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 various types of decoding are performed. 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 (1) 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 1. 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 $K \approx \exp(7) \approx 1100$.

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 1 and 2 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.

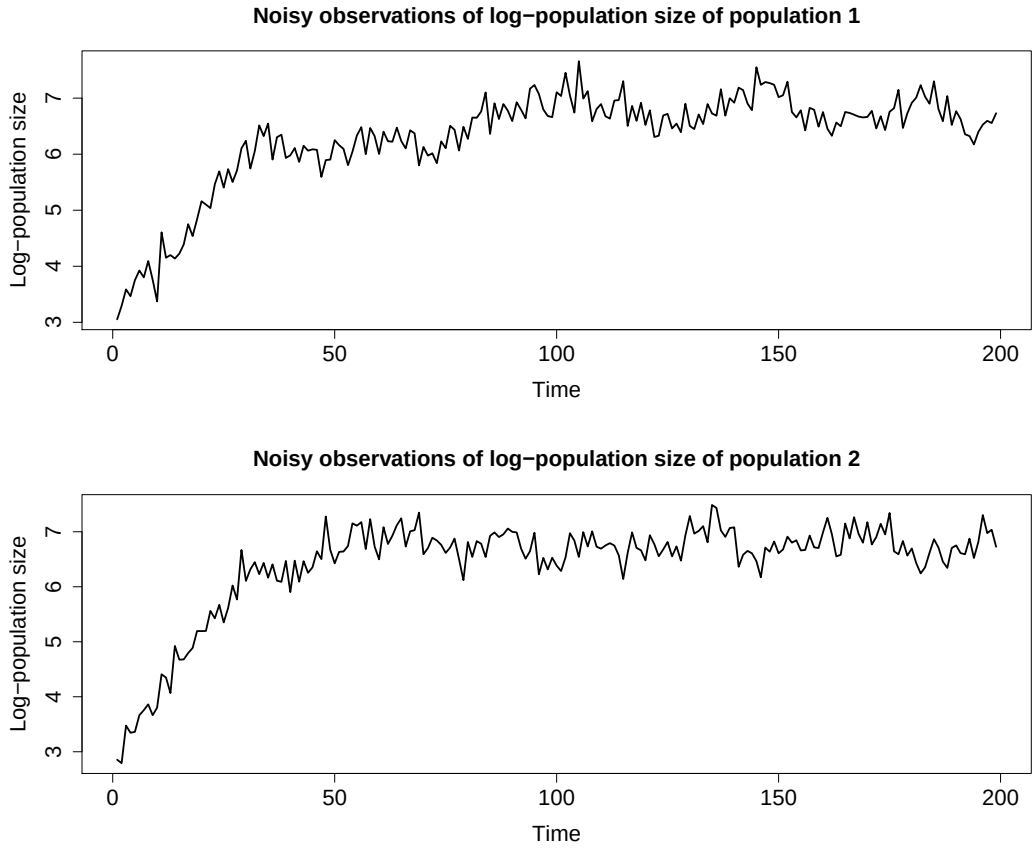

Figure 1: Plot of the two datasets

Each discrete state i is represented by the midpoint p_i of the interval (b_{i-1}, b_i) given by $p_i = 2.1 + w(i - 0.5)$. 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 (2). 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 1. 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 [1] 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 [2] 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 [1].

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 1. 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 1 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 A.1. 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 A.1 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

```
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 1

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

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

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

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(params)
        AIC = 2*(mllk+num.params)
        num.obs = sum(!is.na(x))
        BIC = 2*mllk+num.params*log(num.obs)
        var.cov = inv.hessian.from.working.hessian(res$hessian, params)
        list(mle=mle, code=res$code, mllk=mllk, AIC=AIC, BIC=BIC,
             var.cov=var.cov)
    }

    return(list(mle=HMM.mle, mllk=HMM.mllk))
}
```

References

[1] Martin Wæver Pedersen. HMM analysis of general state-space models. `http://www2.imm.dtu.dk/~mwp/02433/week7_notes.pdf`, June 2011.

[2] Walter Zucchini and Iain L. MacDonald. *Hidden Markov Models for Time Series*. Chapman & Hall/CRC, 1st edition, 2009.