Written exercise 2

Assignment 2 – 02433 Hidden Markov Models – Anders Hørsted (s082382)

In this exercise two datasets containing noisy observations of log-population sizes are analyzed. The datasets are fitted to a theta logistic model for population growth and local decoding is calculated. The theta logistic model can be written as a state-space model

$$P_t = P_{t-1} + r_0 \left(1 - \left[\frac{\exp(P_{t-1})}{K}\right]^\theta\right) + e_t \tag{1}$$

$$X_t = P_t + u_t \tag{2}$$

where P_t is the log-population size and $e_t \sim N(0, Q)$, $u_t \sim N(0, R)$ are iid. and mutually independent. The constant K is the log-population size when stability is reached. Equation (**??**) expresses the fact that whenever $P_t > \log(K)$ then $P_{t+1} < P_t$ with high probability. Therefore it is assumed that K, θ and r_0 are all positive. Since Q and R are variances they are also assumed positive. All five parameters are combined in the parameter vector $\boldsymbol{\lambda} = (\theta, r_0, K, Q, R)$.

a) Plotting the data

To start the analysis the two datasets are plotted and shown in figure **??**. From the figure it is seen that the two datasets have a very similar overall structure. For both dataset $P_1 \approx 3$ and stationarity is reached around $K \approx \exp(7) \approx 1000$.

b) Approximating the SSM by a HMM

By assuming that the state-space P_t is bounded it can be discretizised and by this discretization the state-space model can be expressed as an Hidden markov model instead.

For the state-space model given in equation **??** and **??** it is now assumed that $P_t \in [2.1, 8.4]$ and that the state-space is partitioned into $m = 250$ intervals $\Omega_i = (b_{i-1}, b_i)$ where $i = 1, \ldots, m$. The width of each interval is then given by

$$w = \frac{8.4 - 2.1}{250} = 0.0252$$

and the boundaries can be expressed as $b_i = 2.1 + wi$. If $P_t \in \Omega_i$ it is said to be in state i which corresponds to the discrete state-space variable C_t being equal to i. Each discrete state i is represented by the midpoint p_i of the interval (b_{i-1}, b_i) given by

