

1 Introduction

This document describes the core TRIM function.

Define a convenience function for console output

```
> printf <- function(fmt,...) { cat(sprintf(fmt,...)) }
```

1.1 Interface

The main TRIM function takes just two parameters: commands, wrapped in a TCF data structure, and a data set.

```
> trim <- function(tcf, dat) {  
>   start = Sys.time()  
> }
```

1.1.1 Preparation

Get job parameters, use defaults if necessary

```
> file <- tcf@file  
> model <- tcf@model  
> title <- ifelse(is.na(tcf@title), "<Untitled>", tcf@title)  
> weight <- ifelse(is.na(tcf@weight), FALSE, tcf@weight)  
> missing_code <- tcf@missing Use any j0 if this is missing  
> serialcor <- ifelse(is.na(tcf@serialcor), FALSE, tcf@serialcor)  
> overdisp <- ifelse(is.na(tcf@overdisp), FALSE, tcf@overdisp)  
  
> if (model==2) {  
>   changepoints <- tcf@changepoints  
> }
```

Finalize input by discarding the TCF objects. All info should have been extracted from it by now.

```
> rm(tcf)
```

Create observation matrix f . Convert the data from a data frame representation to a matrix representation. It's OK to have missing site/time combinations; these will automatically translate to NA values.

```
> nsite <- dat$nsite  
> ntime <- dat$ntime  
> f <- matrix(0, nsite, ntime)  
> rows <- as.integer(dat$df$site) 'site' is a factor, thus this results in 1...I.  
> cols <- as.integer(dat$df$time) idem, 1...J.  
> idx <- (cols-1)*nsite+rows Create column-major linear index from row/column subscripts.  
> f[idx] <- dat$df$count ... such that we can paste all data into the right positions
```

We often need some specific subset of the data, e.g. all observations for site 3. These are conveniently found by combining the following indices:

```
> observed <- is.finite(f) Flags observed (TRUE) / missing (FALSE) data  
> site <- as.vector(row(f)) Internal site identifiers are the row numbers of the original matrix.  
> time <- as.vector(col(f)) Idem for time points.  
> nobs <- rowSums(observed) Number of actual observations per site
```

For model 2, we do not allow for changepoints < 1 or $\geq J$. At the same time, a changepoint 1 must be present

```
> if (model==2) {  
>   stopifnot(all(changepoints>=1))  
>   stopifnot(all(changepoints<ntime))  
>   stopifnot(all(diff(changepoints)>0))  
> }
```

```
>   if (changepoints[1]!=1) changepoints = c(1, changepoints)
> }
```

We make use of the generic model structure

$$\log \mu = A\alpha + B\beta$$

where design matrices A and B both have IJ rows. For efficiency reasons the model estimation algorithm works on a per-site basis. There is thus no need to store these full matrices. Instead, B is constructed as a smaller matrix that is valid for any site, and A is not used at all.

Create matrix B , which is model-dependent.

```
> if (model==2) {
>   ncp <- length(changepoints)
>   J <- ntime
>   B = matrix(0, J, ncp)
>   for (i in 1:ncp) {
>     cp1 <- changepoints[i]
>     cp2 <- ifelse(i<ncp, changepoints[i+1], J)
>     if (cp1>1) B[1:(cp1-1), i] <- 0
>     B[cp1:cp2,i] <- 0:(cp2-cp1)
>     if (cp2<J) B[(cp2+1):J,i] <- B[cp2,i]
>   }
> } else if (model==3) {
```

Model 3 in it's canonical form uses a single time parameter γ per time step, so design matrix B is essentially a $J \times J$ identity matrix. Note, however, that by definition $\gamma_1 = 0$, so effectively there are $J - 1$ γ -values to consider. As a consequence, the first column is deleted.

```
>   B <- diag(ntime) Construct J×J identity matrix
>   B <- B[, -1] Remove first column3fff
> }
```

1.1.2 Setup parameters and state variables

Parameter α has a unique value for each site.

```
> alpha <- matrix(0, nsite,1) Store as column vector
```

Parameter β is model dependent.

```
> if (model==2) {
```

For model 2 we have one β per change points

```
>   beta = matrix(0, length(changepoints), 1)
> } else if (model==3) {
```

For model 3, we have one β per time $j > 1$; all are initialized to 0 (no time effects)

```
>   beta <- matrix(0, ntime-1,1) Store as column vector
> }
```

Variable μ holds the estimated counts.

```
> mu <- matrix(0, nsite, ntime)
```

1.2 Model estimation.

TRIM estimates the model parameters α and β in an iterative fashion:

$$\alpha_i^t = \log z_i' f_i - \log z_i' \exp(B_i \beta^{t-1}) \quad (1)$$

$$\mu^t = \exp(A \alpha^t + B \beta^{t-1} - \log w) \quad (2)$$

$$\beta^t = \beta^{t-1} - (i_b)^{-1} U_b^* \quad (3)$$

where the superscript t refers to the iteration.

