Algorithms

```
library(HMM)
states = 1:10
symbols = 1:10
startProbs = rep(0.1, 10)
transProbs = diag(0.5, 10) + diag(0.5, 10)[,c(10,1:9)]
emissionProbs =
  diag(0.2, 10)[,c(9:10,1:8)] +
  diag(0.2, 10)[,c(10,1:9)] +
  diag(0.2, 10) +
  diag(0.2, 10)[,c(2:10,1)] +
  diag(0.2, 10)[,c(3:10,1:2)]
hmm = initHMM(states, symbols, startProbs = startProbs, transProbs = transProbs, emissionProbs = emissi

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

Forward Backward & Viterbi

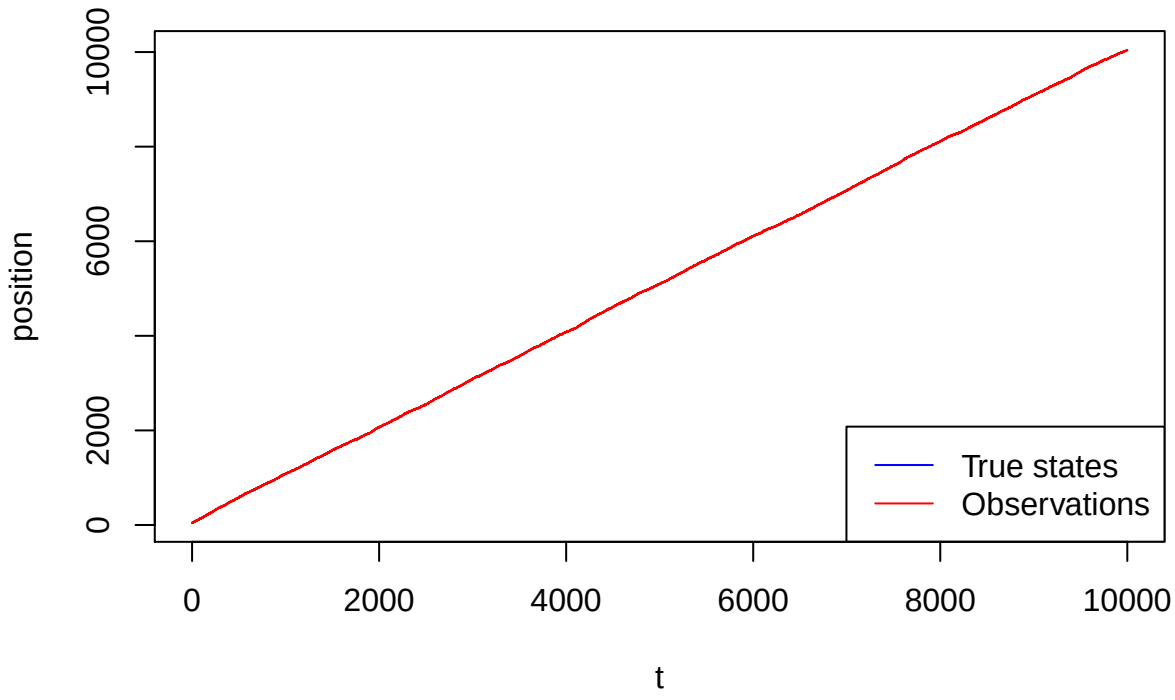
```
# Z0
init_density = function(Z) {
  startProbs[Z]
}

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

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

get_alpha = function(x) {
  alpha = matrix(NA, 10, T)

  # t = 1
  for(Z in 1:10) {
    alpha[Z,1] = emission_density(x[1], Z) * init_density(Z)
  }

  # t = 2..100
  for(t in 2:100) {
    for(Z in 1:10) {
      alpha[Z,t] = emission_density(x[t], Z) * sum(
        sapply(1:10, function(z){
          alpha[z,t-1] * transition_density(Z,z)
```

```
        })
      )
    }
  }

  return(alpha)
}

get_beta = function(x) {
  beta = matrix(NA, 10, T)

  # t = T
  for(Z in 1:10) {
    beta[Z,T] = 1
  }

  # t = T-1 ... 1
  for(t in (T-1):1) {
    for(Z in 1:10) {
      beta[Z,t] = sum(
        sapply(1:10, function(z){
          beta[z,t+1] * emission_density(x[t+1], z) * transition_density(z,Z)
        })
      )
    }
  }

  return(beta)
}

forward_backward = function(x) {
  alpha = get_alpha(x)
  beta = get_beta(x)

  return(list(alpha=alpha, beta=beta))
}

get_filtering = function(alpha) {
  filtering = matrix(NA, 10, T)

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

  return(filtering)
}

get_smoothing = function(alpha, beta) {
  smoothing = matrix(NA, 10, T)

  alphaBetaSum = apply(alpha * beta, 2, sum)
  for(t in 1:T) {
```

```
    smoothing[,t] = (alpha[,t] * beta[,t]) / alphaBetaSum[t]
  }

  return(smoothing)
}

get_viterbi = function(x) {
  w = matrix(NA, 10, T)
  psi = matrix(NA, 10, T)
  z = rep(NA, T)

