

Supplement to “Nonparametric testing for multiple survival functions with non-inferiority margins”

Hsin-Wen Chang* Ian W. McKeague†

February 6, 2018

Contents

S.1 Proof of Theorem 1 (continued from Appendix B)	2
S.1.1 Proof of Lemma 3	2
S.1.2 Proof of (B.5)	5
S.1.3 Proofs of Lemmas 4 and 5	8
S.1.4 Properties of PAVA	10
S.2 Bounds on $\bar{\Delta}_j$ and $\tilde{\Delta}_j$	13
S.3 Remarks on Theorem 1 continued	15
S.4 Retention-of-effect approach	16
S.5 Additional simulation results	18
S.5.1 Accuracy and power	18
S.5.2 Power/sample size calculations and optimal allocation	18
S.6 R code	20
S.7 References	52

*Institute of Statistical Science, Academia Sinica, Taipei 11529, Taiwan.

†Department of Biostatistics, Columbia University, New York, NY 10032.

S.1 Proof of Theorem 1 (continued from Appendix B)

S.1.1 Proof of Lemma 3

Firstly, $-2 \log \mathcal{R}(t)$ can be written as

$$\begin{aligned} & -2 \sum_{j=1}^k \sum_{i \leq N_j(t)} \left\{ d_{ij} \log \left(\frac{d_{ij}}{r_{ij} + \bar{\Delta}_j} \right) + (r_{ij} - d_{ij}) \log \left(1 - \frac{d_{ij}}{r_{ij} + \bar{\Delta}_j} \right) \right\} \\ & + 2 \sum_{j=1}^k \sum_{i \leq N_j(t)} \left\{ d_{ij} \log \left(\frac{d_{ij}}{r_{ij} + \tilde{\Delta}_j} \right) + (r_{ij} - d_{ij}) \log \left(1 - \frac{d_{ij}}{r_{ij} + \tilde{\Delta}_j} \right) \right\}, \end{aligned} \quad (\text{S.1})$$

where we suppress the dependence of $\bar{\Delta}_j$ and $\tilde{\Delta}_j$ on t . The j -th term in (S.1) can be organized into

$$\sum_{i \leq N_j(t)} \left\{ -2(r_{ij} + \tilde{\Delta}_j - d_{ij}) \log \left(1 + \frac{\bar{\Delta}_j - \tilde{\Delta}_j}{r_{ij} + \bar{\Delta}_j - d_{ij}} \right) \right. \quad (\text{S.2a})$$

$$\begin{aligned} & + 2(r_{ij} + \tilde{\Delta}_j) \log \left(1 + \frac{\bar{\Delta}_j - \tilde{\Delta}_j}{r_{ij} + \tilde{\Delta}_j} \right) \\ & \left. + 2\tilde{\Delta}_j \log \left(1 + \frac{\bar{\Delta}_j - \tilde{\Delta}_j}{r_{ij} + \bar{\Delta}_j - d_{ij}} \right) - 2\tilde{\Delta}_j \log \left(1 + \frac{\bar{\Delta}_j - \tilde{\Delta}_j}{r_{ij} + \tilde{\Delta}_j} \right) \right\}. \end{aligned} \quad (\text{S.2b})$$

We deal with (S.2a) first. Since $r_{ij} = O_p(n)$, $d_{ij} = O_p(1)$ and $\bar{\Delta}_j, \tilde{\Delta}_j = O_p(\sqrt{n})$ (by Supplement Section S.2), we have that $(\bar{\Delta}_j - \tilde{\Delta}_j)/(r_{ij} + \bar{\Delta}_j - d_{ij}) = O_p(1/\sqrt{n})$ and $(\bar{\Delta}_j - \tilde{\Delta}_j)/(r_{ij} + \tilde{\Delta}_j) = O_p(1/\sqrt{n})$. Then by the Taylor expansion $\log(1 + x) = x - x^2/2 + x^3/3 - x^4/4 + O(x^5)$ as $x \rightarrow 0$, and rearranging the terms, (S.2a) can be written as

$$(\bar{\Delta}_j - \tilde{\Delta}_j)^2 \sum_{i \leq N_j(t)} \left(\frac{1}{r_{ij} + \bar{\Delta}_j - d_{ij}} - \frac{1}{r_{ij} + \tilde{\Delta}_j} \right) \quad (\text{S.3a})$$