Derivative matrix i_b is defined as

$$-i_b = \sum_i B_i' \left(\Omega_i - \frac{1}{d_i} \Omega_i z_i z_i' \Omega_i \right) B_i \quad (4)$$

where

$$\Omega_i = \text{diag}(\mu_i) V_i^{-1} \text{diag}(\mu_i) \quad (5)$$

with V_i the covariance matrix for site i , and

$$d_i = z_i' \Omega_i z_i \quad (6)$$

Covariance matrix V_i is defined by

$$V_i = \sigma^2 \sqrt{\text{diag}(\mu)} R \sqrt{\text{diag}(\mu)} \quad (7)$$

where σ^2 is a dispersion parameters and R is an (auto)correlation matrix. Both of these two elements are optional. If the counts are perfectly Poisson distributed, $\sigma^2 = 1$, and if autocorrelation is disabled (i.e. counts are independent), Eqn (??) reduces to

$$V_i = \sigma^2 \text{diag}(\mu) \quad (8)$$

Dispersion parameter σ^2 is estimated as

$$\hat{\sigma}^2 = \frac{1}{n_f - n_\alpha - n_\beta} \sum_{i,j} r_{ij}^2 \quad (9)$$

where the n terms are the number of observations, α 's and β 's, respectively. Summation is over the observed i, j only. and r_{ij} are Pearson residuals given by

$$r_{ij} = (f_{ij} - \mu_{ij}) / \sqrt{\mu_{ij}} \quad (10)$$

Summarizing, estimation of α and β involves the iterative computation of, in order, α , μ , r , σ^2 , R , V , d , Ω , i_b , U_b and β .

```
> max_iter <- 200  Define a maximum number of iterations to detect failure to converge
> chi2 = 1000;
> lik = 1000;
> dryruns = 3
> stepsize      = 1
> for (iter in 1:max_iter) {
```

Remember current parameter values to trace convergence

```
>   old_par <- c(as.vector(alpha), as.vector(beta))
>   old_cnt <- as.vector(mu)
>   old_lik <- lik
```

1.2.1 Update site-parameters α

Update α_i using (??):

$$\alpha_i^t = \log z'_i f_i - \log z'_i \exp(B_i \beta^{t-1})$$

where it is noted that for any vector v , $z'v$ is equivalent to the sum of the elements of v

```
> for (i in 1:nsite) {
>   fi <- f[site==i & observed==TRUE] vector
>   Bi <- B[observed[site==i], , drop=FALSE]
>   alpha[i] <- log(sum(fi)) - log(sum(exp(Bi %*% beta)))
> }
```

1.2.2 Update count estimates μ

Update μ using Eqn (??)

$$\mu^t = \exp(A\alpha^t + B\beta^{t-1} - \log w)$$

where it is noted that we do not use matrix A . Instead, the site-specific parameters α_i are used directly:

$$\mu_i^t = \exp(\alpha_i^t + B\beta^{t-1} - \log w)$$

```
> for (i in 1:nsite) {
>   mu[i, ] <- exp(alpha[i] + B %*% beta)
> }
```

1.2.3 Pearson residuals

Compute Pearson residuals using

$$r_{ij} = (f_{ij} - \mu_{ij}) / \sqrt{\mu_{ij}} \quad (11)$$

```
> r <- matrix(0,nsite,ntime) Use 0 instead of NA for missing cases to increase performance
> for (i in 1:nsite) for (j in 1:ntime) if (observed[i,j]) {
>   r[i,j] <- (f[i,j]-mu[i,j]) / sqrt(mu[i,j])
> }
```

1.2.4 (Over)dispersion

Estimate the dispersion parameter σ^2 from

$$\hat{\sigma}^2 = \frac{1}{n_f - n_\alpha - n_\beta} \sum_{i,j} r_{ij}^2 \quad (12)$$

where the n terms are the number of observations, α 's and β 's, respectively. Summation is over the observed i, j only. Note that the dispersion parameter is only computed when asked for.

```
> if (iter>dryruns && overdisp) {
>   df <- sum(nobs) - length(alpha) - length(beta) degrees of freedom
>   sig2 <- sum(r^2) / df
> } else {
>   sig2 <- 1.0
> }
```