  # omega 0
  for(Z in 1:10) {
    w[Z,1] = log(init_density(Z)) + log(emission_density(x[1], Z))
  }

  #
  for(t in 1:(T-1)) {
    for(Z in 1:10) {
      w[Z,t+1] = log(emission_density(x[t+1],Z)) + max(
        sapply(1:10, function(z){
          log(transition_density(Z,z)) + w[z,t]
        })
      )
      psi[Z,t+1] = which.max(
        sapply(1:10, function(z) {
          log(transition_density(Z,z)) + w[z,t]
        })
      )
    }
  }
  z[T] = which.max(w[,T])
  for(t in (T-1):1) {
    z[t] = psi[z[t+1],t+1]
  }
  return(z)
}


fb = forward_backward(observations)
alpha = fb[["alpha"]]
beta = fb[["beta"]]
filtering = get_filtering(alpha)
smoothing = get_smoothing(alpha, beta)
viterbi = get_viterbi(observations)
```

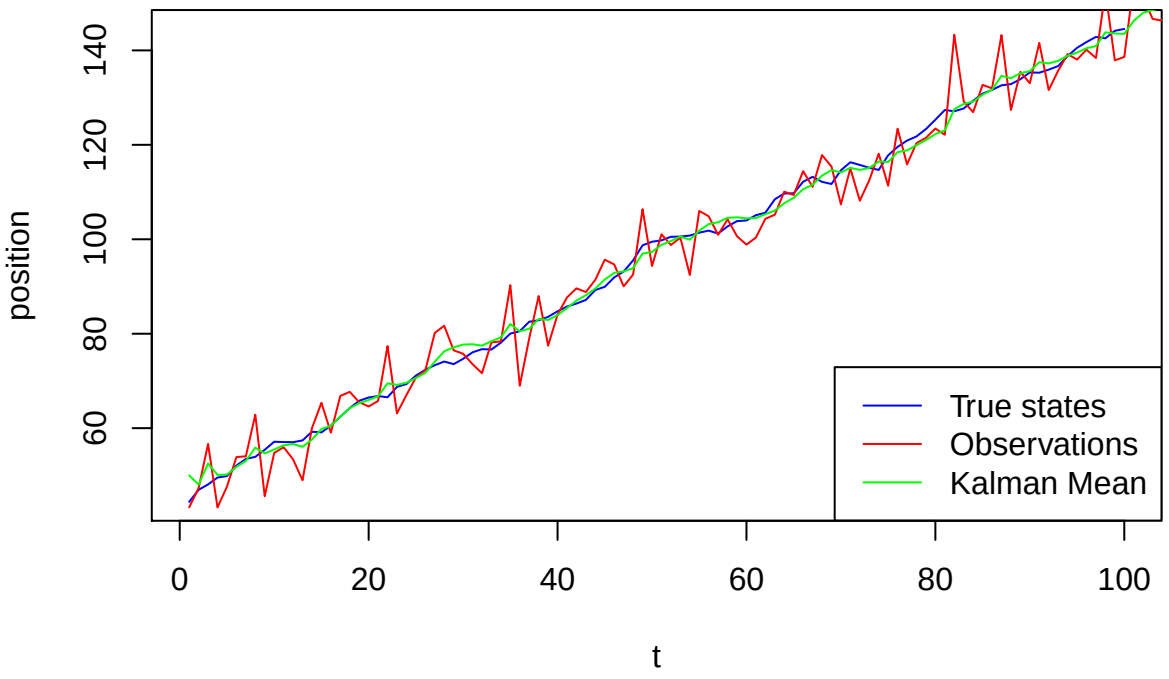
Kalman Filter

```
T = 10000

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

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

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

simulate = function() {
  x = c()
  z = c()
  for(t in 1:T) {
    if(t == 1) x[t] = initSample()
    else x[t] = transitionSample(x[t-1])

    z[t] = emissionSample(x[t])
  }
  return(data.frame(x=x, z=z))
}

set.seed(123)
sim = simulate()
observations = sim$z
states = sim$x
plot(1:T, states, type="l", col="blue", xlab="t", ylab="position")
lines(1:T, observations, col="red")
legend("bottomright", legend=c("True states", "Observations"), col=c("blue", "red"), lty=c(1,1))
```




```
get_kalman_filter = function(z) {
  # Setup
  mu0 = 50
  Sigma0 = 100
  A = 1
  B = 1
  u = 1
  R = 1
  C = 1
  Q = 25

  mu = c()
  muBar = c()
  Sigma = c()
  SigmaBar = c()