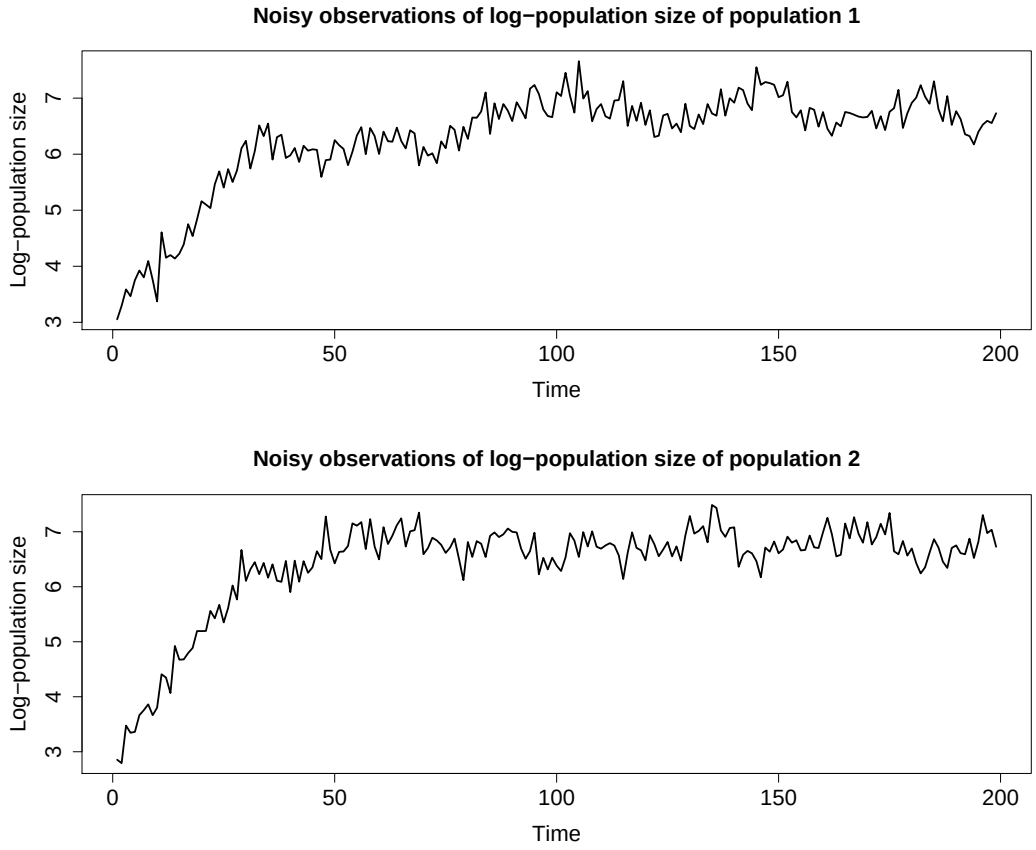

Figure 1: Plot of the two datasets

$p_i = 2.1 + w(i - 0.5)$. This links the discrete, integer valued state-space C_t to the original continous state-space P_t.

To express the state-space model as a Hidden markov model the state dependent distibutions should be found. The distribution of $X_t \,|\, C_t = i$ is found by replacing P_t with the representation point p_i in equation (**??**). Combined with $u_t \sim N(0, R)$ gives that

$$X_t \,|\, C_t = i \;\sim\; N(p_i, R)$$

or equivalently

$$p(x_t \,|\, C_t = i) = \frac{1}{\sqrt{2\pi R}} \exp\left(-\frac{(x_t - p_i)^2}{2R}\right)$$

Apart from the state dependent distributions the transition probabilities $\Pr(P_t \in \Omega_j \,|\, P_{t-1} \in \Omega_i)$ should also be found. Conditioning on P_{t-1} being in state i ($C_{t-1} = i$) can be represented by replacing P_{t-1} with p_i in equation **??**. Using the fact that $e_t \sim N(0, Q)$

then gives

$$P_t \,|\, C_{t-1} = i \;\sim\; N\left(\mu_i, Q\right) \quad \text{with} \quad \mu_i = p_i + r_0 \left(1 - \left[\frac{\exp(p_i)}{K}\right]^{\theta}\right)$$

Letting $n(\bullet, \mu, \sigma^2)$ denote the Gaussian pdf with mean μ and variance σ^2 the transition probabilities can then be calculated by

$$\Pr(P_t \in \Omega_j \,|\, C_{t-1} = i) = \Pr(C_t = j \,|\, C_{t-1} = i) = \int_{\Omega_j} n(p_t, \mu_i, Q)\, dp_t$$

This integral could be calculated by using the cumulative distribution function for the normal distribution, but as mentioned on page 13 in (**?**) this is more computational expensive than approximating the integral using the trapezoidal rule for integration. Using the trapezoidal rule gives

$$\Pr(C_t = j \,|\, C_{t-1} = i) \approx \frac{w}{2}\left(n(b_{j-1}, \mu_i, Q) + n(b_j, \mu_i, Q)\right)$$

c) Computing the likelihood

In this section a function that evaluates the likelihood of the discretized state-space model is created. Since the discretized state-space model is a HMM the algorithm described on page 47 in (**?**) can be used. To implement the algorithm a few details need to be handled.

The markov chain of the discretized SSM can not be assumed stationary so an initial state distribution is needed. This is done by using the approximation mentioned on the bottom of page 7 in (**?**).

Also the constraints on the parameters $\boldsymbol{\lambda} = (\theta, r_0, K, Q, R)$ need to be handled. Since all parameters are constrained to be positive reals the logarithm of the natural parameters will be unconstrained which gives the working parameters

$$\boldsymbol{\tau} = \log(\boldsymbol{\lambda})$$

Going from working parameters to natural parameters is then

$$\boldsymbol{\lambda} = \exp(\boldsymbol{\tau})$$

(where log and exp is element-wise functions).

Finally the state dependent distributions are continous so if one of the means μ_i equals one of the observations x_t and we then let $Q \to 0$ the likelihood will be unbounded. The implementation in this section will not take care of this detail though.