1.2.5 Autocorrelation

The (optional) autocorrelation structure for any site i is stored in $n_i \times n_i$ matrix R_i . In case there are no missing values, $n_i = J$, and the ‘full’ or ‘generic’ autocorrelation matrix R is expressed as

$$R = \begin{pmatrix} 1 & \rho & \rho^2 & \dots & \rho^{J-1} \\ \rho & 1 & \rho & \dots & \rho^{J-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho^{J-1} & \rho^{J-2} & \rho^{J-3} & \dots & 1 \end{pmatrix} \quad (13)$$

where ρ is the lag-1 autocorrelation, estimated as

$$\hat{\rho} = \frac{1}{n_{i,j,j+1}\hat{\sigma}^2} \left(\sum_i^I \sum_j^{J-1} r_{i,j} r_{i,j+1} \right) \quad (14)$$

where the summation is over observed pairs $i, j-i, j+1$, and $n_{i,j,j+1}$ is the number of pairs involved. Again, both ρ and R are computed ρ in a stepwise per-site fashion. Also, site-specific autocorrelation matrices R_i are formed by removing the rows and columns from R corresponding with missing observations.

First estimate ρ

```
> if (iter>dryruns && serialcor) {
>   rho <- 0.0
>   count <- 0
>   for (i in 1:nsite) for (j in 1:(ntime-1)) {
>     if (observed[i,j] && observed[i,j+1]) { short-circuit AND intended
>       rho <- rho + r[i,j] * r[i,j+1]
>       count <- count+1
>     }
>   }
>   rho <- rho / (count * sig2)
> } else rho = 0.0
```

Then build the ‘generic’ (site-independent) matrix R :

```
> if (iter>dryruns && serialcor) {
>   Rg <- rho ^ abs(row(diag(ntime)) - col(diag(ntime)))
> }
```

1.2.6 Update V , i_b and U_b .

Information matrix i_b Scorematrix U_b^* are both dependent on site-specific matrices V_i and Ω_i , which we compute on the fly along with site contributions to i_b and U_b .

```
> i_b <- 0
> U_b <- 0
> Omega <- vector("list", nsite) store  $\Omega_i$ 's for later use
> for (i in 1:nsite) {
>   mu_i <- mu[site==i & observed]
>   f_i <- f[site==i & observed]
>   d_mu_i <- diag(mu_i, length(mu_i)) Length argument guarantees diag creation
>   if (iter>dryruns && serialcor) {
>     idx <- which(observed[i, ])
>     R_i <- Rg[idx,idx]
>     V_i <- sig2 * sqrt(d_mu_i) %*% R_i %*% sqrt(d_mu_i)
>   } else {
>     V_i <- sig2 * d_mu_i
>   }
>   V_inv <- solve(V_i)
>   Omega[[i]] <- d_mu_i %*% V_inv %*% d_mu_i
```

```

> z <- matrix(1, nobs[i], 1)
> d_i <- as.numeric(t(z) %*% Omega[[i]] %*% z)
> #d_i <- sum(Omega[[i]]) equivalent to  $z' \Omega_i z$ .
> B_i <- B[observed[site==i], ,drop=FALSE]
> term <- t(B_i) %*% (Omega[[i]] - (Omega[[i]] %*% z %*% t(z) %*% Omega[[i]]) / d_i) %*% B_i
> i_b <- i_b - term
> U_b <- U_b + t(B_i) %*% d_mu_i %*% V_inv %*% (f_i - mu_i)
> }

```

1.2.7 Check for convergence.

The iterative estimation of α and β is finished if convergence is reached. This measured by the *change in* α, β . Iteration stops if this change drops below a certain threshold. Note that we check for convergence BEFORE we update beta...

```

> lik <- 2*sum(f*log(f/mu), na.rm=TRUE)

```

First compute the change in parameter values

```

> new_par <- c(as.vector(alpha), as.vector(beta))
> new_cnt <- as.vector(mu)
> new_lik <- lik
> max_par_change <- max(abs(new_par - old_par)) use the maximub absolute change
> max_cnt_change <- max(abs(new_cnt - old_cnt)) use the maximub absolute change
> max_lik_change <- max(abs(new_lik - old_lik)) use the maximub absolute change
> rel_lik_change <- 100*abs(old_lik - new_lik)/abs(old_lik)