  # Kalman
  mu[1] = mu0
  Sigma[1] = Sigma0
  for(t in 2:T) {
    muBar[t] = A * mu[t-1] + B * u
    SigmaBar[t] = A * Sigma[t-1] * A + R
    K = SigmaBar[t] * C * (C * SigmaBar[t] * C + Q)^-1
    mu[t] = muBar[t] + K * (z[t] - C * muBar[t])
    Sigma[t] = (1 - K * C) * SigmaBar[t]
  }

  return(data.frame(mu = mu, V = Sigma))
}

kalman = get_kalman_filter(observations)
plot(1:100, states[1:100], type="l", col="blue", xlab="t", ylab="position")
```

```
lines(1:T, observations, col="red")
lines(1:T, kalman$mu, col="green")
legend("bottomright", legend=c("True states", "Observations", "Kalman Mean"), col=c("blue", "red", "gre
```

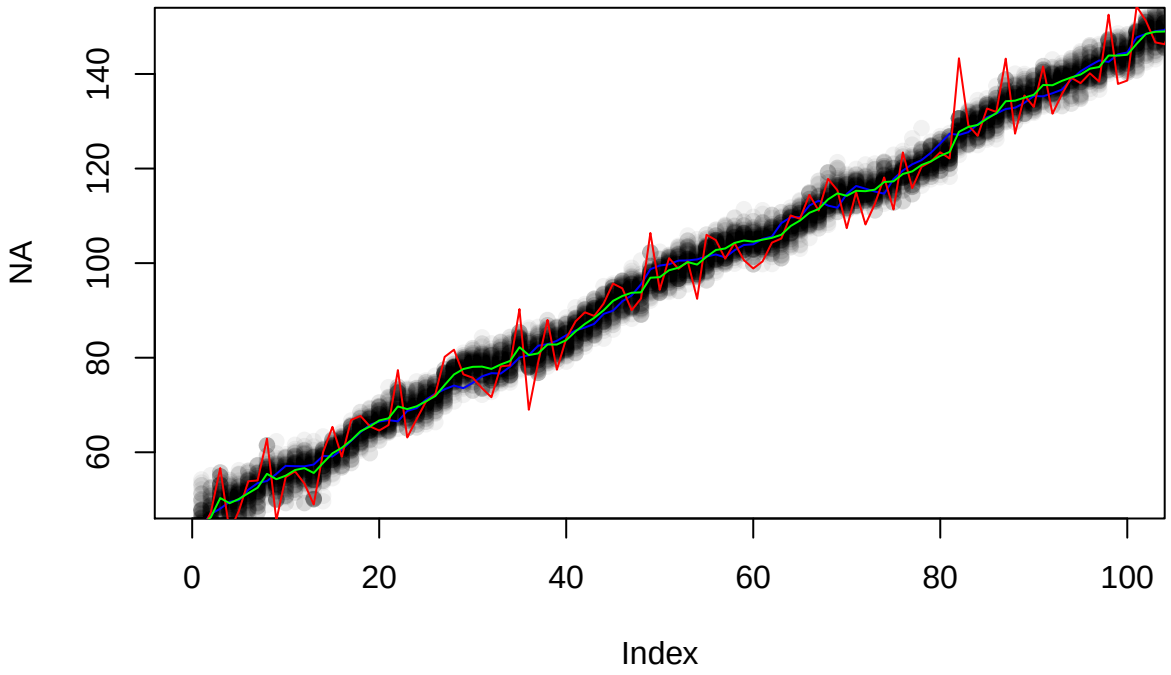


##

Particle filter

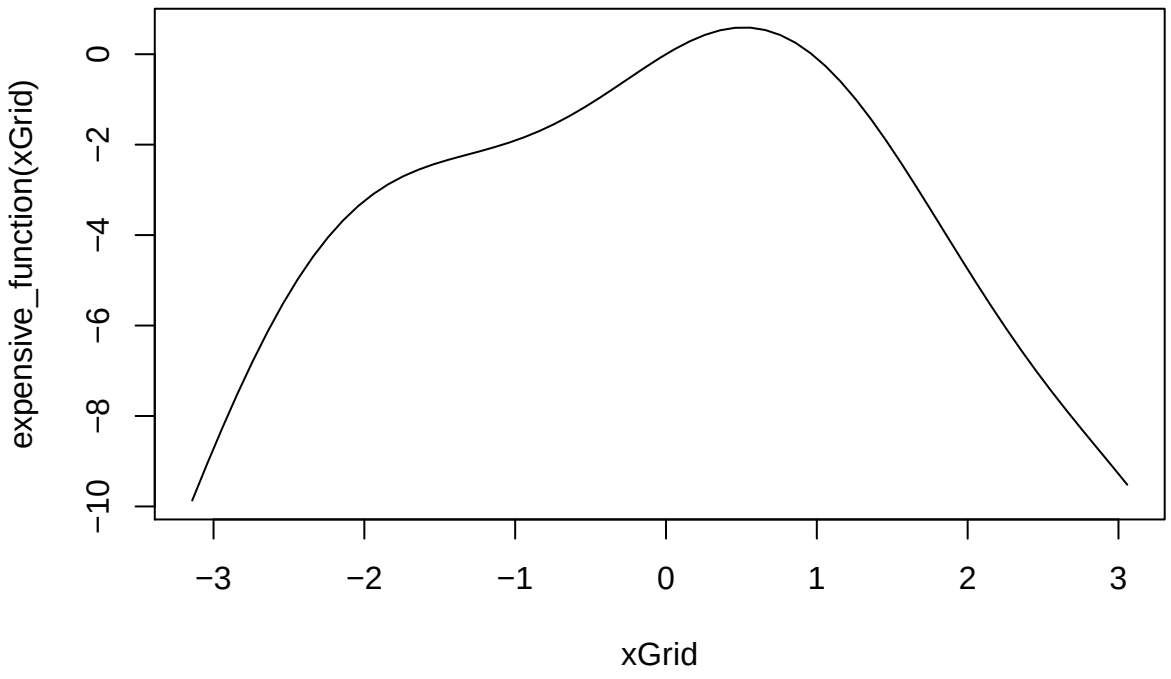
```
get_particle_filter = function(z) {
  M = 100
  X = matrix(NA, M, T)
  for(t in 1:T) {
    x = c()
    w = c()
    for(m in 1:M) {
      if(t == 1) x[m] = rnorm(1, mean = 50, sd = 10)
      else x[m] = rnorm(1, mean = X[m,t-1] + 1, sd = 1)
      w[m] = dnorm(z[t], mean = x[m], sd = 5)
    }
    X[,t] = sample(x, size = m, replace = TRUE, prob = w)
  }
  return(X)
}
set.seed(123)
particles = get_particle_filter(observations)
```

