SEMI-PARAMETRIC GENERALIZED LINEAR MODEL FOR CORRELATED BINARY DATA

ANIKO SZABO

Note: this setup assumes that parts defined in `SPregress.w` are available.

1. Introduction

We extend the Rathouz-Gao semi-parametric generalized linear model to clustered binary outcomes with varying cluster sizes. Treating the number of events Y_i from a cluster scaled by corresponding cluster size n_i as the response variable, we use a parametric regression model for the marginal response probability Y/n_i and assume that for the maximum cluster size N the probability density distribution of Y_i is an exponentially tilted version of the reference density distribution.

The conditional mean model is as follows:

$$E(\frac{Y_i}{n_i}|\boldsymbol{Z}_i;\boldsymbol{\beta}) = \mu(\boldsymbol{Z}_i,\boldsymbol{\beta}) \equiv \mu_i = h^{-1}(\boldsymbol{Z}_i^T\boldsymbol{\beta}) \tag{1}$$

where $\boldsymbol{Z}_i$ is a matrix of cluster-level covariates, the function $h(\cdot)$ is a known strictly increasing link function.

The assumed probability density function for a cluster of maximal size N is:

$$q_{y,N}(\boldsymbol{Z},\boldsymbol{\beta}) \propto q^{(0)}_{y,N} \times \exp[-\omega(\boldsymbol{Z},\boldsymbol{\beta}) \times y] \tag{2}$$

where $q^{(0)}_{y,N}$ is a non-parametric reference distribution, and $\omega(\boldsymbol{Z},\boldsymbol{\beta})$ is the tilting parameter chosen so that the expectation of Y/N equals to the value defined by the mean model (1), i.e. $\sum_{y=0}^{N}\frac{y}{N}q_{y,N}(\boldsymbol{Z},\boldsymbol{\beta}) = \mu = h^{-1}(\boldsymbol{Z}^T\boldsymbol{\beta})$.

For the purpose of identifiability, the mean of marginal probability $\{q^{(0)}_{y,N}\}$ is set at an arbitrary fixed value, which can be the mean of the reference group, or the overall marginal mean of the data. We extend the conditional mean and probability density model for cluster sizes smaller than the maximal size N by assuming marginal compatibility.

2. Main function

"../R/SPGLM.R" 1≡

```
        #' Fit semi-parametric GLM
        #'
        #'@rdname spglm
        #'@param formula a one-sided formula of the form \code{cbind(r, s) ~ predictors} where \code{r} and \code
        #'@param data  an optional matrix or data frame containing the variables in the formula \code{formula}. B
        #'@param subset  an optional vector specifying a subset of observations to be used.
        #'@param weight  an optional vector specifying observation weights.
        #'@param link      a link function for the mean.
        #'@param mu0       an optional numeric value constraining the mean of the baseline distribution
        #'@param control a list with parameters controlling the algorithm.
        #'@return an object of class \code{spglm} with the fitted model.
        #'@export
        #' @importFrom stats terms model.matrix

        spglm <- function(formula, data, subset, weights, offset, link="logit", mu0=NULL,
                          control=list(eps=0.001, maxit=100), ...){
```

Date: June 30, 2021.

```
        ⟨ Create model matrix from formula and data 2 ⟩

        data_object <- list(model_matrix=mm, resp=Y, n=rowSums(Y), weights=weights, offset=offset, maxN=N)

        ⟨ Fit model 6 ⟩

        mt <- attr(mf, "terms")
        names(betas) <- colnames(mm)
        names(referencef0) <- 0:N
        res <- list(coefficients = betas, SE = SEbeta, f0=referencef0, SEf0 = SEf0, mu0=mu0, niter = iter,
                    loglik=llik, link = link, call = mc, terms = mt,
                    xlevels = .getXlevels(mt, mf),
                    data_object=data_object)
        class(res) <- "spglm"
        res

    }
    ◇
```

File defined by 1, 3, 4, 5.