```

Write out progress, and some more info

```

> cat(sprintf("Iteration %2d;", iter))
> cat(sprintf(" lik=%3f", lik))
> if (overdisp) cat(sprintf(" sig^2=%f", sig2))
> if (serialcor) cat(sprintf(" rho=%f;", rho))
> cat(sprintf(" Max change: %10g %10g %10g", max_par_change, max_cnt_change, max_lik_change))
> cat("\n")

```

Exit loop when convergence has been reached

```

> conv_par = max_par_change < 1e-7
> conv_cnt = max_cnt_change < 1e-7
> conv_lik = max_lik_change < 1e-7
> if (conv_par) cat("Convergence reached (in parameters).\n")
> if (conv_cnt) cat("Convergence reached (in counts).\n")
> if (conv_lik) cat("Convergence reached (in likelihood).\n")
> if (conv_par && conv_cnt && conv_lik) break

```

1.2.8 Update β

Finally, we can update β using (??):

$$\beta^t = \beta^{t-1} - (i_b)^{-1} U_b^*$$

```

> dbeta <- - solve(i_b) %*% U_b
> beta <- beta + stepsize*dbeta
> }

```

If we reach the preset maximum number of iterations, we clearly have not reached convergence.

```

> if (iter==max_iter) stop("No convergence reached.")

```

Run the final model

```
> for (i in 1:nsite) {
>   mu[i, ] <- exp(alpha[i] + B %*% beta)
> }
```

1.3 Imputation

The imputation process itself is trivial: just replace all missing observations $f_{i,j}$ by the model-based estimates $\mu_{i,j}$.

```
> imputed <- ifelse(observed, f, mu)
```

1.4 Output and postprocessing

Measured, modelled and imputed count data are stored in a TRIM output object, together with parameter values and other useful information.

```
> z <- list(title=title, data=f, nsite=nsite, ntime=ntime,
>           model=model, mu=mu, imputed=imputed, alpha=alpha, beta=beta)
> class(z) <- "trim"
```

Several kinds of statistics can now be computed, and added to this output object.

1.4.1 Overdispersion and Autocorrelation

```
> z$sig2 <- ifelse(overdisp, sig2, NA)
> z$rho <- ifelse(serialcor, rho, NA)
```

1.4.2 Coefficients and uncertainty

```
> if (model==2) {
>   beta <- as.vector(beta)
>   var_beta <- -solve(i_b)
>   se_beta <- sqrt(diag(var_beta))
> }
```

Again, results are stored in the TRIM object

```
> z$coefficients <- data.frame(
>   Additive = beta,
>   std.err. = se_beta,
>   Multiplicative = exp(beta),
>   std.err. = exp(beta) * se_beta,
>   check.names = FALSE to allow for 2 "std.err." columns
> )
> row.names(z$coefficients) <- "Slope"
> }
> if (model==3) {
```

Model coefficients are output in two types; as additive parameters:

$$\log \mu_{ij} = \alpha_i + \gamma_j$$

and as multiplicative parameters:

$$\mu_{ij} = a_i g_j$$

where $a_i = e^{\alpha_i}$ and $g_j = e^{\gamma_j}$.

```
> gamma <- matrix(c(0, as.vector(beta))) Add  $\gamma_1 \equiv 1$ , and cast as column vector
> g <- exp(gamma)
```

Parameter uncertainty is expressed as standard errors. For the additive parameters γ , the variance is estimated as

$$\text{var}(\gamma) = (-i_b)^{-1}$$

```
> var_gamma <- -solve(i_b)
```

Because $\gamma_1 \equiv 1$, it was not estimated, and as a result $j = 1$ was not included in i_b , nor in $\text{var}(\text{gamma})$ as computed above. We correct this by adding the ‘missing’ rows and columns.

```
> var_gamma <- cbind(0, rbind(0, var_gamma))
```

Finally, we compute the standard error as $\text{S.E.}(\gamma) = \sqrt{\text{diag}(\text{var}(\gamma))}$

```
> se_gamma <- sqrt(diag(var_gamma))
```

