# Exam Practise

*Elvin Granat*

*2018-10-19*

## Forward Backward Algorithm

```
library("HMM")
require(knitr)
```

```
## Loading required package: knitr
```

Setup HMM

```
n = 10
startProbs = rep(0.1, n)
transProbs = diag(0.5, n) + diag(0.5, n)[c(2:n, 1),]
emissionProbs =
  diag(0.2, n)[c(2:n,1),] +
  diag(0.2, n)[c(3:n, 1:2),] +
  diag(0.2, n) +
  diag(0.2, n)[c(n, 1:(n-1)),] +
  diag(0.2, n)[c((n-1):n, 1:(n-2)),]
states = 1:n
symbols = 1:n
hmm = initHMM(states, symbols, startProbs = startProbs, transProbs = transProbs, emissionProbs = emissi
```

Simulate T = 100 steps

```
set.seed(123)
T = 100
sim = simHMM(hmm, T)
observations = sim$observation
states = sim$states
```

Implement Forward Backward algorithm

```
# of Z
startDensity = function(Z) {
  startProbs[Z]
}

# x given Z
emissionDensity = function(x, Z) {
  emissionProbs[Z,x]
}

# Z given z
transitionDensity = function(Z,z) {
  transProbs[z,Z]
}

get_alpha = function(x) {
  alpha = matrix(NA, n, T)

  # t = 1
  for(Z in 1:n) {
    alpha[Z,1] = emissionDensity(x[1],Z) * startDensity(Z)
  }

  # t = 2...T
  for(t in 2:T) {
    for(Z in 1:n) {
      alpha[Z,t] = emissionDensity(x[t], Z) * sum(
        sapply(1:n, function(z) {
          alpha[z,t-1] * transitionDensity(Z,z)
        })
      )
    }
  }
  return(alpha)
}

get_beta = function(x) {
  beta = matrix(NA, n, T)

  # t = T
  for(Z in 1:n) {
    beta[Z,T] = 1
  }

  # t = T-1...1
  for(t in (T-1):1){
    for(Z in 1:n) {
      beta[Z,t] = sum(
        sapply(1:n, function(z) {
          beta[z,t+1] * emissionDensity(x[t+1], z) * transitionDensity(z, Z)
        })
      )
```

```
    }
  }
  return(beta)
}
```

```
# Alpha & Beta
alpha = get_alpha(observations)
beta = get_beta(observations)

# Filtering & Smoothing
filtering = matrix(NA, n, T)
smoothing = matrix(NA, n, T)

alphaSum = apply(alpha, 2, sum)
alphaBetaSum = apply(alpha*beta, 2, sum)
for (t in 1:T) {
  filtering[,t] = alpha[,t] / alphaSum[t]
  smoothing[,t] = (alpha[,t] * beta[,t]) / alphaBetaSum[t]
}
```

### Viterbi algorithm

```
get_viterbi_path = function(x) {
  w = matrix(NA, n, T)
  psi = matrix(NA, n, T)
  z = rep(NA, T)

  # t = 1
  for(Z in 1:n) {
    w[Z,1] = log(startDensity(Z)) + log(emissionDensity(x[1], Z))
  }

  # t = 1..T-1
  for(t in 1:(T-1)) {
    for(Z in 1:n) {
      # omega
      w[Z, t+1] = log(emissionDensity(x[t+1], Z)) + max(sapply(1:n, function(z) {
          log(transitionDensity(Z,z)) + w[z, t]
        })
      )
      # psi
      psi[Z, t+1] = which.max(sapply(1:n, function(z) {
          log(transitionDensity(Z,z)) + w[z, t]
        })
      )
    }
  }
  # zT
  z[T] = which.max(w[,T])

  # zT-1...z1
  for(t in (T-1):1) {
    z[t] = psi[z[t+1],t+1]
  }
  return(z)
}
```