$$- \frac{2}{3}(\bar{\Delta}_j - \tilde{\Delta}_j)^3 \sum_{i \leq N_j(t)} \left(\frac{1}{(r_{ij} + \bar{\Delta}_j - d_{ij})^2} - \frac{1}{(r_{ij} + \tilde{\Delta}_j)^2} \right) \quad (\text{S.3b})$$

$$+ \frac{1}{2}(\bar{\Delta}_j - \tilde{\Delta}_j)^4 \sum_{i \leq N_j(t)} \left(\frac{1}{(r_{ij} + \bar{\Delta}_j - d_{ij})^3} - \frac{1}{(r_{ij} + \tilde{\Delta}_j)^3} \right) \quad (\text{S.3c})$$

$$+ O_p(1/\sqrt{n}).$$

To analyze (S.3), we write the summation in (S.3a) as

$$\sum_{i \leq N_j(t)} \frac{r_{ij} - d_{ij}}{r_{ij} + \bar{\Delta}_j - d_{ij}} \frac{r_{ij}}{r_{ij} + \tilde{\Delta}_j} \frac{d_{ij}}{r_{ij}(r_{ij} - d_{ij})} = \frac{\hat{\sigma}_j^2(t)}{n_j} [1 - O_p(1/\sqrt{n})], \quad (\text{S.4})$$

where we have used $(r_{ij} - d_{ij})/(r_{ij} + \tilde{\Delta}_j - d_{ij}) = 1 - \tilde{\Delta}_j/(r_{ij} - \tilde{\Delta}_j - d_{ij}) = 1 - O_p(1/\sqrt{n})$ and $r_{ij}/(r_{ij} + \tilde{\Delta}_j) = 1 - \tilde{\Delta}_j/(r_{ij} + \tilde{\Delta}_j) = 1 - O_p(1/\sqrt{n})$. Thus the full expression of (S.3a) is

$$(\bar{\Delta}_j - \tilde{\Delta}_j)^2 \frac{\hat{\sigma}_j^2(t)}{n_j} [1 - O_p(1/\sqrt{n})] = (\bar{\Delta}_j - \tilde{\Delta}_j)^2 \frac{\hat{\sigma}_j^2(t)}{n_j} + O_p(1/\sqrt{n}),$$

by $(\bar{\Delta}_j - \tilde{\Delta}_j)^2 = O_p(n)$ (see Supplement Section S.2) and $\hat{\sigma}_j^2(t) = O_p(1)$ (see Section 2.1). Next, the summation in (S.3b) can be written as

$$\begin{aligned} & \sum_{i \leq N_j(t)} \left(\frac{1}{r_{ij} + \tilde{\Delta}_j - d_{ij}} - \frac{1}{r_{ij} + \tilde{\Delta}_j} \right) \left(\frac{1}{r_{ij} + \tilde{\Delta}_j - d_{ij}} + \frac{1}{r_{ij} + \tilde{\Delta}_j} \right) \\ &= O_p\left(\frac{1}{n}\right) \sum_{i \leq N_j(t)} \left(\frac{1}{r_{ij} + \tilde{\Delta}_j - d_{ij}} - \frac{1}{r_{ij} + \tilde{\Delta}_j} \right) \\ &= O_p\left(\frac{1}{n}\right) \frac{\hat{\sigma}_j^2(t)}{n_j} [1 - O_p(1/\sqrt{n})], \end{aligned}$$

where the last equality follows by (S.4). This and the fact that $(\bar{\Delta}_j - \tilde{\Delta}_j)^3 = O_p(n^{3/2})$ imply the full expression of (S.3b) is

$$-\frac{2}{3}(\bar{\Delta}_j - \tilde{\Delta}_j)^3 O_p\left(\frac{1}{n}\right) \frac{\hat{\sigma}_j^2(t)}{n_j} \left[1 - O_p\left(\frac{1}{\sqrt{n}}\right)\right] = O_p\left(\frac{1}{\sqrt{n}}\right).$$