⟨ *Create model matrix from formula and data* 2 ⟩ ≡

```
       if (missing(formula) || (length(formula) != 3L))
            stop("'formula' missing or incorrect")
       if (missing(data))
            data <- environment(formula)
        mc <- match.call(expand.dots = FALSE)
        m <- match(c("formula", "data", "subset", "weights", "offset"), names(mc), 0L)
        m <- mc[c(1L,m)]
        if (is.matrix(eval(m$data, parent.frame())))
            m$data <- as.data.frame(data)
        m[[1L]] <- quote(stats::model.frame)
        mf <- eval(m, parent.frame())

        # extract and check response variable
        Y <- model.response(mf)
        if (ncol(Y) != 2) stop("The response variable should be a 2-column matrix")

        # create model matrix
        mm <- model.matrix(formula, data=mf)

        if (is.character(link)) {
          link <- stats::make.link(link)
        }
        else if (!is.list(link) || !(all(c("linkfun", "linkinv",
                                        "mu.eta") %in% names(link)))) {
          stop(paste0("link should be a character string or a list containing ",
                      "functions named linkfun, linkinv, and mu.eta"))
        }

        # extract offset
        offset <- as.vector(model.offset(mf))
        if (is.null(offset))
          offset <- rep(0, nrow(mm))

        # extract weights
        weights <- as.vector(model.weights(mf))
        if (!is.null(weights) && !is.numeric(weights))
            stop("'weights' must be a numeric vector")
        if (!is.null(weights) && any(weights < 0))
            stop("negative weights not allowed")
        if (is.null(weights))
            weights <- rep(1, nrow(mm))

       # define dimensions
        N <- max(rowSums(Y))
        nobs <- nrow(mm)
        p <- ncol(mm)
```

◇
Fragment referenced in 1.

2.1. **Methods for the spglm class.** First, define a printing method which does not show the saved data and model matrix

"../R/SPGLM.R" 3≡

```
#' @rdname spglm
#' @export
print.spglm <- function(x, digits = max(3L, getOption("digits") - 3L),...){
  # based on print.lm, summary.lm, and print.summary.lm
  cat("\nA semi-parametric generalized linear regression model fit\n")
  cat("\nCall:\n", paste(deparse(x$call), sep = "\n", collapse = "\n"),
        "\n\n", sep = "")
  if (length(coef(x))) {
    beta <- x$coefficients
    SEbeta <- x$SE
    z <- beta / SEbeta
    pval <- 2 * stats::pnorm(abs(z), lower.tail=FALSE)
    coefmat <-  cbind(Estimate = beta, `Std. Error` = SEbeta,
                      `z value` = z, `Pr(>|z|)` = pval)
    cat("Coefficients:\n")
    printCoefmat(coefmat, digits = digits, na.print = "NA", ...)
    }
    else cat("No coefficients\n")

    if (length(x$f0)){
        cat("\n Baseline probabilities (q0):\n")
        q0 <- x$f0
         print.default(format(q0, digits = digits),
            print.gap = 2, quote = FALSE)
    }
    else cat("No baseline joint probabilities\n")

   cat("\n Log-likelihood: ", format(x$loglik, digits=digits), "\n")
   invisible(x)
}
```

◇
File defined by 1, 3, 4, 5.

The prediction method will predict for a variety of scenarios:

- Input data set:
  - the data used in the fitting;
  - new data.

- Results:
  - $\mu(z, \beta)$: the mean event probability, given z;
  - $p_{r,n}(z)$: the probability of observing the given r responses with cluster size n, given z;
  - $\{p_{\cdot,n}(z)\}$: the entire vector of response probabilities for cluster size n, given z (will be a list due to varying lengths);
  - $\beta' z$: the linear predictor value z;
  - $\omega(\beta' z)$: the tilting parameter at predictor value z.

