B

Appendix: Parameter estimation in hidden Markov modelling of migraine

B.1 Model specifications

As mentioned in earlier chapters, general assumptions and estimation methods for hidden Markov models in continuous time have been described in Bureau *et al.* [1]. In this Appendix, they are explained for the migraine model in Chapter 3 of this thesis. The notation used is similar to that in [2].

The data set that is used as input to the hidden Markov model contains the following fields:

- a time variable $T_{i,j}$ giving the times of (irregularly spaced) headache assessments,
- the headache severity $Y_{i,j}$ observed at each assessment. Its value is an element of the vector of headache scores $\{0, 1, 2, 3\}$, the values of which denote no pain, mild pain, moderate pain and severe pain, respectively.
- Sumatriptan concentrations $Z_{i,j}$, simulated based on pharmacokinetic data from early-phase trials at the times of headache assessment. $Z_{i,j}$ is zero in patients that were administered placebo.

Subscript i is used to refer to the ith patient in the data set. Within the subject-level, subscript j denotes the jth measurement.

The hidden Markov model (HMM) of the migraine attack assumes that there exists a variable $X_{i,j}$ that is not observed. $X_{i,j}$ represents the states of the hidden Markov process underlying the observed headache scores $Y_{i,j}$. $X_{i,j}$ takes values from the vector of states $\{1, 2, 3\}$. These states can be interpreted clinically as "no relief", "relief" and "pain free" status.

The relationship between $X_{i,j}$ and $Y_{i,j}$ is given by equation B.1 which states that if the hidden state at the jth assessment is known, the observed headache score at that assessment, $Y_{i,j}$, is independent of all previous states and all previous scores.

$$P[Y_{i,j}|X_{i,1},\ldots,X_{i,j},Y_{i,1},\ldots,Y_{i,j-1}] = P[Y_{i,j}|X_{i,j}] = f(y_{i,j}|x_{i,j}). \tag{B.1}$$

Since no subject-specific parameters are defined in this model, the subscript i is omitted in the remainder of the text. The absence of subject-specific parameters is the main difference with hierarchical models using non-linear mixed effects.

The dynamics of the hidden process are governed by the Markov property (equation B.2) which states that if the previous state of the process is known, the current state is independent of all other states in the past.

$$P[X_j|X_1,\ldots,X_{j-1},T_1,\ldots,T_j] = P[X_j|X_{j-1},T_{j-1},T_j]. \tag{B.2}$$

Since the current HMM is in continuous time, the probability of being in state j also depends on the time between subsequent observations.

Instantaneous transition rates $r_{x1,x2}(t)$ define the probabilities of moving from one state to another on an infinitesimally small interval. These rates constitute a matrix $R(\theta, t)$, where θ is a vector of parameters (see equation 1 in Appendix A). The rates are dependent on time (t) as covariate Z is time-varying.

When calculating the transition probabilities $P(t)$ from the instantaneous rates, the algorithm for parameter estimation was based on the following relationship between these variables:

$$P(t) = \exp(R(\theta,t)\cdot t)$$

In the calculation of confidence intervals on mean-predicted headache responses (Appendix A), the Kolmogorov differential equations were used instead of this relation. The use of these differential equations renders calculations faster and more stable.

The probabilities $p_{Y_j|X_j}$ of observing a score Y_j given state X_j are parameterised as logarithms of odds. For example, if the current state is state 1, the probabilities are:

$$\begin{aligned}
p_{0|X_j=1} &= \frac{1}{1+exp(\beta_1)+exp(\beta_2)+exp(\beta_3)}\\
p_{1|X_j=1} &= \frac{exp(\beta_1)}{1+exp(\beta_1)+exp(\beta_2)+exp(\beta_3)}\\
p_{2|X_j=1} &= \frac{exp(\beta_2)}{1+exp(\beta_1)+exp(\beta_2)+exp(\beta_3)}\\
p_{3|X_j=1} &= \frac{exp(\beta_3)}{1+exp(\beta_1)+exp(\beta_2)+exp(\beta_3)}
\end{aligned}$$

where $\beta_{1,2,3}$ are the parameters.

The parameterisation used to specify the probability distribution of starting states π_{x_1} is similar to that of the observed score probabilities. In the migraine model, π_{x_1} is fixed to $\{1, 0, 0\}$ for states 1, 2 and 3, respectively.