Most probable path

```
path = get_viterbi_path(observations)
```

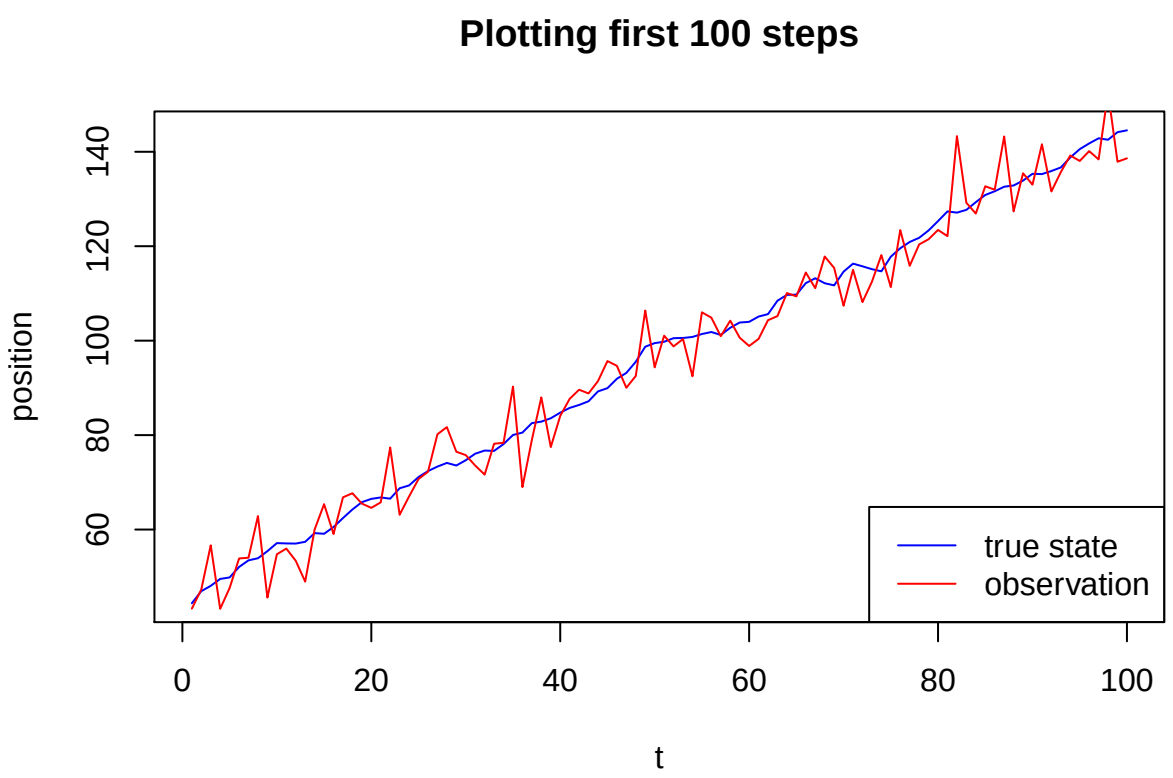
## Kalman Filter

Simulation

```
initTransitionSample = function() {
  rnorm(1, mean = 50, sd = 10)
}

transitionSample = function(x) {
  rnorm(1, mean = x + 1, sd = 1)
}

emissionSample = function(x) {
  rnorm(1, mean = x, sd = 5)
}

simulate = function() {
  T = 10000
  x = rep(NA, T)
  z = rep(NA, T)
  for(t in 1:T) {
    if(t == 1) x[t] = initTransitionSample()
    else x[t] = transitionSample(x[t-1])

    z[t] = emissionSample(x[t])
  }
  return(data.frame(x=x, z=z))
}
set.seed(123)
sim = simulate()
```