Lastly, the summation in (S.3c) equals

$$\begin{aligned} & \sum_{i \leq N_j(t)} \left(\frac{1}{r_{ij} + \tilde{\Delta}_j - d_{ij}} - \frac{1}{r_{ij} + \tilde{\Delta}_j} \right) \times \\ & \quad \left[\frac{1}{(r_{ij} + \tilde{\Delta}_j - d_{ij})^2} + \frac{1}{(r_{ij} + \tilde{\Delta}_j - d_{ij})(r_{ij} + \tilde{\Delta}_j)} + \frac{1}{(r_{ij} + \tilde{\Delta}_j)^2} \right] \\ &= O_p\left(\frac{1}{n^2}\right) \sum_{i \leq N_j(t)} \left(\frac{1}{r_{ij} + \tilde{\Delta}_j - d_{ij}} - \frac{1}{r_{ij} + \tilde{\Delta}_j} \right) \\ &= O_p\left(\frac{1}{n^2}\right) \frac{\hat{\sigma}_j^2(t)}{n_j} [1 - O_p(1/\sqrt{n})], \end{aligned}$$

where the last equality again follows by (S.4). This and the fact that $(\bar{\Delta}_j - \tilde{\Delta}_j)^4 = O_p(n^2)$ imply the full expression of (S.3c) is

$$\frac{1}{2} (\bar{\Delta}_j - \tilde{\Delta}_j)^4 O_p\left(\frac{1}{n^2}\right) \frac{\hat{\sigma}_j^2(t)}{n_j} \left[1 - O_p\left(\frac{1}{\sqrt{n}}\right)\right] = O_p\left(\frac{1}{n}\right).$$

Combining the above results for (S.3a), (S.3b) and (S.3c), we have

$$(S.2a) = (\bar{\Delta}_j - \tilde{\Delta}_j)^2 \frac{\hat{\sigma}_j^2(t)}{n_j} + O_p\left(\frac{1}{\sqrt{n}}\right). \quad (S.5)$$

Now we deal with (S.2b). Again by a Taylor expansion and rearranging the terms, (S.2b) can be written as

$$2\tilde{\Delta}_j(\bar{\Delta}_j - \tilde{\Delta}_j) \sum_{i \leq N_j(t)} \left(\frac{1}{r_{ij} + \tilde{\Delta}_j - d_{ij}} - \frac{1}{r_{ij} + \tilde{\Delta}_j} \right) \quad (S.6a)$$

$$- \tilde{\Delta}_j(\bar{\Delta}_j - \tilde{\Delta}_j)^2 \sum_{i \leq N_j(t)} \left(\frac{1}{(r_{ij} + \tilde{\Delta}_j - d_{ij})^2} - \frac{1}{(r_{ij} + \tilde{\Delta}_j)^2} \right) \quad (S.6b)$$

$$+ \frac{2}{3}\tilde{\Delta}_j(\bar{\Delta}_j - \tilde{\Delta}_j)^3 \sum_{i \leq N_j(t)} \left(\frac{1}{(r_{ij} + \tilde{\Delta}_j - d_{ij})^3} - \frac{1}{(r_{ij} + \tilde{\Delta}_j)^3} \right) \quad (S.6c)$$

$$+ O_p(1/\sqrt{n}).$$

Utilizing our earlier results concerning the summations in (S.3a), (S.3b) and (S.3c),

$$(S.6a) = 2\tilde{\Delta}_j(\bar{\Delta}_j - \tilde{\Delta}_j) \frac{\hat{\sigma}_j^2(t)}{n_j} + O_p\left(\frac{1}{\sqrt{n}}\right),$$

$$(S.6b) = -\tilde{\Delta}_j(\bar{\Delta}_j - \tilde{\Delta}_j)^2 O_p\left(\frac{1}{n}\right) \frac{\hat{\sigma}_j^2(t)}{n_j} [1 - O_p(1/\sqrt{n})] = O_p\left(\frac{1}{\sqrt{n}}\right),$$

$$(S.6c) = \frac{2}{3}\tilde{\Delta}_j(\bar{\Delta}_j - \tilde{\Delta}_j)^3 O_p\left(\frac{1}{n^2}\right) \frac{\hat{\sigma}_j^2(t)}{n_j} [1 - O_p(1/\sqrt{n})] = O_p\left(\frac{1}{n}\right),$$

so

$$(S.2b) = 2\tilde{\Delta}_j(\bar{\Delta}_j - \tilde{\Delta}_j) \frac{\hat{\sigma}_j^2(t)}{n_j} + O_p\left(\frac{1}{\sqrt{n}}\right). \quad (S.7)$$