B.2 The Baum-Welch algorithm

The algorithm that maximises the log-likelihood of HMMs is referred to as the Baum-Welch algorithm and is also known as the forward-backward algorithm or Expectation-Maximisation (EM) algorithm for HMMs.

The likelihood of the sequence of headache scores of patient i, given the HMM parameters and covariate values, is the product of

- the probability of starting in a certain state,
- the probabilities of passing through a certain sequence of hidden states at subsequent times,
- the probabilities of observing certain scores given the sequence of states.

As the actual sequence of states is not observed, all possible state sequences need to be evaluated in order to find the most likely one. However, using the properties in equations B.1 and B.2, this part of the likelihood can be calculated iteratively, enabling maximisation of the likelihood.

The following steps describe in more detail the estimation procedure and assumptions. The logarithm of the likelihood is taken as this simplifies calculations.

The log-likelihood of the observed sequence of scores is

$$l(\theta|y) = \Sigma_x log\pi_{x_1} + \Sigma_x\Sigma_{j>1} logP_{x_{j-1},x_j|z_j}(t_j - t_{j-1}) + \Sigma_x\Sigma_j logf(y_j|x_j),$$

where Σ_x denotes the sum over all possible states and Σ_j the sum over all assessments in a sequence. $P_{x_{j-1},x_j|z_j}(t_j - t_{j-1})$ is the transition probability from state x_{j-1} to state x_j over the interval between the two assessments, given the sumatriptan concentration.

Instead of evaluating the log-likelihood of the observations, the log-likelihood of the observations *and* the hidden states will be evaluated. This log-likelihood is called the complete log-likelihood.

$$l(\theta|x,y) = log\pi_{x_1} + \Sigma_{j>1} logP_{x_{j-1},x_j|z_j}(t_j - t_{j-1}) + \Sigma_j logf(y_j|x_j).$$

The complete log-likelihood itself cannot be evaluated. However, its expectation with respect to the hidden state sequence, given the observed score sequence and the current parameter estimates θ', can be evaluated. It has been shown that maximising this expectation also maximises the log-likelihood of the observed data [3]. The expectation of the complete log-likelihood is given by

$$\begin{aligned} l(\theta|\theta',x,y) \quad = \quad & \Sigma_x log\pi_{x_1} P(X|Y,\theta',Z) + \\ & \Sigma_{x_a,x_b} \Sigma_{j>1} logP_{x_a,x_b|z_j}(t_j - t_{j-1}) P(X_a,X_b|Y,\theta',Z) + \\ & \Sigma_x \Sigma_j logf(y_j|x_j) P(X|Y,\theta',Z). \end{aligned}$$

This result is directly obtained by applying the following probability rule:

$$E[h(X)|Y] = \Sigma_X h(X)P(X|Y)$$

The evaluation of the expectation is the E-step in the EM algorithm. This step uses a number of convenient recurrent procedures that are based on the properties of HMMs (equations B.1 and B.2)[3]. These procedures replace the parts of the expectation for which evaluation is difficult: $P(X|Y,\theta',Z)$ and $P(X_a,X_b|Y,\theta',Z)$.

The M-step maximises the expected value by updating the current parameter values. This is done using the gradient descent method in the NPSOL library for nonlinear optimisation [4].

The EM algorithm alternates between the Expectation and the Maximisation step until a maximum value of the expected log-likelihood has been found.

B.3 Model implementation: user-written routines

Modelling the anti-migraine response to sumatriptan required the implementation of a more complicated transition-rate model. When changing any part of the existing model structure of the HMM, related structures need to be changed as well: Equations for the gradients (first-order derivatives) of the model parameters need to be adapted in order to ensure minimisation of the objective function. The Hessians (second-order derivatives with respect to the parameters) need to be redefined if estimates of standard errors are required.

User-written models for transition rates and their derivatives were created and implemented by adapting available C code and corresponding functions in the S-Plus wrapper.

In addition, a S-Plus routine was written to derive confidence intervals for mean-predicted headache responses.

This section lists three code files that were edited for the implementation used in Chapters 3, 4 and 5 of this thesis.

- `model.c`, the file that contains the structures of the observed score probabilities, the instantaneous transition rates and the initial state probabilities.
- `model.s`, the corresponding S wrapper.
- `probtit.s`, the iterative S-Plus routine for constructing confidence intervals around predicted headache response profiles.