"../R/SPGLM.R" 4≡

```
#' Predict methods for SPGLM fits
#' Obtains predictions from a fitted semi-parametric generalized linear model. Note that \code{offset}
#' and \code{weight} terms are not implemented for predicting from new data.
#' @param object fitted model of class \code{spglm}
#' @param newdata optionally, a data frame in which to look for covariate values for prediction.
#'  If NULL, the original data set is used
#' @param type the type of prediction requested. The default is "mean", the mean event probability;
#' "prob" requests the probability of the observation (given number of responses with given cluster size)
#' "tilt" returns the tilting parameter which achieves the modeled mean from the baseline distribution;
#' and "lp" requests the linear predictor.
#' @param newn if \code{newdata} is provided and \code{type="prob"}, an integer or integer vector specify
#' @param newevents if \code{newdata} is provided and \code{type="prob"}, an integer or integer vector sp
#'  number of events for the predictions
#' @export

predict.spglm <- function(object, newdata=NULL,
                              type=c("mean", "prob", "tilt", "lp"),
                              newn=NULL, newevents=NULL, ...){
  type <- match.arg(type)
  tt <- terms(object)
  if (!missing(newdata)){
    Terms <- delete.response(tt)
    m <- model.frame(Terms, newdata, xlev = object$xlevels)
    if (!is.null(cl <- attr(Terms, "dataClasses")))
            .checkMFClasses(cl, m)
    mm <- model.matrix(Terms, m)
    data_object <- list(model_matrix=mm, offset = rep(0, nrow(mm)), weights=rep(1, nrow(mm)),
                        maxN = object$data_object$maxN)
    if (!missing(newn)){
       if (!(length(newn) == 1L || length(newn) == nrow(newdata)))
          stop("'newn' should have length 1 or equal to the number of rows of 'newdata'")
        data_object$n <- rep(newn, length=nrow(newdata))
        if (!all(data_object$n %in% 0:data_object$maxN))
          stop("Values in 'newn' should be integers between 0 and the maximum cluster size in the origina
       }
    if (!missing(newevents)){
       if (!(length(newevents) == 1L || length(newevents) == nrow(newdata)))
          stop("'newevents' should have length 1 or equal to the number of rows of 'newdata'")
        data_object$resp <- cbind(rep(newevents, length=nrow(newdata)))
       }
  } else {
    data_object <- object$data_object
  }

  if (type=="mean"){
    pred <- spglm_pred_mean(beta=object$coefficients, data_object, link=object$link)
  } else
  if (type=="prob"){
    if (!missing(newdata) && (missing(newn) || missing(newevents)))
```

```
        stop("For prediction of probability vectors with new data, cluster sizes should be specified in 'n
      pred <- spglm_probs(beta=object$coefficients, f0=object$f0,
                               data_object=data_object, link=object$link)
   } else
   if (type == "lp"){
      pred <- spglm_lp(beta=object$coefficients, data_object=data_object)
   } else
   if (type == "tilt"){
      pred <- spglm_tilt(beta=object$coefficients, f0=object$f0,
                        data_object=data_object, link=object$link)
   }
   return(pred)
}
◇
```

File defined by 1, 3, 4, 5.

3. EM algorithm for model fitting

We will combine an Expectation Maximization with the Newton-Raphson algorithm proposed by Rathouz and Gao (2009) to estimate the model parameters. Recall that the observed data consists of the number of events, cluster size, and cluster-level covariates: $(y_i, N_i, \boldsymbol{Z}_i)$, $i = 1, \cdots, I$.

3.1. **Observed data log-likelihood.**

Under the marginal compatibility assumption, the observed log-likelihood function is:

$$\begin{aligned}\ell(q_{y,N}, \boldsymbol{\beta}) &= \sum_{i=1}^{I} \log\{q_{y_i,N_i}(\boldsymbol{Z}_i, \boldsymbol{\beta})\} \\ &= \sum_{i=1}^{I} \log\left[\binom{N_i}{y_i} \sum_{t=y}^{N-N_i+y_i} \frac{\binom{N-N_i}{t-y_i}}{\binom{N}{t}} \times \frac{q^{(0)}_{t,N} \times \exp\{\omega(\boldsymbol{Z}_i;\boldsymbol{\beta})t\}}{\sum_{t'=0}^{N} q^{(0)}_{t',N} \times \exp\{\omega(\boldsymbol{Z}_i;\boldsymbol{\beta})t'\}}\right].\end{aligned} \tag{3}$$

3.2. **Model prediction and likelihood.**

We define internal function to calculate predicted values and the log-likelihood.