The standard error of the multiplicative parameters g_j is approximated by using the delta method, which is based on a Taylor expansion:

$$\text{var}(f(\theta)) = (f'(\theta))^2 \text{var}(\theta) \quad (15)$$

which for $f(\theta) = e^\theta$ translates to

$$\text{var}(g) = \text{var}(e^\gamma) = e^{2\gamma} \text{var}(\gamma)$$

leading to

$$\text{S.E.}(g) = e^\gamma \text{S.E.}(\gamma) = g \text{S.E.}(\gamma)$$

```
> se_g <- g * se_gamma
```

Again, results are stored in the TRIM object

```
> z$coefficients <- data.frame(
>   Time           = 1:ntime,
>   Additive       = gamma,
>   std.err.       = se_gamma,
>   Multiplicative = g,
>   std.err.       = g * se_gamma,
>   check.names    = FALSE to allow for 2 "std.err." columns
> )
> }
```

1.4.3 Goodness-of-fit

The goodness-of-fit of the model is assessed using three statistics: Chi-squared, Likelihood Ratio and Akaike Information Content.

The χ^2 (Chi-square) statistic is given by

$$\chi^2 = \sum_{ij} \frac{f_{i,j} - \mu_{i,j}}{\mu_{i,j}} \quad (16)$$

where the summation is over the observed i, j 's only. Significance is assessed by comparing against a χ^2 distribution with df degrees of freedom, equal to the number of observations minus the total number of parameters involved, i.e. $df = n_f - n_\alpha - n_\beta$.

```
> chi2 <- sum((f-mu)^2/mu, na.rm=TRUE)
> df   <- sum(observed) - length(alpha) - length(beta)
> p    <- 1 - pchisq(chi2, df=df)
```

Results are stored in the TRIM output object.

```
> z$chi2 <- list(chi2=chi2, df=df, p=p)
```

Similarly, the *Likelihood ratio* (LR) is computed as

$$\text{LR} = 2 \sum_{ij} f_{ij} \log \frac{f_{i,j}}{\mu_{i,j}} \quad (17)$$

and again compared against a χ^2 distribution.

```
> LR <- 2 * sum(f * log(f / mu), na.rm=TRUE)
> df <- sum(observed) - length(alpha) - length(beta)
> p <- 1 - pchisq(LR, df=df)
> z$LR <- list(LR=LR, df=df, p=p)
```

The Akaike Information Content (AIC) is related to the LR as:

```
> AIC <- LR - 2*df
> z$AIC <- AIC
```

1.4.4 Time Totals

Recompute i_b with final μ 's

```
> ib <- 0
> for (i in 1:nsite) {
>   mu_i <- mu[site==i & observed]
>   n_i <- length(mu_i)
>   d_mu_i <- diag(mu_i, n_i) Length argument guarantees diag creation
>   OM <- Omega[[i]]
>   d_i <- sum(OM) equivalent with z' Omega z, as in the TRIM manual
>   B_i <- B[observed[site==i], ,drop=FALSE]
>   om <- colSums(OM)
>   OMzzOM <- om %*% t(om) equivalent with OM z z' OM, as in the TRIM manual
>   term <- t(B_i) %*% (OM - (OMzzOM) / d_i) %*% B_i
>   ib <- ib - term
> }
```

Matrices E and F take missings into account

```
> E <- -ib
> nbeta <- length(beta)
> F <- matrix(0, nsite, nbeta)
> d <- numeric(nsite)
> for (i in 1:nsite) {
>   d[i] <- sum(Omega[[i]])
>   w_i <- colSums(Omega[[i]])
>   B_i <- B[observed[site==i], ,drop=FALSE]
>   F_i <- (t(w_i) %*% B_i) / d[i]
>   F[i, ] <- F_i
> }
```

Matrices G and H are for all μ 's

```
> GddG <- matrix(0, ntime, ntime)
> for (i in 1:nsite) {
>   for (j in 1:ntime) for (k in 1:ntime) {
>     GddG[j,k] <- GddG[j,k] + mu[i,j]*mu[i,k]/d[i]
>   }
> }
> GF <- matrix(0, ntime, nbeta)
> for (i in 1:nsite) {
>   for (j in 1:ntime) for (k in 1:nbeta) {
>     GF[j,k] <- GF[j,k] + mu[i,j] * F[i,k]
>   }
> }
> H <- matrix(0, ntime, nbeta)
> for (i in 1:nsite) {
```

```

>   for (k in 1:nbeta) for (j in 1:ntime) {
>     H[j,k] <- H[j,k] + B[j,k] * mu[i,j]
>   }
> }

> GFminH <- GF - H

All building blocks are ready. Use them to compute the variance

> var_tt_mod <- GddG + GFminH %*% solve(E) %*% t(GFminH)

Time totals of the model, and it's standard error

> tt_mod <- colSums(mu)
> se_tt_mod <- round(sqrt(diag(var_tt_mod)))

> tt_imp <- colSums(imputed)
> var_tt_imp = matrix(NA, ntime, ntime)
> se_tt_imp <- round(sqrt(diag(var_tt_imp)))

> z$time.totals <- data.frame(
>   Time      = 1:ntime,
>   Model     = round(tt_mod),
>   std.err.  = se_tt_mod,
>   Imputed   = round(tt_imp),
>   std.err.  = se_tt_imp,
>   check.names = FALSE
> )