```
plot(NA, xlim=c(0,100), ylim=c(50,150))
for(t in 1:T) {
  points(rep(t, 100), particles[,t], pch=19, col=rgb(0,0,0,0.05))
}
lines(1:T, states, type="l", col="blue")
lines(1:T, observations, col="red")
lines(1:T, apply(particles, 2, mean), col="green")
```

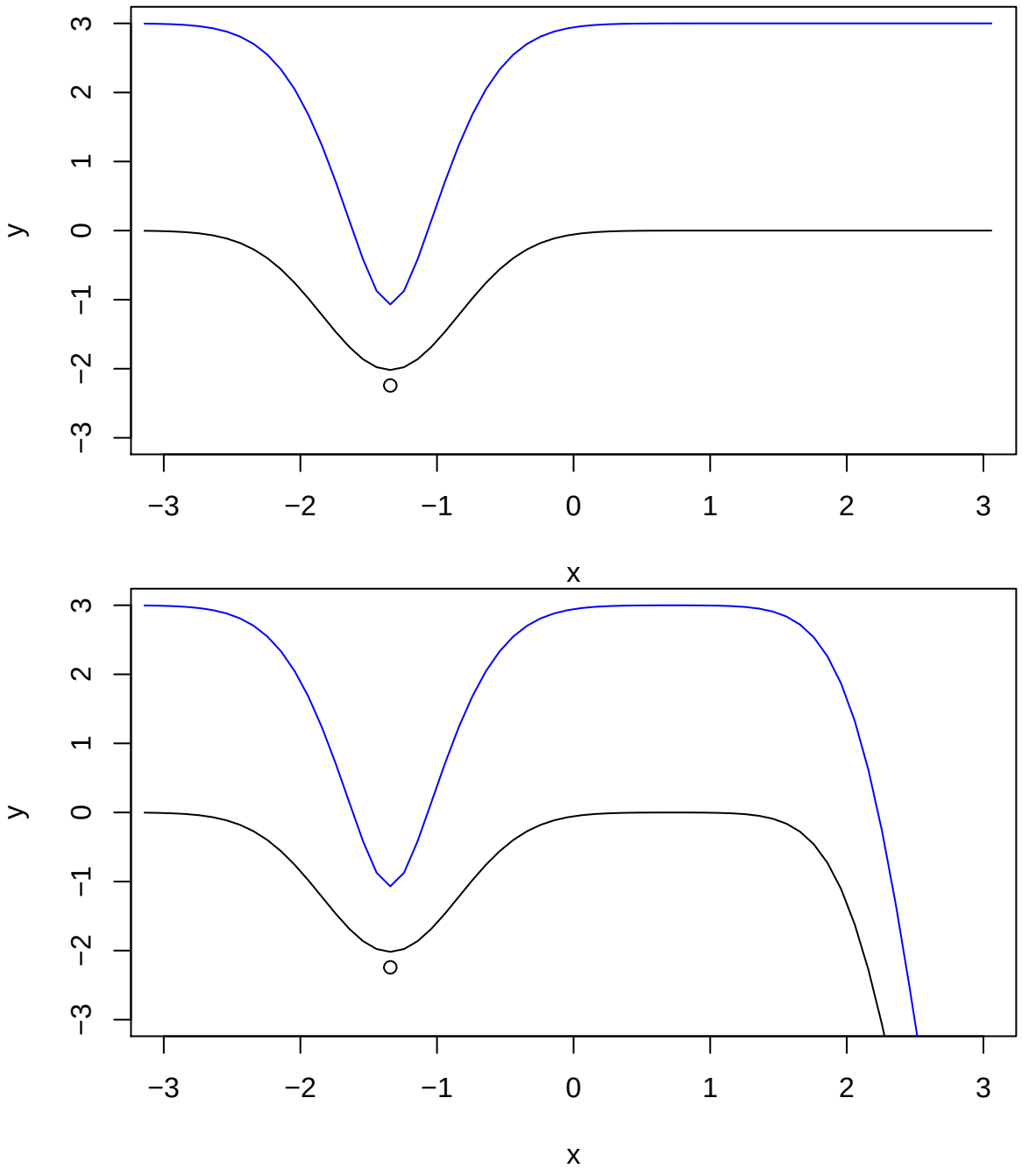


Bayesian Optimization

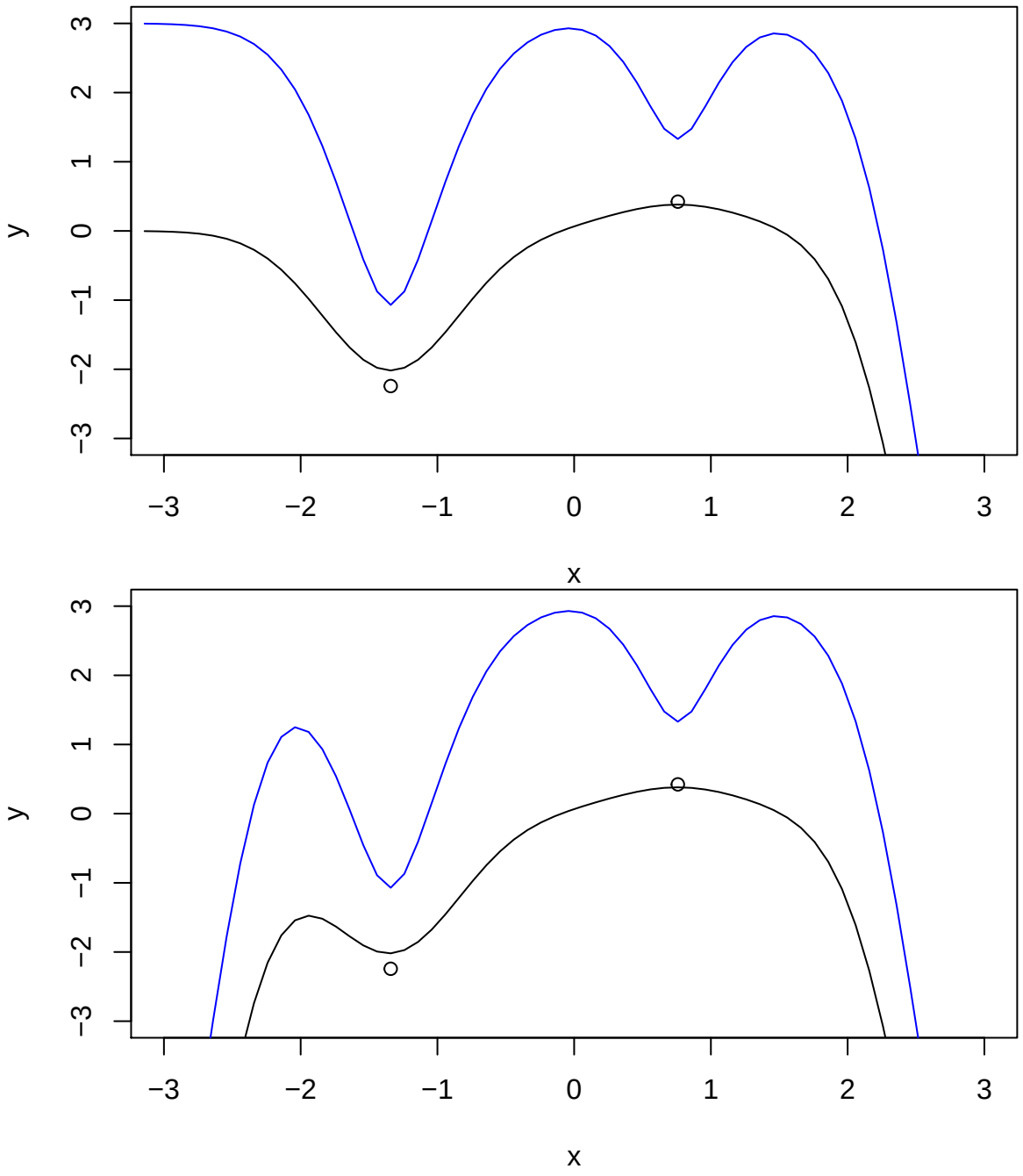
```
library("kernlab")
xGrid = seq(-pi, pi, by=0.1)