Excerpt from `model.c`, containing the edited transition model specification and associated first and second derivatives. A description of the full code can be found on

`http://www.crulrg.ulaval.ca/pages_perso_chercheurs/bureau_a/logiciel.html`

```
/* Function Body */
   if (*mode > 0) {
    if (*mode > 2)
    for (i = 1; i <= *ns; ++i)
        for (j = 1; j <= *ns; ++j)
        for (ijk = 1; ijk <= *nh; ++ijk) {
            grad[i + (j + ijk * (*ns)) * (*ns)] = zero;
    /*                for (ijk2 = 1; ijk2 <= ijk; ijk2++)
    27/8/2002 Line modified by H.Maas: ijk2 <= *nh , to allow
    initialisation of all cells in 'hes'. However, change does
    not influence output */
        for (ijk2 = 1; ijk2 <= *nh; ijk2++)
            hessian[i + (j + (ijk + ijk2 * (*nh)) * (*ns)) *
                (*ns)] = zero;
        }
      else
        for (i = 1; i <= *ns; ++i)
        for (j = 1; j <= *ns; ++j)
        for (ijk = 1; ijk <= *nh; ++ijk)
            grad[i + (j + ijk * (*ns)) * (*ns)] = zero;
    }
/* H.Maas 27/8/2002; ijk initialisation for linear covariate
relation at time zero: */
    ijk = (*ns - 1) * (*nph + 1);
    for (i = 1; i <= *ns; ++i) {
        /* NLIN start */
    /* NLIN end */
    sum = zero;
    ifst = ijk;
    for (j = 1; j <= *ns; ++j) {
        if (j == i) continue ;
    /* Linear model left-outs signified by LIN
    LIN ++ijk;
        work = hpar[ijk];  */
        work = zero;
        for (k = 1; k <= *nph; ++k) {
        ++ijk;
    /* LIN   work += hpar[ijk] * hcov[k]; */
              wosm[k-1] = hcov[k] / (exp(hpar[ijk]) + hcov[k]);
         ++ijk;
         wosm[k-1] *= hpar[ijk];
        }
    /* Nonlinear model add-ins surrounded by: NLIN start */

        for( k=1; k<=*nph; ++k)
        {
        work += wosm[k-1];
        }

        ++ijk;
        work += hpar[ijk];
    /* NLIN end */
        a[i + j * (*ns)] = exp(work);
        sum += a[i + j * (*ns)];
    }
    a[i + i * (*ns)] = -sum;
    if (*mode > 0) {
        ijk = ifst;
        for (j = 1; j <= *ns; ++j) {
        if (i == j) continue ;
        /* LIN
        ++ijk;
        grad[i + (i + ijk * (*ns)) * (*ns)] = -a[i + j * (*ns)];
        grad[i + (j + ijk * (*ns)) * (*ns)] = a[i + j * (*ns)];
        */
        for (k = 1; k <= *nph; ++k) {
        ++ijk;
        /* NLIN start (linear model overwritten)  */
        klm = ijk + 1;
        grad[i + (i + ijk * (*ns)) * (*ns)] = (hcov[k] * hpar[klm]
                 * exp(hpar[ijk]) *
         a[i + j * (*ns)]) / ((exp(hpar[ijk]) + hcov[k]) *
                 (exp(hpar[ijk]) + hcov[k]));
         grad[i + (j + ijk * (*ns)) * (*ns)] = -(hcov[k] * hpar[klm]
                 * exp(hpar[ijk]) *
         a[i + j * (*ns)]) / ((exp(hpar[ijk])+hcov[k])*(exp(hpar[ijk])
                 + hcov[k]));
         ++ijk;
         kji = ijk - 1;
         grad[i + (i + ijk * (*ns)) * (*ns)] = -(hcov[k] *
         a[i + j * (*ns)]) / (exp(hpar[kji]) + hcov[k]);
         grad[i + (j + ijk * (*ns)) * (*ns)] = (hcov[k] *
         a[i + j * (*ns)]) / (exp(hpar[kji]) + hcov[k]);
        }

        ++ijk;
        grad[i + (i + ijk * (*ns)) * (*ns)] = -a[i + j * (*ns)];
        grad[i + (j + ijk * (*ns)) * (*ns)] = a[i + j * (*ns)];