Visualization

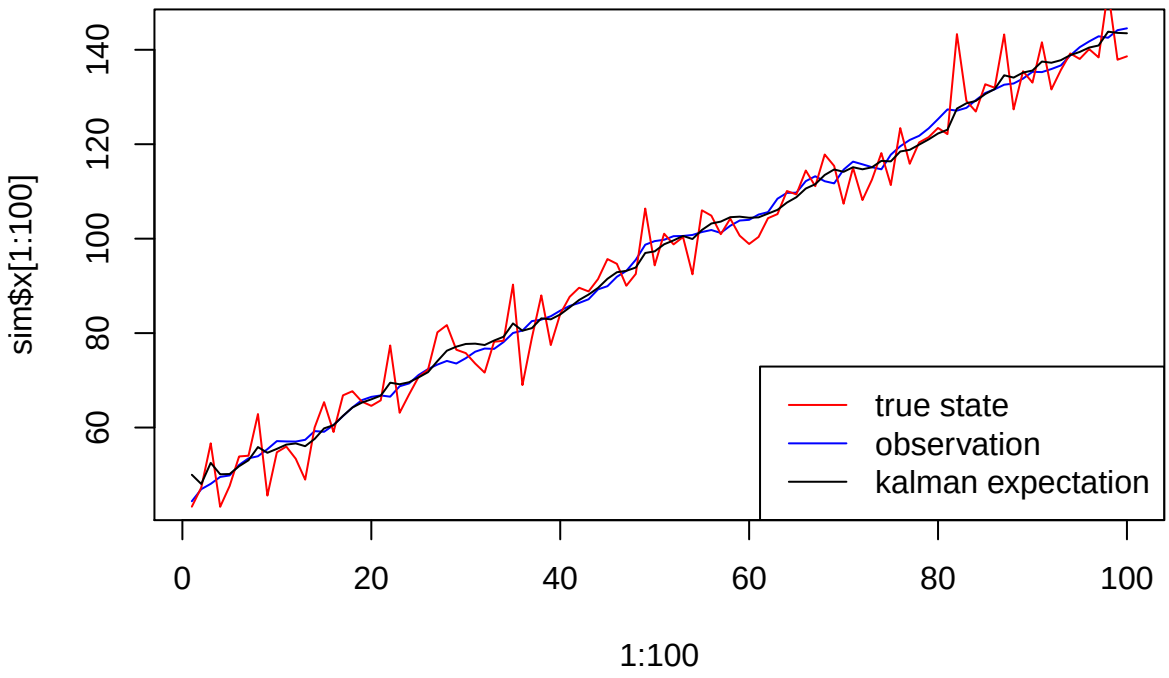
```
plot(1:100, sim$x[1:100], col="blue", type="l",
     main="Plotting first 100 steps", xlab="t", ylab="position")
lines(1:100, sim$z[1:100], col="red")
legend("bottomright", legend=c("true state", "observation"), col=c("blue", "red"), lty=c(1,1))
```


Plotting first 100 steps
position
60
80
100
120
140
0
20
40
60
80
100
t
true state
observation

Kalman Filter

```
get_kalman_path = function(z) {
  T = 10000
  mu = rep(NA, T)
  sigma = rep(NA, T)

  mu[1] = 50
  sigma[1] = 100

  A = 1
  B = 1
  u = 1
  R = 1
  Q = 25
  C = 1

  for (t in 2:T) {
    muBar = A * mu[t-1] + B * u
    sigmaBar = A * sigma[t-1] + R
    K = sigmaBar * C * 1 / ( C * sigmaBar * C + Q )
    mu[t] = muBar + K * (z[t] - C * muBar)
    sigma[t] = (1 - K * C) * sigmaBar
  }

  return(data.frame(mu = mu, sigma = sigma))
}

kalman = get_kalman_path(sim$z)
```

```
plot(1:100, sim$x[1:100], col="blue", type="l")
lines(1:100, sim$z[1:100], col="red")
lines(1:100, kalman$mu[1:100])
legend("bottomright", legend=c("true state", "observation", "kalman expectation"), col=c("red","blue","
       lty=c(1,1,1))
```




Errors

```
get_error = function(mean, x) {
  errors = abs(mean - x)
  errors.mean = round(mean(errors),3)
  errors.sd = round(sd(errors),3)
  return(data.frame(mean = errors.mean, sd = errors.sd))
}
kalman.error = get_error(kalman$mu, sim$x)
paste("Kalman Filtering error mean:", kalman.error$mean)
```

```
## [1] "Kalman Filtering error mean: 1.713"
```

```
paste("Kalman Filtering error standard deviation", kalman.error$sd)
```