expensive_function = function(x) {
  -x^2 + sin(2*x)
}
plot(xGrid, expensive_function(xGrid), type="l")
```

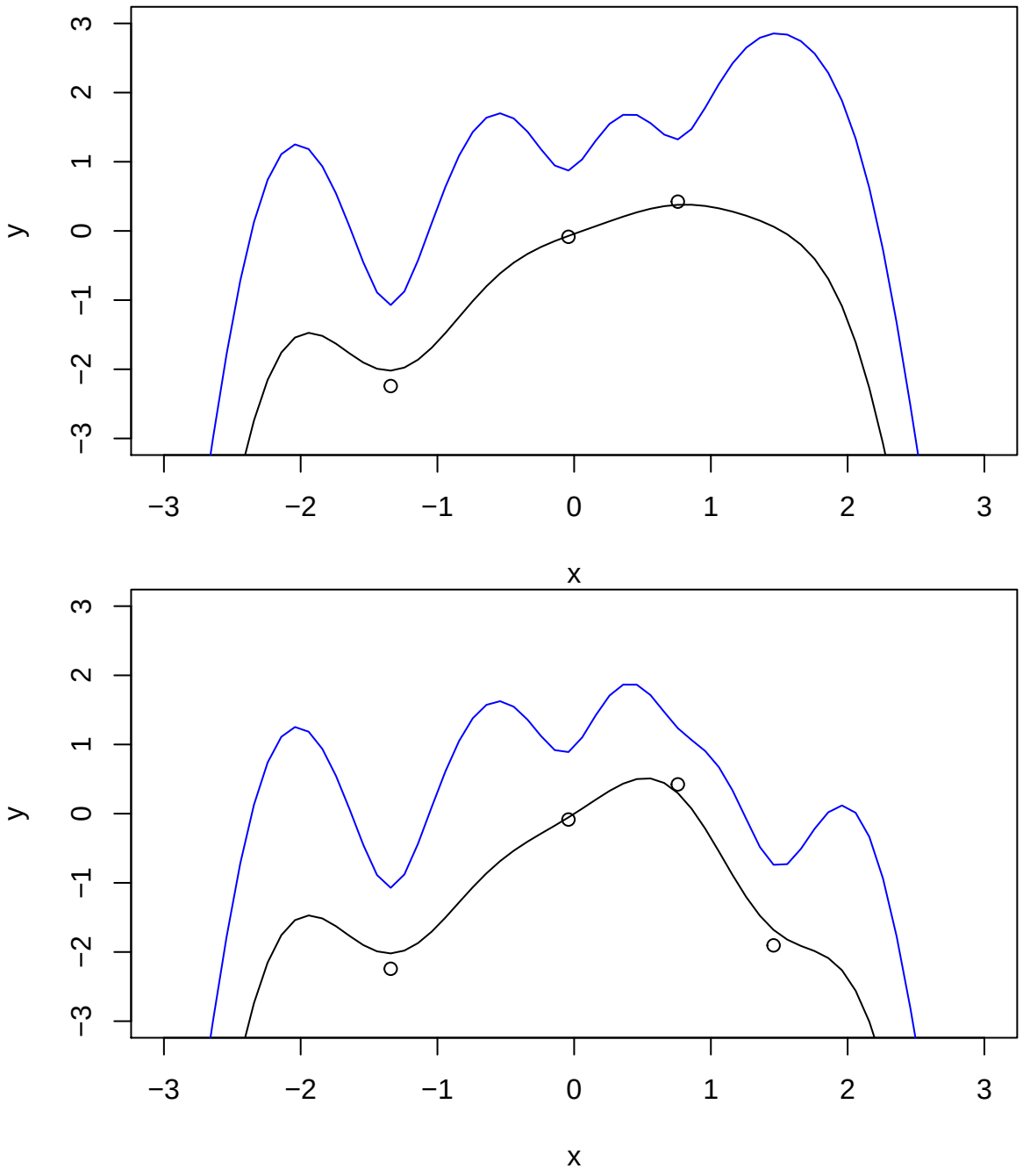


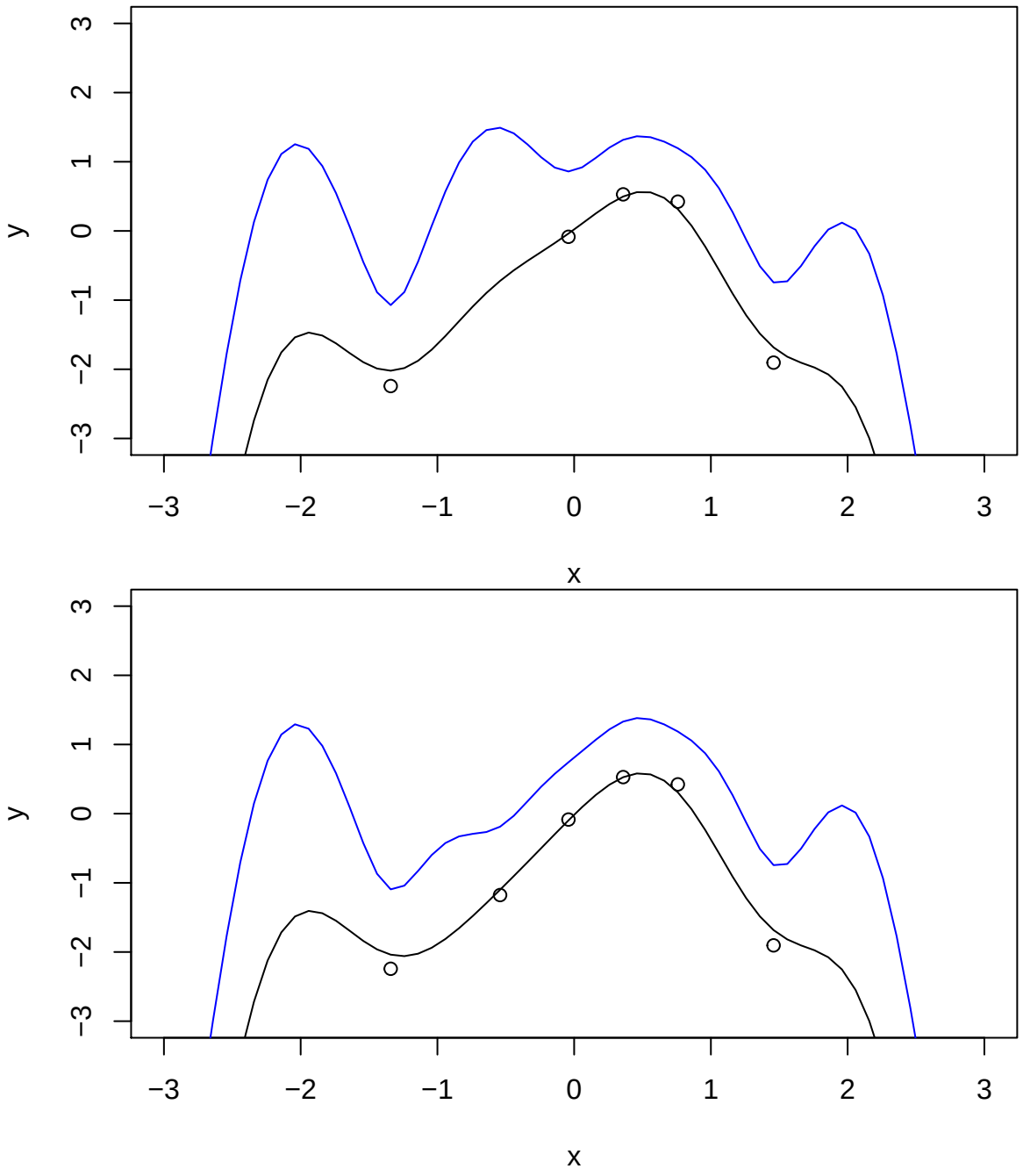
```r
set.seed(123)