```

1.4.5 Time indices

Time index τ_j is defined as time totals, normalized by the time total for the base year, i.e.

$$\tau_j = \mu_{+j} / \mu_{+1}$$

. Indices are computed for both the modelled and the imputed counts.

```

> ti_mod <- tt_mod / tt_mod[1]
> ti_imp <- tt_imp / tt_imp[1]

```

Uncertainty is again quantified as a standard error \sqrt{var} , approximated using the delta method, now extended for the multivariate case:

$$\text{var}(\tau_j) = \text{var}(f(\mu_{+1}, \mu_{+j})) = d^T V(\mu_{+1}, \mu_{+j}) d \quad (18)$$

where d is a vector containing the partial derivatives of $f(\mu_{+1}, \mu_{+j})$

$$d = \begin{pmatrix} -\mu_{+j} \mu_{+1}^{-2} \\ \mu_{+1}^{-1} \end{pmatrix} \quad (19)$$

and V the covariance matrix of μ_{+1} and μ_{+j} :

$$V(\mu_{+1}, \mu_{+j}) = \begin{pmatrix} \text{var}(\mu_{+1}) & \text{cov}(\mu_{+1}, \mu_{+j}) \\ \text{cov}(\mu_{+1}, \mu_{+j}) & \text{var}(\mu_{+j}) \end{pmatrix} \quad (20)$$

Note that for the base year, where $\tau_1 \equiv 1$, Eqn (??) results in $\text{var}(\tau_1) = 0$, which is also expected conceptually because τ_1 is not an estimate but an exact and fixed result.

```

> var_ti_mod <- numeric(ntime)
> for (j in 1:ntime) {
>   d <- matrix(c(-tt_mod[j] / tt_mod[1]^2, 1/tt_mod[1]))
>   V <- var_tt_mod[c(1,j), c(1,j)]
>   var_ti_mod[j] <- t(d) %*% V %*% d
> }

```

```

> }
> se_ti_mod <- sqrt(var_ti_mod)

```

Similarly for the Indices based on the imputed counts

```

> se_ti_imp <- numeric(ntime)
> for (j in 1:ntime) {
>   d <- matrix(c(-tt_imp[j]/tt_imp[1]^2, 1/tt_imp[1]))
>   V <- var_tt_imp[c(1,j), c(1,j)]
>   se_ti_imp[j] <- sqrt(t(d) %*% V %*% d)
> }

```

Store in TRIM output object

```

> z$time.index <- data.frame(
>   Time      = 1:ntime,
>   Model     = ti_mod,
>   std.err.  = se_ti_mod,
>   Imputed   = ti_imp,
>   std.err.  = se_ti_imp,
>   check.names = FALSE
> )

```

1.4.6 Reparameterisation of Model 3

Here we consider the reparameterization of the time-effects model in terms of a model with a linear trend and deviations from this linear trend for each time point. The time-effects model is given by

$$\log \mu_{ij} = \alpha_i + \gamma_j, \quad (21)$$

with γ_j the effect for time j on the log-expected counts and $\gamma_1 = 0$. This reparameterization can be expressed as

$$\log \mu_{ij} = \alpha_i^* + \beta^* d_j + \gamma_j^*, \quad (22)$$

with $d_j = j - \bar{j}$ and \bar{j} the mean of the integers j representing the time points. The parameter α_i^* is the intercept and the parameter β^* is the slope of the least squares regression line through the J log-expected time counts in site i and γ_j^* can be seen as the residuals of this linear fit. From regression theory we have that the ‘residuals’ γ_j^* sum to zero and are orthogonal to the explanatory variable, i.e.

$$\sum_j \gamma_j^* = 0 \quad \text{and} \quad \sum_j d_j \gamma_j^* = 0. \quad (23)$$