            /* NLIN end */
        }
    }
        if (*mode > 2) {
            ij2 = ifst;
            for (j = 1; j <= *ns; ++j) {
          if (i != j)
         { /* Start the parameter index at 1st parameter of
                      transition from i to j
                 LIN
             isd = ijk2 = ij2 = j < i ? (j + i * (*ns - 1)) *
                     (*nph + 1):
             (j - 1 + i * (*ns - 1))
                      * (*nph + 1);
         hessian[i + (i + (ij2 + ijk2 *(*nh)) * (*ns)) * (*ns)]
                           = -a[i + j * (*ns)];
         hessian[i + (j + (ij2 + ijk2 *(*nh)) * (*ns)) * (*ns)]
                           = a[i + j * (*ns)];
NLIN start ; next line initiates parameter index in the
case of nonlinear transitions BUT linear initial distributions */
            isd = ijk2 = ij2 = j < i ? (j + i * (*ns - 1)) *
                 (2*(*nph) + 1) - 4*(*nph):(j - 1 + i * (*ns - 1))
                  * (2*(*nph) + 1) - 4*(*nph);
            for (l = 1; l <= *nph; ++l) {
              kml = ij2 + 1;
        for (k = 1; k <= *nph; ++k) {
                  isq = l == k ? 1 : 0;
                  klm = ijk2 + 1;
  hessian[i + (i + (ij2 + ijk2 *(*nh)) * (*ns)) * (*ns)]
 = -hcov[l] * hpar[kml] * exp(hpar[ij2]) * (hpar[klm] * hcov[k] *
   exp(hpar[ijk2]) + 2 * isq * exp(hpar[ij2]) * (exp(hpar[ij2]) +
   hcov[l]) -  isq * (exp(hpar[ij2]) + hcov[l]) * (exp(hpar[ij2])
   + hcov[l])) * a[i + j * (*ns)] / ((exp(hpar[ij2]) + hcov[l]) *
   (exp(hpar[ij2]) + hcov[l]) * (exp(hpar[ijk2]) + hcov[k]) *
   (exp(hpar[ijk2]) + hcov[k]));
   hessian[i + (j + (ij2 + ijk2 *(*nh)) * (*ns)) * (*ns)]
   = hcov[l] * hpar[kml] * exp(hpar[ij2]) * (hpar[klm] * hcov[k] *
     exp(hpar[ijk2]) +
2 * isq * exp(hpar[ij2]) * (exp(hpar[ij2]) + hcov[l]) -
 isq * (exp(hpar[ij2]) + hcov[l]) * (exp(hpar[ij2]) + hcov[l])) *
 a[i + j * (*ns)] /
  ((exp(hpar[ij2]) + hcov[l]) * (exp(hpar[ij2]) + hcov[l]) *
   (exp(hpar[ijk2]) + hcov[k]) * (exp(hpar[ijk2]) + hcov[k]));
                  ++ijk2;
kji = ijk2 - 1;
 hessian[i + (i + (ij2 + ijk2 *(*nh)) * (*ns)) * (*ns)]
= hcov[l] * exp(hpar[ij2]) * (hcov[k] * hpar[kml] + isq *
  exp(hpar[ij2]) +
 isq * hcov[l]) * a[i + j * (*ns)] / ((exp(hpar[ij2]) + hcov[l]) *
 (exp(hpar[ij2]) + hcov[l]) * (exp(hpar[kji]) + hcov[k]));
 hessian[i + (j + (ij2 + ijk2 *(*nh)) * (*ns)) * (*ns)]
 = -hcov[l] * exp(hpar[ij2]) * (hcov[k] * hpar[kml] + isq *
   exp(hpar[ij2]) +
   isq * hcov[l]) * a[i + j * (*ns)] / ((exp(hpar[ij2]) + hcov[l]) *
   (exp(hpar[ij2]) + hcov[l]) * (exp(hpar[kji]) + hcov[k]));
         ++ijk2;
               }
           hessian[i + (i + (ij2 + ijk2 *(*nh)) * (*ns))
                   * (*ns)]= hpar[kml] * hcov[l] *
                   exp(hpar[ij2]) * a[i + j * (*ns)] /
                  ((exp(hpar[ij2]) + hcov[l]) * (exp(hpar[ij2])
                  + hcov[l]));
           hessian[i + (j + (ij2 + ijk2 *(*nh)) * (*ns)) * (*ns)]
                   = -hpar[kml] * hcov[l] * exp(hpar[ij2]) *
                   a[i + j * (*ns)] / ((exp(hpar[ij2]) + hcov[l])
                  * (exp(hpar[ij2]) + hcov[l]));

           ijk2 = isd;
           ++ij2;
           kkl = ij2 - 1;