SquaredExpKernel = function(x1, x2, sigmaF, l) {
  n1 = length(x1)
  n2 = length(x2)
  K = matrix(NA, n1, n2)
  for(i in 1:n2) {
    for(j in 1:n1) {
      K[j,i] = sigmaF^2 * exp(-0.5 * ( (x2[i] - x1[j])/l)^2 )
    }
  }
  return(K)
}

posteriorGP = function(X, y, XStar, sigmaNoise, K, ...) {
  n = length(X)
  I = diag(n)
  L = t(chol(K(X,X, ...) + sigmaNoise^2 * I))
  KStar = K(X, XStar, ...)

  # Predictive Mean
  alpha = solve(t(L), solve(L, y))
  fStarBar = t(KStar) %*% alpha

  # Predictive Variance
  v = solve(L, KStar)
  V = K(XStar, XStar, ...) - t(v) %*% v

  return(data.frame(mean = fStarBar, variance = diag(V)))
}

get_bayesian_optimization = function(f, k, xGrid) {
  kappa = 1
  x = c()
  y = c()
  for(i in 1:10) {
    if (i == 1) x[i] = sample(xGrid, 1)
    else x[i] = xGrid[which.max(posterior$mean + kappa * sqrt(posterior$variance))]

    y[i] = f(x[i])

    posterior = posteriorGP(X = x, y = y, XStar = xGrid, sigmaNoise = 1, K = k, sigmaF = 3, l = 0.5)
    plot(x,y, xlim=c(-3,3), ylim=c(-3,3))
    lines(xGrid, posterior$mean)
    lines(xGrid, posterior$mean + kappa * sqrt(posterior$variance), col="blue")
  }
  return(x[which.max(y)])
}