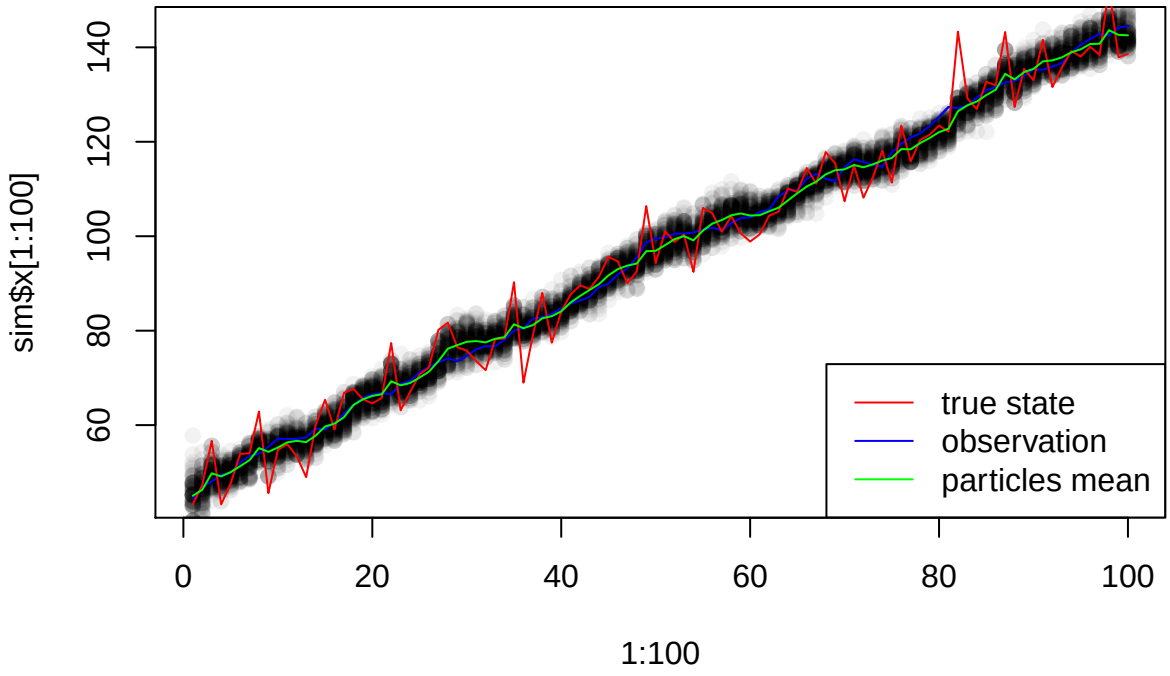
```
## [1] "Kalman Filtering error standard deviation 1.295"
```

### Particle filtering

```
w_density = function(z, x) {
  w = c()
  M = length(x)
  for(m in 1:M) {
    w[m] = dnorm(z, mean = x, sd = 5, log = FALSE)
  }
  return(w)
}

particle_draw = function(x_particles, w) {
  M = length(w)
  draw = c()
  draws = rmultinom(1, M, w)
  for (i in 1:M) {
    if (draws[i]) {
      for (j in 1:draws[i]) {
        draw = append(draw, x_particles[i])
      }
    }
  }
  return(draw)
}

particle_filtering = function(z, M) {
  set.seed(123)

  T = 10000
  X = matrix(0, nrow = M, ncol = T)
  Xbar = matrix(0, nrow = M, ncol = T)

  for (t in 1:T) {
    x_particles = c()
    w = c()
    for (m in 1:M) {
      if (t == 1) x_particles[m] = initTransitionSample()
      else x_particles[m] = transitionSample(X[m,t-1])

      w[m] = w_density(z[t], x_particles[m])
    }
    Xbar[,t] = x_particles
    X[,t] = particle_draw(Xbar[,t], w)
  }
  return(X)
}

M = 100
particles = particle_filtering(sim$z, M = M)
particle.mean = apply(particles, 2, mean)
```