By (S.2), (S.5), and (S.7),

$$\begin{aligned} -2 \log \mathcal{R}(t) &= \sum_{j=1}^k \left\{ (\bar{\Delta}_j - \tilde{\Delta}_j)^2 \frac{\hat{\sigma}_j^2(t)}{n_j} + 2\tilde{\Delta}_j(\bar{\Delta}_j - \tilde{\Delta}_j) \frac{\hat{\sigma}_j^2(t)}{n_j} + O_p\left(\frac{1}{\sqrt{n}}\right) \right\} \\ &= \sum_{j=1}^k (\bar{\Delta}_j^2 - \tilde{\Delta}_j^2) \frac{\hat{\sigma}_j^2(t)}{n_j} + O_p\left(\frac{1}{\sqrt{n}}\right), \end{aligned}$$

which completes the proof.

S.1.2 Proof of (B.5)

For simplicity here we suppress dependence on t . The matrix \mathbf{T} in (B.3) is specified as

$$\mathbf{T} \equiv \begin{bmatrix} \hat{\theta}_1 + \hat{\theta}_2 & -\hat{\theta}_2 & 0 & \cdots & & 0 \\ -\hat{\theta}_2 & \hat{\theta}_2 + \hat{\theta}_3 & -\hat{\theta}_3 & 0 & \cdots & 0 \\ 0 & -\hat{\theta}_3 & \hat{\theta}_3 + \hat{\theta}_4 & -\hat{\theta}_4 & 0 & \cdots \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ \vdots & & & & & 0 \\ 0 & \cdots & & -\hat{\theta}_{k-2} & \hat{\theta}_{k-2} + \hat{\theta}_{k-1} & -\hat{\theta}_{k-1} \\ 0 & \cdots & & 0 & -\hat{\theta}_{k-1} & \hat{\theta}_{k-1} + \hat{\theta}_k \end{bmatrix}. \quad (\text{S.8})$$

From (B.3), we can solve for $\bar{\lambda}$ by inverting \mathbf{T} using readily available recursive formulas for the inverse of tridiagonal matrices (Usmani, 1994). Specifically, for a general $(k-1) \times (k-1)$ tridiagonal matrix

$$\begin{bmatrix} a_1 & b_1 & & & \\ c_1 & \ddots & \ddots & & \\ & \ddots & \ddots & b_{k-2} & \\ & & & c_{k-2} & a_{k-1} \end{bmatrix}, \quad (\text{S.9})$$

the (i,j) -th element of its inverse is given by

$$\tau_{ij} \equiv \begin{cases} (-1)^{i+j} b_i \cdots b_{j-1} \zeta_{j+1} \frac{\phi_{i-1}}{\phi_{k-1}}, & \text{if } i < j, \\ \zeta_{j+1} \frac{\phi_{i-1}}{\phi_{k-1}}, & \text{if } i = j, \\ (-1)^{i+j} c_j \cdots c_{i-1} \zeta_{i+1} \frac{\phi_{j-1}}{\phi_{k-1}}, & \text{if } i > j, \end{cases} \quad (\text{S.10})$$

where ϕ_j satisfies the recurrence relation $\phi_j = a_j \phi_{j-1} - c_{j-1} b_{j-1} \phi_{j-2}$ for $j = 1, \dots, k-1$, $\phi_0 = 1$, $\phi_{-1} = 0$, $\phi_1 = a_1$, ζ_j satisfies the recurrence relation $\zeta_j = a_j \zeta_{j+1} - b_j c_j \zeta_{j+2}$ for $j = k-2, \dots, 1$, $\zeta_k = 1$, and $\zeta_{k-1} = a_{k-1}$. In our case we can take advantage of the more specific structure of \mathbf{T} to obtain explicit expressions for ϕ_j and ζ_j , and hence for \mathbf{T}^{-1} . Indeed, it can be shown by induction that