"../R/SPGLM.R" 5≡

```
#' @keywords internal
spglm_lp <- function(beta, data_object){
  eta <- c(data_object$model_matrix %*% beta + data_object$offset)
  eta
}

spglm_pred_mean <- function(beta, data_object, link){
  eta <- c(data_object$model_matrix %*% beta + data_object$offset)
  mu <- link$linkinv(eta)
  mu
}

spglm_tilt <- function(beta, f0, data_object, link){

    mu <- spglm_pred_mean(beta=beta, data_object=data_object, link=link)
    N <- data_object$maxN
    nobs <- nrow(data_object$model_matrix)

    # ySptIndex is only used to calculate the log-likelihood, which we will not be using
    th <- getTheta(spt=(0:N)/N, f0=f0, mu=mu, weights=data_object$weights, ySptIndex=rep(1, nobs),
                   thetaStart=NULL, thetaControl=theta.control())
    th$theta
}

spglm_probs <- function(beta, f0, data_object, link){

    mu <- spglm_pred_mean(beta=beta, data_object=data_object, link=link)
    N <- data_object$maxN
    nobs <- nrow(data_object$model_matrix)

    # ySptIndex is only used to calculate the log-likelihood, which we will not be using
    th <- getTheta(spt=(0:N)/N, f0=f0, mu=mu, weights=data_object$weights, ySptIndex=rep(1, nobs),
                   thetaStart=NULL, thetaControl=theta.control())

    hp <- sapply(0:N, function(t)dhyper(x=data_object$resp[,1], m=data_object$n, n=N-data_object$n, k=t))
    probs <- rowSums(t(th$fTilt) * hp)
    probs
}

spglm_loglik <- function(beta, f0, data_object, link){

    probs <- spglm_probs(beta, f0, data_object, link)
    llik <- log(probs) %*% data_object$weights
    c(llik)
}
◇
```

File defined by 1, 3, 4, 5.

It is difficult to utilize the observed log-likelihood directly for parameter estimation. Its score function is complex because there is a logarithm of a sum in (3). Therefore, we implement an EM algorithm for parameter estimation.

⟨ *Fit model* 6 ⟩ ≡

```
 ⟨ Set starting values 9a ⟩
 ⟨ Set up for E-step 7 ⟩

 iter <- 0
 difference <- 100
 while (iter < control$maxit & difference > control$eps) {
    iter <- iter + 1
    referencef0Pre <- referencef0
    betasPre <- betas
    llikPre <- llik

    ⟨ E-step 8a ⟩
    ⟨ M-step 8b ⟩

    difference <- abs(llik - llikPre)
  }

  ⟨ Calculate standard errors 9b ⟩

◇
```

Fragment referenced in 1.

3.3. **Missing data setup.** Stefanescu et al (2003) have proved that the marginal compatibility assumption is equivalent to assuming that each cluster of size N' arises from a cluster of maximal cluster size N with some binary observations missing completely missing at random (MCAR). The missing data setup and the expectation step from EM algorithm are based on this MCAR interpretation.

For the setup of the EM algorithm, the complete data are clusters of size N with a corresponding number of events s_i, where $s_i \in \{y_i, \cdots, N\}$. The missing data correspond to the unobserved outcomes in of the $(N - N_i)$ cluster components. The missing data can be summarized as $(s_i - y_i)$ events out of $(N - N_i)$ elements.

The complete data log-likelihood is:

$$\begin{aligned}\ell_{\text{complete}}(q_{y,N}, \boldsymbol{\beta} \mid N_i, s_i) &= \sum_{i=0}^{I} \log\{q_{s_i,N}(\boldsymbol{Z}_i; \boldsymbol{\beta})\} \\ &= \sum_{i=0}^{I} \sum_{s=y_i}^{N} \mathbb{1}(s_i = s) \times \log\{q_{s,N}(\boldsymbol{Z}_i; \boldsymbol{\beta})\},\end{aligned} \tag{4}$$

where $q_{s_i,N}$ is the probability of achieving s_i events in a cluster of size N in the complete data set. The $\mathbb{1}$ denotes the indicator function.

Equation 4 can be interpreted as the log-likelihood of a model with fixed cluster size N for an expanded data set: each observation with cluster size N_i and number of responses y_i is expanded to $(N + 1)$ replicates with $0, 1, \ldots, N$, responses, respectively.

⟨ *Set up for E-step* 7 ⟩ ≡

```
# replace each cluster with N+1 clusters of size N
new <- cbind(y=0:N)
rep_idx <- rep(1:nrow(Y), each=nrow(new))
Y2 <- cbind(1:nrow(Y), Y)[rep_idx,]
colnames(Y2) <- c("i", "resp","nonresp")

rep_idx2 <- rep(1:nrow(new), times=nrow(Y))
Ycomb <- cbind(Y2, new[rep_idx2, ,drop=FALSE])
# select possible combinations
possible <- (Ycomb[,"y"] >= Ycomb[,"resp"]) & (N-Ycomb[,"y"] >= Ycomb[,"nonresp"])
Ycomb <- Ycomb[possible,]

obs_start <- match(1:nobs, Ycomb[,"i"])

mm2 <- mm[Ycomb[,"i"], ,drop=FALSE]
weights2 <- weights[Ycomb[,"i"]]
offset2 <- offset[Ycomb[,"i"]]
```
◇

Fragment referenced in 6.