           for (k = 1; k <= *nph; ++k) {
           isq = l == k ? 1 : 0;
           klm = ijk2 + 1;
           hessian[i + (i + (ij2 + ijk2 *(*nh)) * (*ns)) * (*ns)]
                  = hcov[l] * exp(hpar[ijk2]) * (hcov[k] *
                    hpar[klm] + isq * exp(hpar[kkl]) + isq * hcov[l])
                    * a[i + j * (*ns)] / ((exp(hpar[kkl])+ hcov[l]) *
                   (exp(hpar[ijk2]) + hcov[k]) * (exp(hpar[ijk2])
                    + hcov[k]));
           hessian[i + (j + (ij2 + ijk2 *(*nh)) * (*ns)) * (*ns)]
                  = -hcov[l] * exp(hpar[ijk2]) * (hcov[k] * hpar[klm] +
                    isq * exp(hpar[kkl]) +
                    isq * hcov[l]) * a[i + j * (*ns)] / ((exp(hpar[kkl])
                    + hcov[l]) * (exp(hpar[ijk2]) + hcov[k]) *
                    (exp(hpar[ijk2]) + hcov[k]));
           ++ijk2;
           kji = ijk2 - 1;
           hessian[i + (i + (ij2 + ijk2 *(*nh)) * (*ns)) * (*ns)]
                  = - hcov[l] * hcov[k] * a[i + j * (*ns)] /
                    ((exp(hpar[kkl]) + hcov[l]) * (exp(hpar[kji]) +
                    hcov[k]));
           hessian[i + (j + (ij2 + ijk2 *(*nh)) * (*ns)) * (*ns)]
                  = hcov[l] * hcov[k] * a[i + j * (*ns)] /
                    ((exp(hpar[kkl]) + hcov[l]) * (exp(hpar[kji]) +
                    hcov[k]));
           ++ijk2;
           }
           hessian[i + (i + (ij2 + ijk2 *(*nh)) * (*ns)) * (*ns)]
                  = - hcov[l] * a[i + j * (*ns)] /
                    (exp(hpar[kkl]) + hcov[l]);
           hessian[i + (j + (ij2 + ijk2 *(*nh)) * (*ns)) * (*ns)]
                  = hcov[l] * a[i + j * (*ns)] /
                    (exp(hpar[kkl]) + hcov[l]);
         ++ij2;
         ijk2 = isd;
           }

          for (k = 1; k <= *nph; ++k) {
              klm = ijk2 + 1;
          hessian[i + (i + (ij2 + ijk2 *(*nh)) * (*ns)) * (*ns)]
                  = hcov[k] * hpar[klm] * exp(hpar[ijk2]) *
```

```
                     a[i + j * (*ns)] /
                     ((exp(hpar[ijk2]) + hcov[k]) * (exp(hpar[ijk2])
                     + hcov[k]));
           hessian[i + (j + (ij2 + ijk2 *(*nh)) * (*ns)) * (*ns)]
                   = - hcov[k] * hpar[klm] * exp(hpar[ijk2]) *
                     a[i + j * (*ns)] /((exp(hpar[ijk2]) + hcov[k]) *
                    (exp(hpar[ijk2]) + hcov[k]));
           ++ijk2;
           kji = ijk2 - 1;
           hessian[i + (i + (ij2 + ijk2 *(*nh)) * (*ns)) * (*ns)]
                   = - hcov[k] * a[i + j * (*ns)] /
                     (exp(hpar[kji]) + hcov[k]);
           hessian[i + (j + (ij2 + ijk2 *(*nh)) * (*ns)) * (*ns)]
                   = hcov[k] * a[i + j * (*ns)] /
                    (exp(hpar[kji]) + hcov[k]);
           ++ijk2;
           }
           hessian[i + (i + (ij2 + ijk2 *(*nh)) * (*ns)) * (*ns)]
                   = - a[i + j * (*ns)];
           hessian[i + (j + (ij2 + ijk2 *(*nh)) * (*ns)) * (*ns)]
                   = a[i + j * (*ns)];

        /* NLIN end */
          } /* closing "if (i != j)" loop */
            } /* closing "j" loop ("to state" loop) */
        } /* closing mode2 */
    }  /* closing "i"loop ("from state" loop) */
    return 0;
} /* intsty_ */
```