$$\phi_{j-1} = \sum_{l=1}^j \prod_{g \in E_l^{1,j}} \hat{\theta}_g \quad (\text{S.11})$$

for $j \geq 2$, where $E_l^{i,j} = \{i, \dots, j\} \setminus \{l\}$ for $i < j$, and similarly

$$\zeta_{k-j} = \sum_{l=k-j}^k \prod_{g \in E_l^{k-j,k}} \hat{\theta}_g \quad (\text{S.12})$$

for $j = 1, \dots, k-1$ and $k \geq 2$. Next, multiplying both sides of (B.3) by the $k \times (k-1)$ matrix

$$\mathbf{D}_M \equiv \begin{bmatrix} M_1 & 0 & 0 & \cdots & & 0 \\ -M_2 & M_2 & 0 & 0 & \cdots & 0 \\ 0 & -M_3 & M_3 & 0 & 0 & \cdots & 0 \\ & \ddots & \ddots & \ddots & \ddots & & \\ & \vdots & & & & & 0 \\ 0 & \cdots & & 0 & -M_{k-1} & M_{k-1} \\ 0 & \cdots & & 0 & 0 & -M_k \end{bmatrix} \quad (\text{S.13})$$

and noting that $\hat{\psi} = \mathbf{D}_M^T \mathbf{A}$ under H_0 , we get (B.4). It remains to find an explicit expression for $\mathbf{D}_M \mathbf{T}^{-1} \mathbf{D}_M^T = \mathbf{M} [\tilde{\mathbf{D}} \mathbf{T}^{-1} \tilde{\mathbf{D}}^T] \mathbf{M}^T$, where $\mathbf{D}_M = \mathbf{M} \tilde{\mathbf{D}}$,

$$\tilde{\mathbf{D}} \equiv \begin{bmatrix} 1 & 0 & 0 & \cdots & & 0 \\ -1 & 1 & 0 & 0 & \cdots & 0 \\ 0 & -1 & 1 & 0 & 0 & \cdots & 0 \\ & \ddots & \ddots & \ddots & \ddots & & \\ & \vdots & & & & & 0 \\ 0 & \cdots & & 0 & -1 & 1 \\ 0 & \cdots & & 0 & 0 & -1 \end{bmatrix}_{k \times (k-1)}$$

and $\mathbf{M} = \text{diag}(M_1, \dots, M_k)$. Using the notation τ_{ij} given in (S.10),

$$\begin{aligned} [\tilde{\mathbf{D}}\mathbf{T}^{-1}\tilde{\mathbf{D}}^T]_{ij} &= \begin{cases} \tau_{i-1,j-1} - \tau_{i-1,j} - \tau_{i,j-1} + \tau_{ij} & i \neq 1, k \quad j \neq 1, k \\ -\tau_{1,j-1} + \tau_{1,j} & i = 1 \quad j \neq 1, k \\ \tau_{11} & i = 1 \quad j = 1 \\ -\tau_{1,k-1} & i = 1 \quad j = k \\ \tau_{k-1,j-1} - \tau_{k-1,j} & i = k \quad j \neq 1, k \\ -\tau_{k-1,1} & i = k \quad j = 1 \\ \tau_{k-1,k-1} & i = k \quad j = k. \end{cases} \\ &= \begin{cases} \frac{1}{\hat{\theta}_i \hat{\phi}} \sum_{l \neq i} \prod_{g \in E_l} \hat{\theta}_g, & i = j, \\ -\frac{1}{\hat{\theta}_i \hat{\phi}} \prod_{g \in E_j} \hat{\theta}_g, & i \neq j, \end{cases} \end{aligned} \quad (\text{S.14})$$

where the second equality will be shown in the next paragraph.