Implementation

The implementation of the log likelihood function is shown in code listing **??**. The function takes the working parameters in a vector and returns the minus log likelihood value. This makes it easy to use the build-in unconstrained minization function `nlm` to maximize the likelihood. The implementation shown in code listing **??** is taken a bit out of context. The function definition is part of the parent function `get.HMM.model` for which the source code is shown in appendix **??**. The reason that the `HMM.mllk` and `HMM.mle` functions are created by another function is to let them easily share variables (eg. the data `x`, the boundary points `b.i`, the representative points `p.i` etc.) without cluttering the global namespace. The concept that functions created within a parent function has access to the variables of the parent function – even after the parent function has returned its value – is called closure.

```
HMM.mllk = function(working.params, ...) {
    params = exp(working.params)
    R = params[R.ind]

    Ps = outer(x, p.i, dnorm, sqrt(R))
    Ps[is.na(Ps)] = 1
    gamma = calc.gamma(params)

    phi = Ps[1,]
    phi.sum = sum(phi)
    llk = log(phi.sum)
    phi = phi/phi.sum

    for (t in 2:T) {
        phi = phi %*% gamma * Ps[t,]
        phi.sum = sum(phi)
        llk = llk + log(phi.sum)
        phi = phi/phi.sum
    }

    return(-llk)
}
```

Code Listing 1: Function that calculates the minus log likelihood of the discretized SSM. The function depends on variables defined in the closure of the function definition. See appendix **??** *for the complete source code*

The `HMM.mllk` function also uses a helper function `calc.gamma` to calculate the transition probability matrix. The `calc.gamma` function is also defined within the `get.HMM.model` function and therefore also have access to eg. the representative points `p.i`.

```
calc.gamma = function (params) {
    theta = params[theta.ind]
    r0 = params[r0.ind]
    K = params[K.ind]
    Q = params[Q.ind]

    mu = p.i + r0*(1 - (exp(p.i)/K)^theta)
    trans.prob = matrix(0, m, m)
```

```
        for (i in 1:m) {
            row = (dnorm(b.i-w, mu[i], sqrt(Q)) + dnorm(b.i, mu[i], sqrt(Q)))*w/2
            trans.prob[i,] = row/sum(row)
        }

        return(trans.prob)
    }
```

Code Listing 2: The helper function `calc.gamma` used in the `HMM.mllk` function

To actually calculate the minus log likelihood for a specific choice of parameters we call

```
source('src/loaddata.R')
source('src/functions.R')
source('src/model.R')

params = c(0.5, 0.2, 1000, 0.01, 0.05)
model = get.HMM.model(dat1, 250, c(2.1, 8.4))
mllk = model$mllk(params)
```

Code Listing 3: Using the implementation in code listing **??**

The function `get.HMM.model` is called with the data `dat1`, the number of intervals, and the state-space boundaries. The returned list contains all functions related to the model and these functions have access to the data due to the closure created by the `get.HMM.model` function.

d) Estimating parameters

In this section a function to estimate the parameters of the model by maximum likelihood is implemented. The function will also return an estimate of the standard errors of the maximum likelihood estimates. The error estimates are calculated from the hessian of minus the log likelihood at the minimum using equation 3.2 in (**?**) to express the hessian wrt. the natural parameters $\boldsymbol{\lambda}$. The implementation is shown in code listing **??** and the function `HMM.mle` is also defined within the function `get.HMM.model` which gives it access to the data `x` due to the closure explained in the previous section.

```
    HMM.mle = function(initial.params, ...) {
        initial.working.params = log(initial.params)
        res = nlm(HMM.mllk, initial.working.params, x=x, hessian=TRUE, ...)
        mle = exp(res$estimate)
        mllk = res$minimum
        num.params = length(mle)
        AIC = 2*(mllk+num.params)
        num.obs = sum(!is.na(x))
        BIC = 2*mllk+num.params*log(num.obs)
        # M is the coordinate shift matrix as defined
        # in equation 3.2 in Zucchini 2009
        M = diag(mle)
```

```
        var.cov = inv.hessian.from.working.hessian(res$hessian, M)
        list(mle=mle, code=res$code, mllk=mllk, AIC=AIC, BIC=BIC,
             var.cov=var.cov, nlm.res=res)
    }
```

Code Listing 4: Function to estimate the parameters by maximum likelihood

The function `HMM.mle` uses the helper function `inv.hessian.from.working.hessian` to calculate an estimate of the variance-covariance matrix. The function `inv.hessian.from.working.hessian` is shown in code listing **??**

```
inv.hessian.from.working.hessian = function (working.hessian, M) {
    inv.working.hessian = solve(working.hessian)
    return(t(M) %*% inv.working.hessian %*% M)
}
```