model.s

```
intsty<-function(cmode=0, hpar, nst, nph, hcov)
  {
  nh<-length(hpar)
  a<-double(nst*nst)
  grad<-double(nst*nst*nh)

 # hid.par<-matrix(hpar[((nst-1)*(nph+1)+1):nh],nph+1,nst)
 #linear covariates, 2 states only
  hid.par<-matrix(hpar[((nst-1)*(nph+1)+1):nh],nrow=2*nph+1)
  #emax covariates, all numbers/states

  if(nph == 1) a[-seq(1,nst*nst,nst+1)]<-exp(hid.par[seq(2,2*nph,2),]*
  hcov/(exp(hid.par[seq(1,2*nph-1,2),])+hcov)+hid.par[nph*2+1,])
  #off-diagonal elements;ln(EC50) model!

  else if(nph > 1) a[-seq(1,nst*nst,nst+1)]<-
  exp(apply(hid.par[seq(2,2*nph,2),]*hcov/(exp(hid.par[seq(1,2*nph-1,2),])
  +hcov),2,sum)+
  hid.par[nph*2+1,]) #off-diagonal elements;ln(EC50) model!

  else a[-seq(1,nst*nst,nst+1)]<-exp(hid.par)
  # Column order of "a" is (1,1)(1,2)(1,3)(2,1)(2,2)(2,3)(3,1)(3,2)(3,3)

  temp<-matrix(a,nst,nst)

  a[seq(1,nst*nst,nst+1)]<--apply(temp,2,sum)


  if (cmode > 0)
     {
     ij2<-(nst-1)*(nph+1)
     hidpar.col<-0
     for (i in 1:nst)
       {
       for (j in 1:nst) {

         if (j!=i) {
        hidpar.col<-hidpar.col + 1

# NEW 21/08/2005 if functions for hmm functions with 0 hidden
covariates # NEW 22/08/2005 switched j and i subscripts in grad
code

        if (nph >0 ) {

            for (k in 1:nph) {

     grad[i + (i + ij2*nst-1)*nst]<-(hcov[k]*hid.par[2*k,hidpar.col]*
     exp(hid.par[2*k-1,hidpar.col])*a[j + (i-1)*nst]) /
     ((exp(hid.par[2*k-1,hidpar.col])+
     hcov[k])**2) #logec50 diag
     grad[i + (j + ij2*nst-1)*nst]<--(hcov[k]*hid.par[2*k,hidpar.col]*
     exp(hid.par[2*k-1,hidpar.col])*a[j + (i-1)*nst]) /
     ((exp(hid.par[2*k-1,hidpar.col])+
     hcov[k])**2) #logec50 off-diag

        ij2<-ij2 + 1

            grad[i + (i + ij2*nst-1)*nst]<--(hcov[k]*a[j + (i-1)*nst]) /
            (exp(hid.par[2*k-1,hidpar.col])+hcov[k])  #emax diag
            grad[i + (j + ij2*nst-1)*nst]<-(hcov[k]*a[j + (i-1)*nst]) /
            (exp(hid.par[2*k-1,hidpar.col])+hcov[k])  #emax off-diag

        ij2<-ij2 + 1
                          }
              }

        grad[i + (i + ij2*nst-1)*nst]<--a[j + (i-1)*nst] #base diag
        grad[i + (j + ij2*nst-1)*nst]<-a[j + (i-1)*nst] #base off-diag

        ij2<-ij2 + 1

             } #j!=i
           } #j
            } #i
         } #grad

 # cat ("cov = ", hcov, "\n")
 # cat ("a = ",a, " grad<-a = ",
 # grad, "\n")
  list(a=a, grad=grad)
  } #intsty

first<-function(cmode=0, hpar, nst, nph, hcov) # Linear covariate
structure assumed
  {
  nh<-length(hpar)
  delta<-double(nst)
  grad<-double(nst*nh)

  first.par<-matrix(hpar[1:((nst-1)*(nph+1))],nrow=nph+1)
  delta[1]<-1.0
   if (nph > 0) delta[-1]<-exp(apply(first.par*c(1,hcov),2,sum))
   # for all numbers of states,
   # linear covariates. If nph>1, 'hcov' is 'combinh'in simulation mode

  else delta[-1]<-exp(first.par)
  delta<-delta/sum(delta)

# cmode=0
  if (cmode > 0) {
     for (i in 1:nst) {
     for (j in 2:nst) {
  if (j==i) dterm<-delta[i] * (1 - delta[i])
    else dterm<--delta[i] * delta[j]
  if (nph > 0) grad[seq(i+(j-2)*nst,i+(j-2+nph)*nst,nst)]<-dterm*c(1,hcov)
    else grad[i+(j-2)*nst]<-dterm
         }
```

```
          }
        }
#        cat ("cov = ", hcov, "\n")
#        cat ("delta = ",delta,"\n")
#        cat ("grad = ", grad, "\n")
        list (delta=delta, grad)  }}
```

