,0,0,0.2,0.2,
                           0.2,0.2,0.2,0.2,0,0,0,0,0,0.2,
                           0.2,0.2,0.2,0.2,0.2,0,0,0,0,0,
                           0,0.2,0.2,0.2,0.2,0.2,0,0,0,0,
                           0,0,0.2,0.2,0.2,0.2,0.2,0,0,0,
                           0,0,0,0.2,0.2,0.2,0.2,0.2,0,0,
                           0,0,0,0,0.2,0.2,0.2,0.2,0.2,0,
                           0,0,0,0,0,0.2,0.2,0.2,0.2,0.2,
                           0.2,0,0,0,0,0,0.2,0.2,0.2,0.2,
                           0.2,0.2,0,0,0,0,0,0.2,0.2,0.2),10))

ringStates = seq(1,10,1)
sensorStates = c("C1","C2","C3","C4","C5","C6","C7","C8","C9","C10")
model = initHMM(States=ringStates, Symbols = sensorStates, transProbs= transPr
obs, emissionProbs = emissionProbs)

## Assignment 2
nSims = 100
sims = simHMM(model, nSims)
obs = sims$observation

alphas = exp(forward(model,obs))
probs.filtered = sweep(alphas, 2, apply(alphas,2,sum), '/')

probs101 = t(transProbs) %*% probs.filtered[,100]
probs101 = probs101[,1]/sum(probs101[,1])
```

```
barplot(probs101,names.arg = ringStates, ylab="probs", ylim=c(0,1),xlab="State
 101")
print("### Probabilities ###")
for(i in 1:length(probs101)){
  print(sprintf("State %i: %g",i,probs101[i]))
}
```

Result

```
[1] "### Probabilities ###"
[1] "State 1: 0"
[1] "State 2: 0"
[1] "State 3: 0"
[1] "State 4: 0"
[1] "State 5: 0"
[1] "State 6: 0"
[1] "State 7: 0.108051"
[1] "State 8: 0.32839"
[1] "State 9: 0.391949"
[1] "State 10: 0.17161"
```