Code Listing 5: Function to transform the hessian from working parameters to natural parameters

Estimating parameters for dataset 1

To estimate the parameters of dataset 1 some initial parameters need to be determined. From figure **??** it seems that $K \approx e^7 \approx 1000$. The other parameters are a bit harder to determine but it seems plausible to keep θ and r_0 small (0.5-1). After some fiddling the starting parameters* was chosen as

$$\boldsymbol{\lambda}_0 = \begin{pmatrix} 0.5 & 0.2 & 1000 & 0.01 & 0.05 \end{pmatrix}$$

Calculating the maximum likelihood gave the parameter estimates

$$\widehat{\boldsymbol{\lambda}} = \begin{pmatrix} 4.62 \cdot 10^{-1} & 1.42 \cdot 10^{-1} & 8.23 \cdot 10^{2} & 9.05 \cdot 10^{-3} & 4.07 \cdot 10^{-2} \end{pmatrix}$$

with $-\ell = 3.114^{\dagger}$ and AIC $= 16.228$. Using the hessian returned from the `nlm` function an approximate variance-covariance matrix was calculated as

$$\widehat{\mathrm{Var}[\widehat{\boldsymbol{\lambda}}]} = \begin{pmatrix} 1.83 \cdot 10^{-1} & -3.62 \cdot 10^{-2} & 4.23 & 2.88 \cdot 10^{-4} & -1.75 \cdot 10^{-4} \\ -3.62 \cdot 10^{-2} & 7.86 \cdot 10^{-3} & -1.35 & -4.09 \cdot 10^{-5} & 2.38 \cdot 10^{-5} \\ 4.23 & -1.35 & 9.52 \cdot 10^{3} & -1.48 \cdot 10^{-2} & 7.87 \cdot 10^{-3} \\ 2.88 \cdot 10^{-4} & -4.09 \cdot 10^{-5} & -1.48 \cdot 10^{-2} & 7.43 \cdot 10^{-6} & -4.51 \cdot 10^{-6} \\ -1.75 \cdot 10^{-4} & 2.38 \cdot 10^{-5} & 7.87 \cdot 10^{-3} & -4.51 \cdot 10^{-6} & 2.76 \cdot 10^{-5} \end{pmatrix}$$

*To ensure that the maximum found isn't a local maximum much smaller than the global maximum, some other starting parameters was tried. They all gave the same maximum which indicates that the parameter estimates may be the maximum likelihood estimates.

†The likelihood value at the maximum is very high. This is probably due to the fact that the state dependent probabilities are calculated as point values of the normal density function instead of "real" probabilities.

From the approximate variance-covariance matrix the correlation of all pairs of parameter estimates can be calculated. The three numerically largest was

$$\text{Corr}[\widehat{\theta}, \widehat{r_0}] = -0.954 \qquad \text{and} \qquad \text{Corr}[\widehat{Q}, \widehat{R}] = -0.315 \qquad \text{and} \qquad \text{Corr}[\widehat{\theta}, \widehat{Q}] = 0.247$$

The large negative correlation between $\widehat{\theta}$ and $\widehat{r_0}$ can be explained by looking at the expression

$$r_0 \left(1 - \left[\frac{\exp\left(P_{t-1}\right)}{K}\right]^{\theta}\right)$$

An increase in θ will always make the above expression numerically larger. To counteract this we can decrease r_0. Therefore datasets that makes $\widehat{\theta} > \text{E}[\widehat{\theta}]$ will with high probability make $\widehat{r_0} < \text{E}[\widehat{r_0}]$, since the above expression represents the expected change in population level and the true expected change in population level is independent of how we choose to parameterize our model.

That $\widehat{Q}$ and $\widehat{R}$ are negatively correlated can be explained by substituting equation (**??**) into equation (**??**), which gives a term $u_t + e_t$ that is normal distributed with variance $Q + R$.

From the variance-covariance matrix confidence intervals for the parameter estimates can be calculated and the result is shown in table **??**.