Using these constraints we obtain the equations:

$$\log \mu_{ij} = \alpha_i^* + \beta^* d_j + \gamma_j^* = \alpha_i + \gamma_j \quad (24)$$

$$\sum_j \log \mu_{ij} = J \alpha_i^* = J \alpha_i + \sum_j \gamma_j \quad (25)$$

$$\sum_j d_j \log \mu_{ij} = \beta^* \sum_j d_j^2 = \sum_j d_j \gamma_j, \quad (26)$$

where (24) is the re-parameterization equation itself and (25) and (26) are obtained by using the constraints (23)

From (25) we have that $\alpha_i^* = \alpha_i + \frac{1}{J} \sum_j \gamma_j$. Now, by using the equations (24) thru (26) and defining

$D = \sum_j d_j^2$, we can express the parameters β^* and γ^* as functions of the parameters γ as follows:

$$\beta^* = \frac{1}{D} \sum_j d_j \gamma_j, \quad (27)$$

$$\begin{aligned} \gamma_j^* &= \alpha_i + \gamma_j - \alpha_i^* - \beta^* d_j \quad (\text{using (5)}) \\ &= \alpha_i - \left(\alpha_i + \frac{1}{J} \sum_j \gamma_j \right) + \gamma_j - d_j \frac{1}{D} \sum_j d_j \gamma_j \\ &= \gamma_j - \frac{1}{J} \sum_j \gamma_j - d_j \frac{1}{D} \sum_j d_j \gamma_j. \end{aligned} \quad (28)$$

Since β^* and γ_j^* are linear functions of the parameters γ_j they can be expressed in matrix notation by

$$\begin{pmatrix} \beta^* \\ \gamma^* \end{pmatrix} = \mathbf{T} \gamma, \quad (29)$$

with $\gamma^* = (\gamma_1^*, \dots, \gamma_J^*)^T$, $\gamma = (\gamma_1, \dots, \gamma_J)^T$ and \mathbf{T} the $(J+1) \times J$ transformation matrix that transforms γ to $(\beta^*, (\gamma^*)^T)^T$. From (??) and (??) it follows that the elements of \mathbf{T} are given by:

$$\mathbf{T}_{(1,j)} = \frac{d_j}{D} \quad (i = 1, j = 1, \dots, J)$$

$$\mathbf{T}_{(i,j)} = 1 - \frac{1}{J} - \frac{1}{D} d_{i-1} d_j \quad (i = 2, \dots, J+1, j = 1, \dots, J, i-1 = j)$$

$$\mathbf{T}_{(i,j)} = -\frac{1}{J} - \frac{1}{D} d_{i-1} d_j \quad (i = 2, \dots, J+1, j = 1, \dots, J, i-1 \neq j)$$

```
> if (model==3) {
>   TT <- matrix(0, ntime+1, ntime)
>   J <- ntime
>   j <- 1:J; d <- j - mean(j) i.e. d_j = j - 1/J \sum_j j
>   D <- sum(d^2) i.e., D = \sum_j d_j^2
>   TT[1, ] <- d / D
>   for (i in 2:(J+1)) for (j in 1:J) {
>     if (i-1 == j) {
>       TT[i,j] <- 1 - (1/J) - d[i-1]*d[j]/D
>     } else {
>       TT[i,j] <- - (1/J) - d[i-1]*d[j]/D
>     }
>   }
> }
> xstar <- TT %*% gamma
> bstar <- xstar[1]
> gstar <- xstar[2:(J+1)]
```

The covariance matrix of the transformed parameter vector can now be obtained from the covariance matrix $\mathbf{T}\gamma$ of γ as

$$V \begin{pmatrix} \beta^* \\ \gamma^* \end{pmatrix} = \mathbf{T} V(\gamma) \mathbf{T}^T \quad (30)$$

```
> var_xstar <- TT %*% var_gamma %*% t(TT)
> var_bstar <- as.numeric(var_xstar[1,1])
> var_gstar <- var_xstar[-1,-1]
> se_bstar <- sqrt(var_bstar)
> se_gstar <- sqrt(diag(var_gstar))
> z$linear.trend <- data.frame(
>   Additive      = bstar,
>   std.err       = se_bstar,
```

```

> Multiplicative = exp(bstar),
> std.err.      = exp(bstar) * se_bstar,
> row.names     = "Slope",
> check.names   = FALSE)