We just consider the case $i \neq 1, k$ and $j \neq 1, k$, the remaining cases being similar. We split the argument into three parts: $i = j$, $i < j$, and $i > j$. Since $\tilde{\mathbf{D}}\mathbf{T}^{-1}\tilde{\mathbf{D}}^T$ is symmetric, it suffices to examine $i < j$ and $i = j$. For $i < j$, using (S.10) and rearranging terms, we get

$$\tau_{i-1,j-1} - \tau_{i-1,j} - \tau_{i,j-1} + \tau_{ij} = \beta(-1)^{i+j} (b_{i-1}\phi_{i-2} + \phi_{i-1})(\zeta_j + b_{j-1}\zeta_{j+1})/\phi_{k-1}, \quad (\text{S.15})$$

where $\beta = b_i \cdots b_{j-2} = (-1)^{j-i-1} \hat{\theta}_{i+1} \cdots \hat{\theta}_{j-1}$ for $j \geq i+2$ and $\beta = 1$ for $j = i+1$. Surprisingly, the middle two terms of (S.15) reduce to the elegant and tractable expressions

$$b_{i-1}\phi_{i-2} + \phi_{i-1} = (-\hat{\theta}_i) \sum_{\ell=1}^{i-1} \prod_{g \in E_l^{1,i-1}} \hat{\theta}_g + \sum_{\ell=1}^i \prod_{g \in E_l^{1,i}} \hat{\theta}_g = \hat{\theta}_1 \cdots \hat{\theta}_{i-1} \quad (\text{S.16})$$

for $i \geq 2$, where we define $\prod_{g \in E_l^{i,i}} \hat{\theta}_g \equiv 1$ for all $i = 1, \dots, k$, and

$$\zeta_j + b_{j-1}\zeta_{j+1} = \sum_{\ell=j}^k \prod_{g \in E_l^{j,k}} \hat{\theta}_g + (-\hat{\theta}_j) \sum_{\ell=j+1}^k \prod_{g \in E_l^{j+1,k}} \hat{\theta}_g = \hat{\theta}_{j+1} \cdots \hat{\theta}_k \quad (\text{S.17})$$

for $j \leq k-1$. Plugging in (S.16) and (S.17) into (S.15), we get the second part of (S.14). For $i = j$, using (S.10) and rearranging the terms, we have

$$\tau_{i-1,j-1} - \tau_{i-1,j} - \tau_{i,j-1} + \tau_{ij} = \frac{\phi_{i-2}}{\phi_{k-1}} (\zeta_i + b_{i-1}\zeta_{i+1}) + \frac{\zeta_{i+1}}{\phi_{k-1}} (b_{i-1}\phi_{i-2} + \phi_{i-1}). \quad (\text{S.18})$$

Noting that (S.16) and (S.17) hold for $i = j$ as well, and plugging (S.11), (S.12), (S.16),

and (S.17) into (S.18) lead to the first part of (S.14).

Finally we pre- and post-multiply (S.14) by the diagonal matrix \mathbf{M} to arrive at (B.5).

S.1.3 Proofs of Lemmas 4 and 5

As in the previous section, we suppress dependence on t . The asymptotic equivalence assertion in Lemmas 4 is obtained by combining the terms in (B.7) and (B.9) for $j = 1, \dots, k$:

$$\begin{aligned} \sum_{j=1}^k (\bar{\Delta}_j^2 - \tilde{\Delta}_j^2) \frac{\hat{\sigma}_j^2}{n_j} &= \sum_{B \in \mathcal{B}} \sum_{j \in B} (\bar{\Delta}_j^2 - \tilde{\Delta}_j^2) \frac{\hat{\sigma}_j^2}{n_j} \\ &= \sum_{B \in \mathcal{B}} \sum_{j \in B} \left\{ w_j \left(\frac{\hat{U}_j}{\sqrt{w_j}} - \check{U} \right)^2 - w_j \left(\frac{\hat{U}_j}{\sqrt{w_j}} - \hat{U}_B \right)^2 \right\} + o_p(1) \\ &= \sum_{B \in \mathcal{B}} \sum_{j \in B} w_j \left\{ \hat{U}_B - \check{U} \right\}^2 + o_p(1). \end{aligned} \quad (\text{S.19})$$

The weak convergence part of Lemma 4 will follow from the discussion in Section 4.2, along with Lemma 5.