	Estimate	95% Conf.Int		Std.Dev
θ	0.4615	-0.3769	1.3000	0.4278
r_0	0.1423	-0.0315	0.3160	0.0886
K	822.9574	631.7430	1014.1718	97.5584
Q	0.0090	0.0037	0.0144	0.0027
R	0.0407	0.0304	0.0510	0.0053

Table 1: Parameter estimates with 95% confidence intervals calculated from the inverse hessian at the maximum

It is seen that the confidence intervals for both $\widehat{\theta}$ and $\widehat{r_0}$ includes 0 and in general the standard deviation of the estimators are large relative to the corresponding estimates.

Estimating parameters for dataset 2

From figure **??** is seems that dataset 2 is from a population that is similar to the population of dataset 1. Therefore the same initial parmeters are used

Calculating the maximum likelihood estimates gives the parameter estimate

$$\widehat{\boldsymbol{\lambda}} = \begin{pmatrix} 1.05 & 1.32 \cdot 10^{-1} & 8.86 \cdot 10^{2} & 8.09 \cdot 10^{-3} & 4.26 \cdot 10^{-2} \end{pmatrix}$$

with $-\ell = 0.574$ and AIC $= 11.148$. Comparing the estimates for dataset 2 with the estimates for dataset 1 it is seen that most parameter estimates are almost equal. Only the estimate of θ differs; the estimate from dataset 2 is twice the size of the estimate

from dataset 1, but notice that $\widehat{\theta}$ from dataset 2 is within the 95% confidence interval of $\widehat{\theta}$ from dataset 1.

The approximate variance-covariance matrix is found as

$$\widehat{\mathrm{Var}[\widehat{\boldsymbol{\lambda}}]} = \begin{pmatrix} 3.02 \cdot 10^{-1} & -1.44 \cdot 10^{-2} & 3.61 & 8.22 \cdot 10^{-4} & -6.17 \cdot 10^{-4} \\ -1.44 \cdot 10^{-2} & 1.05 \cdot 10^{-3} & -2.47 \cdot 10^{-1} & -2.23 \cdot 10^{-5} & 1.56 \cdot 10^{-5} \\ 3.61 & -2.47 \cdot 10^{-1} & 2.36 \cdot 10^{3} & 2.43 \cdot 10^{-4} & -2.28 \cdot 10^{-3} \\ 8.22 \cdot 10^{-4} & -2.23 \cdot 10^{-5} & 2.43 \cdot 10^{-4} & 1.10 \cdot 10^{-5} & -9.09 \cdot 10^{-6} \\ -6.17 \cdot 10^{-4} & 1.56 \cdot 10^{-5} & -2.28 \cdot 10^{-3} & -9.09 \cdot 10^{-6} & 3.49 \cdot 10^{-5} \end{pmatrix}$$

and the numerically largest correlations are found to be

$$\mathrm{Corr}[\widehat{\theta}, \widehat{r_0}] = -0.807 \qquad \text{and} \qquad \mathrm{Corr}[\widehat{Q}, \widehat{R}] = -0.465 \qquad \text{and} \qquad \mathrm{Corr}[\widehat{\theta}, \widehat{Q}] = 0.452$$

It is seen that it is the same pairs of estimators as for dataset 1.

From the variance-covariance matrix confidence intervals for the parameter estimates can be calculated and the result is shown in table **??**

	Estimate	95% Conf.Int		Std.Dev
θ	1.0458	-0.0313	2.1230	0.5496
r_0	0.1324	0.0689	0.1959	0.0324
K	885.7958	790.4891	981.1025	48.6259
Q	0.0081	0.0016	0.0146	0.0033
R	0.0426	0.0311	0.0542	0.0059

Table 2: Parameter estimates for dataset 2 with 95% confidence intervals calculated from the inverse hessian at the maximum

It is seen that the 95% confidence interval for θ still includes 0 but the standard errors for both r_0 and K has decreased compared with the standard errors calculated from dataset 1.

e) Local decoding

In this section local decoding for the two fitted HMMs will be calculated. The implementation is based on A.2.2, A.2.5 and A.2.6 in (**?**) and can be seen in appendix **??**. Using the implementation local decodings was calculated and plotted which is shown in figure **??**.