```

Deviations from the linear trend

```

> z$deviations <- data.frame(
>   Time      = 1:ntime,
>   Additive   = gstar,
>   std.err.   = se_gstar,
>   Multiplicative = exp(gstar),
>   std.err.   = exp(gstar) * se_gstar,
>   check.names = FALSE
> )
> }

```

1.4.7 Wald test

```

> if (model==2) {
>   theta = beta[1]
>   var_theta = var_beta[1,1]
>   W <- t(theta) %*% solve(var_theta) %*% theta  Compute the Wald statistic
>   W <- as.numeric(W)  Convert from 1 x 1 matrix to proper atomic
>   df <- 1  degrees of freedom
>   p <- 1 - pchisq(W, df=df)  p-value, based on W being  $\chi^2$  distributed.
>   z$wald <- list(model=model, W=W, df=df, p=p)}

```

For Model 3, we use the Wald test to test if the residuals around the overall trend (i.e., the γ_j^*) significantly differ from 0. The Wald statistic used for this is defined as

$$W = \theta^T (\text{var}(\theta))^{-1} \theta \quad (31)$$

```

> if (model==3) {
>   theta <- matrix(gstar)  Column vector of all  $J$   $\gamma^*$ .
>   var_theta <- var_gstar  Covariance matrix; drop the  $\beta^*$  terms.

```

We now have J equations, but due to the double constraints 2 of them are linear dependent on the others. Let's confirm this:

```

> eig <- eigen(var_theta)$values
> stopifnot(sum(eig<1e-7)==2)

```

Shrink θ and it's covariance matrix to remove the dependent equations.

```

> theta <- theta[3:J]
> var_theta <- var_theta[3:J, 3:J]
> W <- t(theta) %*% solve(var_theta) %*% theta  Compute the Wald statistic
> W <- as.numeric(W)  Convert from 1 x 1 matrix to proper atomic
> df <- J-2  degrees of freedom
> p <- 1 - pchisq(W, df=df)  p-value, based on W being  $\chi^2$  distributed.
> z$wald <- list(model=model, W=W, df=df, p=p)
> }

```

1.4.8 Overall slope

The overall slope is computed for both the modeled and the imputed μ_+ 's. So we define a function to do the actual work

```

> .compute.overall.slope <- function(tt, var_tt) {
Use Ordinary Least Squares (OLS) to estimate slope parameter  $\beta$ 
> X <- cbind(1, seq_len(ntime)) design matrix
> y <- matrix(log(tt))
> bhat <- solve(t(X) %*% X) %*% t(X) %*% y OLS estimate of  $b = (\alpha, \beta)^T$ 
> yhat <- X %*% bhat

Apply the sandwich method to take heteroskedasticity into account
> dvtt <- 1/tt_mod derivative of  $\log \mu_+$ 
> Om <- diag(dvtt) %*% var_tt %*% diag(dvtt) var( $\log \mu_+$ )
> var_beta <- solve(t(X) %*% X) %*% t(X) %*% Om %*% X %*% solve(t(X) %*% X)
> b_err <- sqrt(diag(var_beta))

```

Compute the p -value, using the t -distribution

```

> df <- ntime - 2
> t_val <- bhat[2] / b_err[2]
> p <- 2 * pt(abs(t_val), df, lower.tail=FALSE)

```

Also compute effect size as relative change during the monitoring period.

```

> effect <- abs(yhat[J] - yhat[1]) / yhat[1]

```

Reverse-engineer the SSR (sum of squared residuals) from the standard error

```

> j <- 1:J
> D <- sum((j-mean(j))^2)
> SSR <- b_err[2]^2 * D * (J-2)

```

Export the results

```

> df <- data.frame(
> Additive = bhat,
> std.err. = b_err,
> Multiplicative = exp(bhat),
> std.err. = exp(bhat) * b_err,
> row.names = c("Intercept", "Slope"),
> check.names = FALSE
> )
> list(coef=df, p=p, effect=effect, J=J, tt=tt, err=z$time.totals[[3]], SSR=SSR)
> }

```

Compute the overall trends for both the modelled and the imputed counts, and store the results in the TRIM output

```

> z$overall <- list()
> z$overall$mod <- .compute.overall.slope(tt_mod, var_tt_mod)
> z$overall$imp <- .compute.overall.slope(tt_imp, var_tt_imp)

```

1.5 Return results

```

> duration = Sys.time() - start
> print(duration)

```

The TRIM result is returned to the user...

```

> z
> }

```

...which ends the main TRIM function.