3.4. Expectation step.

We take the expectation of complete data log-likelihood (4) given parameters from the previous k^{th} iteration, which is shown as follows:

$$\begin{aligned}Q\left(q_{y,N}^{(k)}, \boldsymbol{\beta}^{(k)}\right) &= \mathbb{E}\left\{\ell_{\text{complete}}(q_{y,N}, \boldsymbol{\beta} \mid N_i, s_i) \mid q_{y,N}^{(k)}, \boldsymbol{\beta}^{(k)}, N_i, y_i\right\} \\ &= \mathbb{E}\left[\sum_{i=0}^{I}\sum_{s=0}^{N} \mathbb{1}(s = s_i) \times \log\{q_{s,N}(\boldsymbol{Z}_i; \boldsymbol{\beta})\} \mid q_{y,N}^{(k)}, y_i, N_i\right] \\ &= \sum_{i=0}^{I}\sum_{s=0}^{N} \underbrace{\mathbb{E}\left\{\mathbb{1}(s = s_i) \mid q_{y,N}^{(k)}, \boldsymbol{\beta}^{(k)}, y_i, N_i\right\}}_{p^{(k)}_{N_i y_i s_i}} \times \log\{q_{s,N}(\boldsymbol{Z}_i; \boldsymbol{\beta})\},\end{aligned} \tag{5}$$

where $p_{N'y'y} = \mathbf{Pr}_N(Y = y \mid Y' = y', N')$ denotes the conditional probability of achieving y events in the complete cluster of size N, given that y' events have been observed in the original cluster of size N'. The expression for $p_{N'y'y}$ can be derived from $q_{y,N}$ using Bayes theorem and the MCAR interpretation as follows:

$$\begin{aligned} p_{N'y'y} &= \mathbf{Pr}_N(Y = y \mid Y' = y', N') \\ &= \frac{\mathbf{Pr}_{N'}(Y' = y' \mid Y = y)\mathbf{Pr}_N(Y = y)}{\sum_{t=0}^{N} \mathbf{Pr}_{y'}(Y' = y' \mid Y = t)\mathbf{Pr}_N(Y = t)} \\ &= \frac{\binom{y}{y'}\binom{N-y}{N'-y'} q_{y,N}}{\sum_{t=y'}^{N-N'+y'} \binom{t}{y'}\binom{N-t}{N'-y'} q_{t,N}}, \end{aligned} \tag{6}$$

and $p^{(k)}_{N_i y_i s_i}$ at the kth step of the iteration can be obtained by using $q^{(k)}_{y,N}$ on the right-hand side of the expression.

The value of $q^{(k)}_{y,N}((\boldsymbol{Z}_i; \boldsymbol{\beta}^{(k)})$ can be obtained from ${q^{(0)}_{y,N}}^{(k)}$ using 2. But it is also returned by `gldrmFit` as `fTiltMatrix`.

⟨ *E-step* 8a ⟩ ≡

```
# convert fTiltMatrix to long vector, watching out for potentially different support
y_idx <- match(Ycomb[,"y"], spt)
fTilt2 <- ifelse(!is.na(y_idx), fTiltMatrix[cbind(Ycomb[,"i"], y_idx)], 0)

# numerator of weights
pp_num <- choose(n=Ycomb[,"y"], k=Ycomb[,"resp"]) *
          choose(n=N-Ycomb[,"y"], k=Ycomb[,"nonresp"]) *
          fTilt2
# denominator
pp_denom <- tapply(pp_num, list(i=Ycomb[,"i"]), sum, simplify=TRUE)
pp <- c(pp_num / pp_denom[Ycomb[,"i"]])
if (!is.null(weights)) pp <- weights2 * pp
```
◇

Fragment referenced in 6.