probtit.s

```
meanscores <- function(x,timevec,cmode,hcov,scorevec) {
 # Required arguments
 #   x           - An S-plus hmm object
 #   timevec     - Vector containing observational time points
 #                 (length = ntime)
 #   cmode       - Should prediction interval be estimated
 #                (yes=1,no=0) ?
 #   scorevec    - Which score or composite of scores should
 #                be predicted?
 #                e.g. c(0,1) for having either score 0 or 1
 #
 # Optional arguments
 #   hcov        - matrix (not vector!) of
 #                 hidden process covariate values
 #                (ncol=nph,nrow=ntime)
 # preparations

  nh <- as.integer(length(x$hidden.par))
  no <- as.integer(length(x$obs.par))
  nx <- nh + no

    if (missing(hcov))
    {
        if (x$nph>0)
        hcov <- matrix(0,ncol=x$nph,nrow=length(timevec))
    else
        hcov <- matrix(0,ncol=1,nrow=length(timevec))
    }

    if (timevec[1] != 0.0)
        stop("First element of timevec should be zero")

    probab <- matrix(0,nrow=(x$ns*length(timevec)),ncol=x$ns+1)
    # transition probabilities conditional on starting states
    probab[,1] <- rep(timevec,1,each=x$ns)
    probab[1:x$ns,-1] <- diag(rep(1,x$ns))
    probability <- probab
    # score probabilities taking into account initial conditions

    lx <- x$ns*length(timevec)
    ltheta <- length((((x$ns-1)*(x$nph+1)+1):nh)[eval(x$call$bl)
    [((x$ns-1)*(x$nph+1)+1):nh] <
        eval(x$call$bu)[((x$ns-1)*(x$nph+1)+1):nh]])
    probabderiv <- array(0,dim=c(lx,x$ns+1,ltheta))
    # deriv of transition probabilities
    # conditional on starting state
    probabderiv[,1,] <- rep(timevec,1,each=x$ns)

    variance <- rep(0,length(timevec))
    variance[1] <- 0

    # extracting relevant elements from var-covar matrix
    lvarf <- length((1:((x$ns-1)*(x$nph+1))))
```

```
[eval(x$call$bl)[1:((x$ns-1)*(x$nph+1))]
< eval(x$call$bu)[1:((x$ns-1)*(x$nph+1))]])

if (length((((x$ns-1)*(x$nph+1)):nh)[eval(x$call$bl)
[((x$ns-1)*(x$nph+1)):nh] < eval(x$call$bu)
        [((x$ns-1)*(x$nph+1)):nh]]) > 0)
varh <- (lvarf+1):(lvarf + length((((x$ns-1)*(x$nph+1))
    :nh)[eval(x$call$bl)
[((x$ns-1)*(x$nph+1)):nh] < eval(x$call$bu)[((x$ns-1)*
    (x$nph+1)):nh]]))
else
stop("No standard errors available for intensity matrix
    elements")

varcov <- x$var[c(varh),c(varh)]

# preparing output matrix
outputmatrix <- piout(x,print=F)
outputmatrix <- outputmatrix[scorevec+1,]

if (length(scorevec) > 1)
    outputmatrix <- as.matrix(apply(outputmatrix,2,sum))

# iterating over the times in the time vector
for( i in 2:length(timevec) )
{
a <- as.double(intsty(cmode,x$hidden.par,x$ns,
     x$nph,hcov[i,])$a)
grad <- as.double(intsty(cmode,x$hidden.par,x$ns,
     x$nph,hcov[i,])$grad)

transmatrix <- matrix(a,x$ns,x$ns,byrow=T)
gradarray <- array(grad,dim=c(x$ns,x$ns,nh))


# extracting relevant elements from gradient matrix
#(partial derivatives of transition RATES)
    gradcolselect <- (((x$ns-1)*(x$nph+1)+1):nh)
    [eval(x$call$bl)[((x$ns-1)*(x$nph+1)+1):nh]
    < eval(x$call$bu)[((x$ns-1)*(x$nph+1)+1):nh]]

grada <- gradarray[,,gradcolselect]

# calculating mean of transition probabilities and their
# partial derivatives
    probab[((i-1)*x$ns+1):(x$ns*i),-1] <-
    probab[((i-2)*x$ns+1):((i-1)*x$ns),-1] +
     (probab[((i-2)*x$ns+1):
     ((i-1)*x$ns),-1] %*% transmatrix)*