max = get_bayesian_optimization(f = expensive_function, SquaredExpKernel, xGrid)
```


y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
x

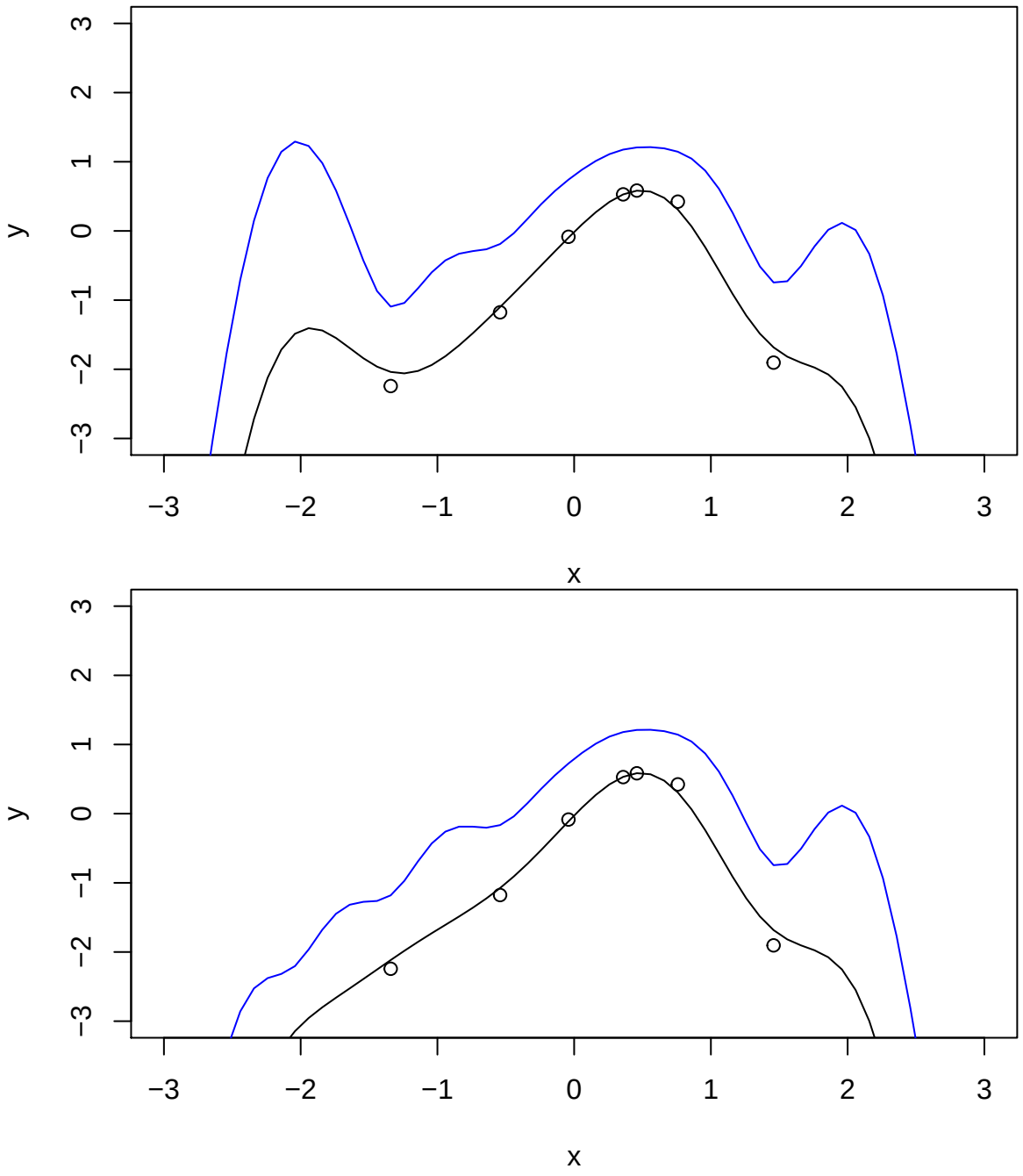
y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
x


y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
x
y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
x


y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
x
y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
x


y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
x

y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
x



Gaussian Process

```
library("kernlab")
load("GPdata.RData")
SEKernel = function(sigmaF, l) {
  kernel = function(x, y) {
    r = sqrt(crossprod(x-y))
    sigmaF^2 * exp(-0.5 * (r/l)^2)
  }
  class(kernel) = "kernel"
```

```
  return(kernel)
}
```

```
xStar = seq(min(x), max(x), by=0.01)
kernel = SEKernel(sigmaF = 1, l = 1)
sigmaN = 0.2
n = length(x)
I = diag(n)

Kxx = kernelMatrix(kernel = kernel, x = x, y = x)
Kss = kernelMatrix(kernel = kernel, x = xStar, y = xStar)
Kxs = kernelMatrix(kernel = kernel, x = x, y = xStar)
Ksx = t(Kxs)

postMean = Ksx %*% solve(Kxx + sigmaN^2 * I) %*% y
postVariance = Kss - Ksx %*% solve(Kxx + sigmaN^2 * I) %*% Kxs

plot(x,y)
lines(xStar, postMean, col="red")
lines(xStar, postMean + 1.96 * sqrt(diag(postVariance)))
lines(xStar, postMean - 1.96 * sqrt(diag(postVariance)))
```