3.5. **Maximization step.** Equation (5) can be interpreted as the log-likelihood for the SPGLM with fixed cluster size N for an expanded data set: each observation with cluster size N_i and number of responses y_i is expanded to $(N + 1)$ replicates with $0, 1, \ldots, N$, responses, respectively. The probabilities $p^{(k)}_{N_i y_i s_i}$ serve as cluster weights in the expanded data set. From here we can use a slight generalization of the Newton-Raphson algorithm developed by Wurm and Rathouz (2018) to estimate the baseline density distribution $\boldsymbol{q}^{(0)}_N$ and the regression coefficients $\boldsymbol{\beta}$.

Default settings are used for the algorithm, except the starting values is updated based on the previous iteration

⟨ *M-step* 8b ⟩ ≡

```
gldrmControl0 <- gldrm.control(returnfTiltMatrix = TRUE, returnf0ScoreInfo = FALSE, print=FALSE,
                               betaStart = betasPre, f0Start = referencef0Pre)

mod <- gldrmFit(x = mm2, y=Ycomb[,"y"]/N, linkfun=link$linkfun, linkinv = link$linkinv,
                mu.eta = link$mu.eta, mu0 = mu0, offset = offset2, weights = pp,
                gldrmControl = gldrmControl0,  thetaControl=theta.control(),
                betaControl=beta.control(), f0Control=f0.control())

fTiltMatrix <- mod$fTiltMatrix[obs_start,]
betas <- mod$beta
referencef0 <- mod$f0
spt <- round(mod$spt * N)
llik <- c(log(rowSums(fTiltMatrix * hp)) %*% weights)
```
◇

Fragment referenced in 6.

3.6. **Starting values.** We start the regression coefficients are set to 0, so $q_N^{(0)}$ would be the overall estimate under marginally compatibility. The default value of mu0 is set to the mean of y_i/n_i.

⟨ *Set starting values* 9a ⟩ ≡

```
betas <- NULL
pooled <- CBData(data.frame(Trt = "All", NResp = Y[,1], ClusterSize = rowSums(Y), Freq=ceiling(weights)
                  trt="Trt", clustersize="ClusterSize", nresp="NResp", freq="Freq")
est <- mc.est(pooled)
referencef0 <- est$Prob[est$ClusterSize == N]
# ensure all positive values
referencef0 <- (referencef0 + 1e-6)/(1+(N+1)*1e-6)

fTiltMatrix <- matrix(rep(referencef0, times=nobs), byrow=TRUE, nrow=nobs, ncol=N+1)
spt <- 0:N

if (is.null(mu0))
  mu0 <- weighted.mean(Y[,1]/rowSums(Y), weights)

# hypergeometric terms for log-likelihood calculation
hp <- sapply(0:N, function(t)dhyper(x=Y[,1], m=rowSums(Y), n=N-rowSums(Y), k=t))

# initial log-likelihood
llik <- log(rowSums(fTiltMatrix * hp)) %*% weights
```

◇

Fragment referenced in 6.

3.7. **Standard errors.** We will use numeric derivation to obtain the hessian of the observed data likelihood. The variance-covariance matrix can be estimated as the inverse of the negative hessian. The sum-to-one and fixed-mean constraints of the reference distribution are included by expanding the hessian to a bordered hessian before inversion by including the derivatives of the constraints (this can be derived based on the Lagrange multiplier method). The reference distribution is included in the hessian on the log-scale, and is multiplied by the gradient before inversion to revert to the original scale.

⟨ *Calculate standard errors* 9b ⟩ ≡

```
ll <- function(x){
  spglm_loglik(beta=x[1:p], f0 = exp(x[-(1:p)]), data_object=data_object, link=link)
}

hess <- numDeriv::hessian(func=ll,  x=c(betas, log(referencef0)))

# revert to unlogged f0
grad <- c(rep(1, p), 1/referencef0)
hess1 <- diag(grad) %*% hess %*% diag(grad)

# create bordered hessian
border1 <- c(rep(0, p), rep(1, N+1))  # gradient of sum-to-one constraint
border2 <- c(rep(0, p), 0:N)          # gradient of fixed-mean constraint
bhess <- rbind(cbind(hess1, border1, border2), c(border1,0,0), c(border2,0,0))

# calculate variance-covariance matrix
vc <- solve(-bhess)
SEbeta <- sqrt(diag(vc)[1:p])
```

```
    SEf0 <- sqrt(diag(vc)[p+1+(0:N)])
  ◇
```

Fragment referenced in 6.