     (timevec[i] - timevec[i-1])

pgrad <- array(0,dim=c(x$ns,x$ns+1,ltheta))
derivr <- array(0,dim=c(x$ns,x$ns+1,ltheta))

# iterating over the theta dimension in arrays
```

```
        for( j in 1:ltheta)
        {
            pgrad[1:x$ns,-1,j] <-
                probab[((i-2)*x$ns+1):((i-1)*x$ns),-1]
            %*% grada[,,j]
            # pgrad = probababilities * gradient
            derivr[1:x$ns,-1,j] <-
                probabderiv[((i-2)*x$ns+1):((i-1)*x$ns),-1,j]
            %*% transmatrix
            # derivr = derivative of
            # probabilities * intensity matrix
        probabderiv[((i-1)*x$ns+1):(x$ns*i),-1,j]
            <- probabderiv
        [((i-2)*x$ns+1):((i-1)*x$ns),-1,j] +
         pgrad[1:x$ns,-1,j]*(timevec[i] - timevec[i-1]) +
         derivr[1:x$ns,-1,j]*(timevec[i] - timevec[i-1])

        }

    # calculating mean score probabilities (no OCOV handling yet)
        probability[(i-1)*x$ns+1,-1] <-
            probab[((i-1)*x$ns+1):(i*x$ns),-1]
        %*% outputmatrix
        probability[(i-1)*x$ns+1,-1] <-
            x$initd %*% probability[(i-1)*x$ns+1,-1]

    # calculating partial derivatives of score probabilities
        probabderivative <- rep(0,ltheta)
        for( j in 1:ltheta)
        {
            probabderivative[j] <- x$initd %*%
                       probabderiv[((i-1)*x$ns+1):(i*x$ns),-1,j]
            %*% outputmatrix
        }

    # calculating variance of score probabilities
        variance[i] <-
            probabderivative %*% varcov %*% probabderivative

    } # time-loop finished


    # completing mean score probabilities for time zero
    probability[1,-1] <- probab[1:x$ns,-1] %*% outputmatrix
    probability[1,-1] <- x$initd %*% probability[1,-1]
    probability <-
     as.data.frame(probability[seq(1,length(timevec)*x$ns,x$ns),])

    # logit transformation on variances and score probabilities
    variance <- variance*(1/(probability[,2]*(1-probability[,2])))**2
    # partial derivative of score probability
    # is 1/p*(1-p), then apply delta method
    probability[,-1] <- log(probability[,-1]/(1-probability[,-1]))

    # calculating boundaries of score probabilities
    probability[,3] <- probability[,2] - 1.96 * sqrt(variance)
    probability[,4] <- probability[,2] + 1.96 * sqrt(variance)

    #if ( probability[,3] > 1 )
    #probability[,3] <- 1

    #if ( probability[,4] > 1 )
    #probability[,4] <- 1

    # back-transformation of probabilities
    probability[,-1] <- exp(probability[,-1])/
                        (1+exp(probability[,-1]))

    probability <- probability[,1:4]

    names(probability) <- c("time","mean prob","lower prob","upper prob")

    return(probability)
} }
```

References

[1] A. Bureau, J.P. Hughes, and S.C. Shiboski. An S-Plus implementation of Hidden Markov Models in Continuous Time. 2000, *J.Comp.Graph.Stat.*, 9, 621–632.

[2] A. Bureau, S. Shiboski, and J.P. Hughes. Applications of continuous time hidden Markov models to the study of misclassified disease outcomes. 2003, *Stat. Med.*, 22, 441–462.

[3] L.E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. 1970, *Ann.Math.Stat.*, 41, 164–171.

[4] P.E. Gill, W. Murray, M.A. Saunders, and M.H. Wright. User's Guide for NPSOL: a Fortran package for nonlinear programming. Report SOL 86–2, Department of Operations Research, Stanford University, 1998.