Proof of Lemma 5. For given j and block C (of sample indices) containing j , we denote $\bar{S}_C(t) = \prod_{i \leq N_j(t)} (1 - \check{h}_{ij})^{M_j}$, where \check{h}_{ij} is given in (2.8) for $B = C$. Note that \check{h}_{ij} takes the same form as (2.3), as explained in Section 2.3. Parallel to our previous notation $\bar{\Delta}_j$, specify $\check{\Delta}_j$ to satisfy $\check{h}_{ij} = d_{ij}/(r_{ij} + \check{\Delta}_j)$. Then, using a similar Taylor expansion as in (B.1), we have

$$\begin{aligned} Z(C) &\equiv \sqrt{n} \left\{ \log \bar{S}_C - M_j \log S_j \right\} \\ &= \sqrt{n} \left\{ M_j \log \hat{S}_j + M_j \check{\Delta}_j \hat{\sigma}_j^2 / n_j - M_j \log S_j \right\} + O_p(1/\sqrt{n}) \\ &= \sqrt{n} \left\{ M_j \log \hat{S}_j - M_j \log S_j \right\} + \sqrt{\theta_j} \left(-\hat{U}_j + \frac{1}{\phi_C} \sum_{l \in C} \prod_{g \in E_l^C} \theta_g \hat{U}_l \sqrt{\frac{\theta_l}{\theta_j}} \right) + o_p(1) \\ &= \sqrt{\theta_j w_j} \hat{U}_C + o_p(1), \end{aligned} \quad (\text{S.20})$$

where the second and the third equalities can be shown similarly to (B.8) and (B.9), except without squaring $\check{\Delta}_j$. We view $Z(C)$ as a function of C , with its dependence on the data and t being suppressed. In what follows we are going to apply (S.20) to $C = B(j) \in \mathcal{B}$ or $B'(j) \in \mathcal{B}'$.

We proceed with proving Lemma 5 by induction on j . For $j = 1$, define the event

$A_1 = \{B(1) \subseteq B'(1)\}$, and write

$$\hat{U}_{B(1)} - \hat{U}_{B'(1)} = \left\{ \hat{U}_{B(1)} - \hat{U}_{B'(1)} \right\} I(A_1) + \left\{ \hat{U}_{B(1)} - \hat{U}_{B'(1)} \right\} I(A_1^c), \quad (\text{S.21})$$

where $I(\cdot)$ is the indicator function. On A_1 , Lemmas S.1 and S.3 below imply $\bar{S}_{B(1)}(t) \geq \bar{S}_{B'(1)}(t)$. This can be seen by the *decomposition* $B'(1) = \cup_{j=1}^J B(j) \cup b(j+1)$ for some $j \geq 1$, where $b(j+1) \subseteq B(j+1)$ and noting that $\bar{S}_{b(j+1)} < \bar{S}_{B(j+1)}$ by Lemma S.1; further, by taking $L = j+1$, $C_j = B(j)$ for $j \leq J$, $C_L = b(j+1)$, and $l = 1$ in Lemma S.3. Then we have $Z(B(1)) \geq Z(B'(1))$ by the definition of $Z(\cdot)$. This and (S.20) then give

$$\left\{ \hat{U}_{B(1)} - \hat{U}_{B'(1)} \right\} I(A_1) = \{Z(B(1)) - Z(B'(1)) + o_p(1)\} I(A_1) / \sqrt{\theta_1 w_1} \geq o_p(1),$$

where the inequality also uses the fact that θ_1 and w_1 are bounded away from 0 on $[t_1, t_2]$. Furthermore, we have $\{\hat{U}_{B(1)} - \hat{U}_{B'(1)}\} I(A_1) \leq 0$ by Lemma S.2. These results lead to $\{\hat{U}_{B(1)} - \hat{U}_{B'(1)}\} I(A_1^c) = o_p(1)$. On A_1^c , we have $\bar{S}_{B(1)} > \bar{S}_{B'(1)}$ by Lemma S.1. Then as with A_1 , we have $\{\hat{U}_{B(1)} - \hat{U}_{B'(1)}\} I(A_1^c) \geq o_p(1)$. Furthermore, $\{\hat{U}_{B(1)} - \hat{U}_{B'(1)}\} I(A_1^c) < 0$ by Lemma S.2 and Lemma S.4 (by a similar *decomposition* argument as above), so $\{\hat{U}_{B(1)} - \hat{U}_{B'(1)}\} I(A_1^c) = o_p(1)$. Then by (S.21), we get $\hat{U}_{B(1)} - \hat{U}_{B'(1)} = o_p(1)$, and the conclusion of the lemma holds for $j = 1$.