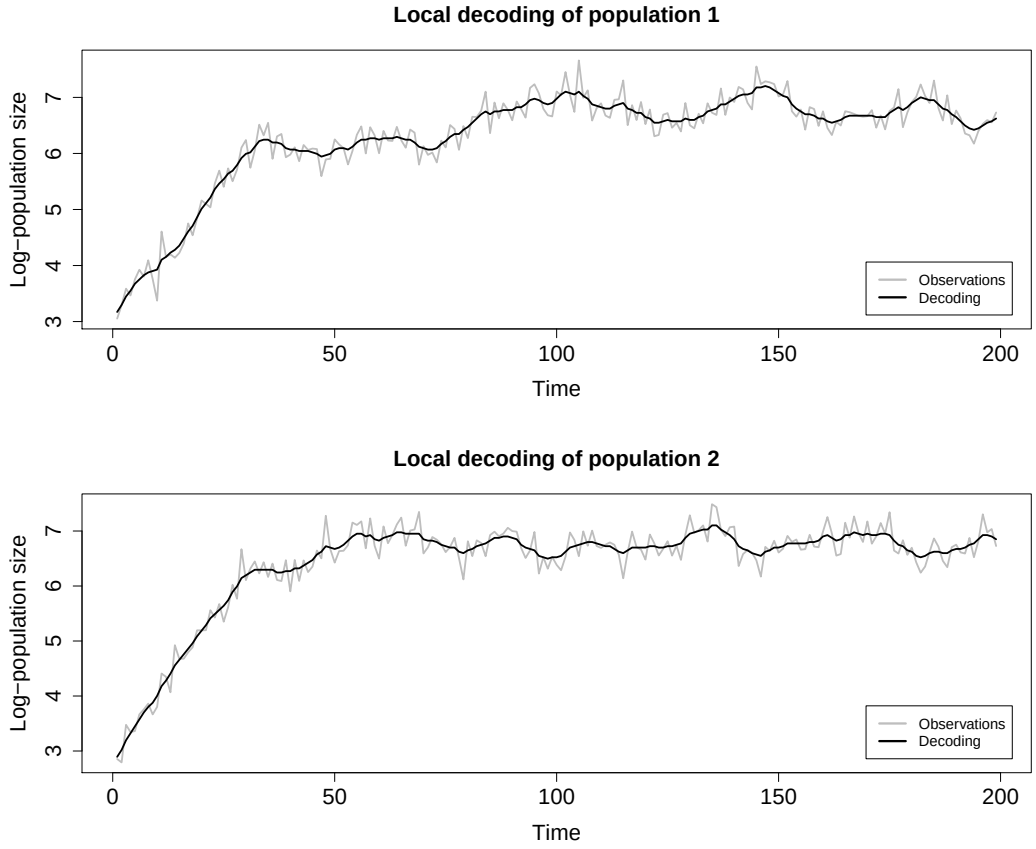

Figure 2: Local decoding for the two datasets using the parameters fitted by maximum likelihood

A Appendices

All R code created for this assignment is included here. All source code incl. latex code for this report can be found at `https://github.com/alphabits/dtu-spring-2012/tree/master/02433/assignment-2`