Particle Filtering Plots

```
plot(1:100, sim$x[1:100], col="blue", type="l")
for(t in 1:100) {
  points(rep(t, M), particles[,t], pch=19, col=rgb(0,0,0,0.05))
}
lines(1:100, sim$x[1:100], col="blue")
lines(1:100, sim$z[1:100], col="red")
lines(1:100, particle.mean[1:100], col="green")
legend("bottomright", legend=c("true state", "observation", "particles mean"), col=c("red","blue","gree
       lty=c(1,1,1))
```




Particle Filtering Errors

```
particle.error = get_error(particle.mean, sim$x)
paste("Particle Filtering error mean:", particle.error$mean)
```

```
## [1] "Particle Filtering error mean: 1.762"
```

```
paste("Particle Filtering error standard deviation", particle.error$sd)
```

```
## [1] "Particle Filtering error standard deviation 1.332"
```

Compare performance

```
for(m in c(1, 10, 50, 100)) {
  startTime = Sys.time()
  if(m == 1) {
    kalman = get_kalman_path(sim$z)
```

```
    error = get_error(kalman$mu, sim$x)
  } else {
    particles = particle_filtering(sim$z, m)
    particle.mean = apply(particles, 2, mean)

    error = get_error(particle.mean, sim$x)
  }
  endTime = Sys.time()
  time = round(endTime - startTime,3)
  if(m == 1)
    print(paste("Kalban Filter", "Mean:", error$mean, "Standard Deviation:", error$sd, "Time:", time))
  else
    print(paste("Particle Filter", "M:", m, "Mean:", error$mean, "Standard Deviation:", error$sd, "Time
}
```

```
## [1] "Kalban Filter Mean: 1.713 Standard Deviation: 1.295 Time: 0.004"
## [1] "Particle Filter M: 10 Mean: 2.261 Standard Deviation: 1.7 Time: 0.779"
## [1] "Particle Filter M: 50 Mean: 1.815 Standard Deviation: 1.365 Time: 3.62"
## [1] "Particle Filter M: 100 Mean: 1.762 Standard Deviation: 1.332 Time: 7.081"
```

Kalban = Fast and effective. Performs well. Given one covariate the inverse is a simple division.

Particle = Linear time increase with increasing M, the mean and sd is improved for larger M. If you have the time, use particle filtering. Given that the transition/emission is normal kalban performs well.

## HMM with doors

```
states = 1:100
symbols = 1:2 # 1 is door, 2 is not door
startProbs = rep(0.01, 100)
transProbs = diag(0.1, 100) + diag(0.9, 100)[,c(100,1:99)]
emissionProbs = matrix(c(0.1,0.9), 100, 2, byrow=TRUE)
for (i in c(11,12,13,20,21,22,30,31,32)) {
  emissionProbs[i,] = c(0.9,0.1)
}
hmm = initHMM(states, symbols, startProbs = startProbs, transProbs = transProbs, emissionProbs = emissi

obs = c(1,1,1,2,2,2,2,2,2,2,2,2,2,2,2)
alpha = exp(forward(hmm ,obs))
alphaSum = apply(alpha, 2, sum)
forward = matrix(NA, 100, length(obs))
for(t in 1:length(obs)) {
  forward[,t] = alpha[,t] / alphaSum[t]
}
which.maxima<-function(x){ # This function is needed since which.max only returns the first maximum.
  return(which(x==max(x)))
}
res = apply(forward, 2, which.maxima)
kable(res)
```

| x | x | x | x | x | x | x | x | x | x | x | x | x | x | x |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 29 | 40 | 41 | 42 | 42 | 43 |
| 12 | 13 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 39 | | | | | |
| 13 | 21 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | | | | | | |
| 20 | 22 | | | | | | | | | | | | | |
| 21 | 31 | | | | | | | | | | | | | |
| 22 | 32 | | | | | | | | | | | | | |
| 30 | | | | | | | | | | | | | | |
| 31 | | | | | | | | | | | | | | |
| 32 | | | | | | | | | | | | | | |