A.1 Function creating model functions

```
get.HMM.model = function (x, m, state.bounds=c(2.1, 8.4)) {
    T = length(x)
    w = diff(state.bounds)/m
    i = 1:m
    p.i = state.bounds[1] + w*(i - 0.5)
    b.i = state.bounds[1] + w*i
    theta.ind = 1
    r0.ind = 2
    K.ind = 3
    Q.ind = 4
    R.ind = 5
```

```
calc.gamma = function (params) {
    theta = params[theta.ind]
    r0 = params[r0.ind]
    K = params[K.ind]
    Q = params[Q.ind]

    mu = p.i + r0*(1 - (exp(p.i)/K)^theta)
    trans.prob = matrix(0, m, m)

    for (i in 1:m) {
        row = (dnorm(b.i-w, mu[i], sqrt(Q)) + dnorm(b.i, mu[i], sqrt(Q)))*w/2
        trans.prob[i,] = row/sum(row)
    }

    return(trans.prob)
}

HMM.mllk = function(working.params, ...) {
    params = exp(working.params)
    R = params[R.ind]

    Ps = outer(x, p.i, dnorm, sqrt(R))
    Ps[is.na(Ps)] = 1
    gamma = calc.gamma(params)

    phi = Ps[1,]
    phi.sum = sum(phi)
    llk = log(phi.sum)
    phi = phi/phi.sum

    for (t in 2:T) {
        phi = phi %*% gamma * Ps[t,]
        phi.sum = sum(phi)
        llk = llk + log(phi.sum)
        phi = phi/phi.sum
    }

    return(-llk)
}

HMM.mle = function(initial.params, ...) {
    initial.working.params = log(initial.params)
    res = nlm(HMM.mllk, initial.working.params, x=x, hessian=TRUE, ...)
    mle = exp(res$estimate)
    mllk = res$minimum
    num.params = length(mle)
    AIC = 2*(mllk+num.params)
    num.obs = sum(!is.na(x))
    BIC = 2*mllk+num.params*log(num.obs)
    # M is the coordinate shift matrix as defined
    # in equation 3.2 in Zucchini 2009
    M = diag(mle)
    var.cov = inv.hessian.from.working.hessian(res$hessian, M)
    list(mle=mle, code=res$code, mllk=mllk, AIC=AIC, BIC=BIC,
         var.cov=var.cov, nlm.res=res)
}

HMM.log.forward.backward = function (params) {
    R = params[R.ind]
    log.alpha = log.beta = matrix(NA, m, T)
```

```
        Ps = outer(x, p.i, dnorm, sqrt(R))
        Ps[is.na(Ps)] = 1

        gamma = calc.gamma(params)

        # Calculate alpha
        phi = Ps[1,]
        phi.sum = sum(phi)
        lscale = log(phi.sum)
        phi = phi/phi.sum
        log.alpha[,1] = log(phi)+lscale
        for (t in 2:T) {
            phi = phi%*%gamma*Ps[t,]
            phi.sum = sum(phi)
            lscale = lscale+log(phi.sum)
            phi = phi/phi.sum
            log.alpha[,t] = log(phi)+lscale
        }

        # Calculate beta
        log.beta[,T] = rep(0, m)
        phi = rep(1/m, m)
        lscale = log(m)
        for (t in (T-1):1) {
            phi = gamma%*%(Ps[t+1,]*phi)
            log.beta[,t] = log(phi)+lscale
            phi.sum = sum(phi)
            phi = phi/phi.sum
            lscale = lscale+log(phi.sum)
        }

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

    HMM.cond.state.probs = function(params) {
        fb = HMM.log.forward.backward(params)
        la = fb$log.alpha
        lb = fb$log.beta
        c = max(la[,T])
        llk = c+log(sum(exp(la[,T]-c)))
        stateprobs = matrix(NA, ncol=T, nrow=m)
        for (i in 1:T) {
            stateprobs[,i] = exp(la[,i]+lb[,i]-llk)
        }
        return(stateprobs)
    }

    HMM.local.decoding = function(params) {
        stateprobs = HMM.cond.state.probs(params)
        ild = rep(NA, T)
        for (t in 1:T) {
            ild[t] = which.max(stateprobs[,t])
        }
        return(ild)
    }

    return(list(mle=HMM.mle, mllk=HMM.mllk, gamma=calc.gamma,
                local.decoding=HMM.local.decoding, p.i=p.i, b.i=b.i,
                data=x))
}
```

A.2 Estimating parameters and decodings

```
source('src/loaddata.R')
source('src/functions.R')
source('src/model.R')

# Define initial parameters to test and define
# a model object for each dataset
params = list(c(0.5, 0.2, 1000, 0.01, 0.05),
              c(0.8, 0.8, 100, 1, 2))
models = list(get.HMM.model(dat1, 250, c(2.1, 8.4)),
              get.HMM.model(dat2, 250, c(2.1, 8.4)))
mles = list()

# Just a helper function
get.label = function (param.num, model.num) {
    return(sprintf('cont-dataset-%s-param-%s', model.num, param.num))
}

# Estimate parameters for each combination of initial parameters
# and models
for (param.num in 1:length(params)) {
    for (model.num in 1:length(models)) {
        label = get.label(param.num, model.num)
        mles[[label]] = models[[model.num]]$mle(params[[param.num]],
                                                print.level=2)
    }
}

# Save the results in .tex files to include in report
for (k in names(mles)) {
    save.mle.res(mles[[k]], k)
}

# Calculate local decodings and save plots as pdf
decodings = list()
for (model.num in 1:length(models)) {
    params = mles[[get.label(1, model.num)]]$mle
    model = models[[model.num]]
    discrete.decoding = model$local.decoding(params)
    cont.decoding = model$p.i[discrete.decoding]
    decodings[[model.num]] = list(discrete=discrete.decoding,
                                  continous=cont.decoding,
                                  model=model)
}
for (model.num in 1:length(models)) {
    save.decoding.plot(decodings[[model.num]], get.label(1, model.num),
                       paste("Local decoding of population", model.num))
}

# Calculate the estimated correlations between parameter estimates
vc = mles[[get.label(1, 2)]]$var.cov
for (i in 1:4) {
    for (j in (i+1):5) {
        print(c(i, j))
        print(vc[i, j]/sqrt(vc[i, i]*vc[j, j]))
    }
}
```

A.3 Plotting data

```
source('src/functions.R')
source('src/loaddata.R')

SAVEPLOTS = TRUE

for (i in 1:2) {
    filename = sprintf('dataset-%s.pdf', i)
    main = paste("Noisy observations of log-population size of population", i)
    plot.and.save(filename, 12, 5, 1.5, plot, dat[[i]], type="l", lwd=2,
                  main=main, xlab="Time", ylab="Log-population size")
}
```

A.4 Helper functions

```
plot.and.save = function(filename, width, height, cex, plotfunction, ...) {
    save.the.plot = exists('SAVEPLOTS') && SAVEPLOTS
    if (save.the.plot) {pdf(sprintf('plots/%s', filename), width, height)}
    plotfunction(..., cex.lab=cex, cex.main=cex, cex.axis=cex)
    if (save.the.plot) {dev.off()}
}

float.str = function (float) {
    return(sprintf('%.03f', float))
}

save.mle.res = function (mle.res, prefix) {
    save.latex.table.content(sprintf('%s-estimate.tex', prefix), matrix(mle.res$mle, nrow=1),
                             "\\num{%.02e}")
    save.digit(sprintf('%s-mllk.tex', prefix), mle.res$mllk)
    save.digit(sprintf('%s-aic.tex', prefix), mle.res$AIC)
    save.digit(sprintf('%s-bic.tex', prefix), mle.res$BIC)
    if (!is.null(mle.res$var.cov)) {
        save.latex.table.content(sprintf('%s-varcov.tex', prefix), mle.res$var.cov,
                                 "\\num{%.02e}")
        sds = sqrt(diag(mle.res$var.cov))
        lower.bound = mle.res$mle - 1.96*sds
        upper.bound = mle.res$mle + 1.96*sds
        estimate.table = cbind(mle.res$mle, lower.bound, upper.bound, sds)
        save.estimate.table(sprintf('%s-confidence.tex', prefix), estimate.table)
    }
}

save.digit = function (filename, digit, digit.format='%.03f') {
    sink(sprintf('results/%s', filename))
    cat(sprintf(digit.format, digit))
    sink()
}

get.estimate.table = function (mat) {
    row.output = c()
    row.labels = c("$\\theta$", "$r_0$", "$K$", "$Q$", "$R$")
    for (i in 1:dim(mat)[1]) {
        row.output[i] = paste(row.labels[i], "&", paste(sprintf("%.04f", mat[i,]), collapse=" & "))
    }
```

```
    return(paste(row.output, collapse="\\\\"))
}

save.estimate.table = function (filename, mat) {
    sink(sprintf('results/%s', filename))
    cat(get.estimate.table(mat))
    sink()
}

latex.table.content = function (mat, digit.format='%.03f') {
    row.output = c()
    for (i in 1:dim(mat)[1]) {
        row.output[i] = paste(sprintf(digit.format, mat[i,]), collapse=" & ")
    }
    return(paste(row.output, collapse="\\\\"))
}

save.latex.table.content = function (filename, mat, digit.format='%.03f') {
    sink(sprintf('results/%s', filename))
    cat(latex.table.content(mat, digit.format))
    sink()
}

save.decoding.plot = function (decoding, prefix, main) {
    plot.and.save(sprintf('%s-decoding.pdf', prefix), 12, 5, 1.5,
                  plot.decoding, decoding, main)
}

plot.decoding = function (decoding, main, ...) {
    plot(decoding$model$data, type="l", xlab="Time", col="gray",
         ylab="Log-population size", main=main, lwd=2, ...)
    lines(decoding$continous, lwd=2)
    legend("bottomright", c("Observations", "Decoding"), lty=c(1,1),
           col=c("gray", "black"), lwd=c(2.5, 2.5), inset=c(0.02, 0.05))
}
```