Now suppose the conclusion of the lemma holds for $j = l$. Write

$$\hat{U}_{B(l+1)} - \hat{U}_{B'(l+1)} = \sum_{i=1}^4 \left\{ \hat{U}_{B(l+1)} - \hat{U}_{B'(l+1)} \right\} I(A_{l+1,i}), \quad (\text{S.22})$$

where we define the events $A_{l+1,1} = \{B'(l) = B'(l+1), B(l) = B(l+1)\}$, $A_{l+1,2} = \{B'(l) \text{ precedes } B'(l+1), B(l) = B(l+1)\}$, $A_{l+1,3} = \{B'(l) = B'(l+1), B(l) \text{ precedes } B(l+1)\}$ and $A_{l+1,4} = \{B'(l) \text{ precedes } B'(l+1), B(l) \text{ precedes } B(l+1)\}$. To complete the proof it suffices to show that each term in (S.22) is $o_p(1)$.

$A_{l+1,1}$: By step $j = l$, $\{\hat{U}_{B(l+1)} - \hat{U}_{B'(l+1)}\} I(A_{l+1,1}) = \{\hat{U}_{B(l)} - \hat{U}_{B'(l)}\} I(A_{l+1,1}) = o_p(1)$.

$A_{l+1,2}$: By $\hat{U}_{B'(l)}(t) \geq \hat{U}_{B'(l+1)}(t)$ and step $j = l$, $\{\hat{U}_{B(l+1)} - \hat{U}_{B'(l+1)}\} I(A_{l+1,2}) \geq \{\hat{U}_{B(l)} - \hat{U}_{B'(l)}\} I(A_{l+1,2}) = o_p(1)$. Next we show that $\{\hat{U}_{B(l+1)} - \hat{U}_{B'(l+1)}\} I(A_{l+1,2}) \leq o_p(1)$. The idea is to partition $A_{l+1,2}$ into

$$A'_{l+1,2} = A_{l+1,2} \cap \{\min(B'(l)) \leq \min(B(l))\}$$

and its relative complement $A''_{l+1,2} = A_{l+1,2} \setminus A'_{l+1,2}$. Here we illustrate the proof of $\{\hat{U}_{B(l+1)} - \hat{U}_{B'(l+1)}\} I(A'_{l+1,2}) \leq o_p(1)$; the proof of $\{\hat{U}_{B(l+1)} - \hat{U}_{B'(l+1)}\} I(A''_{l+1,2}) \leq o_p(1)$ is similar. On $A'_{l+1,2}$, partition $B(l) = J \cup K \cup L$, where $J = B'(l) \cap B(l)$, $K = B'(l+1) \cap B(l)$, and $L = B(l) \setminus (J \cup K)$. Note that $L = \emptyset$ when $\max(B'(l+1)) \leq \min(B(l))$.

$1)) \geq \max(B(l+1))$, so $\sum_{j \in L} w_j = \sum_{j \in \emptyset} w_j \equiv 0$ in the following display. Then

$$\begin{aligned}
& \left\{ \hat{U}_{B(l+1)} - \hat{U}_{B'(l+1)} \right\} I(A'_{l+1,2}) = \left\{ \hat{U}_{B(l)} - \hat{U}_{B'(l+1)} \right\} I(A'_{l+1,2}) \\
& \leq \left\{ \hat{U}_{B(l)} - \hat{U}_K \right\} I(A'_{l+1,2}) \\
& = \left(\hat{U}_{B(l)} - \left[\hat{U}_{B(l)} + \frac{\sum_{j \in J} w_j \left\{ \hat{U}_{B(l)} - \hat{U}_J \right\} + \sum_{j \in L} w_j \left\{ \hat{U}_{B(l)} - \hat{U}_L \right\}}{\sum_{j \in K} w_j} \right] \right) \times \\
& \quad I(A'_{l+1,2}) \\
& = \left[\frac{\sum_{j \in J} w_j \{ Z(J) - Z(B(l)) + o_p(1) \}}{\sqrt{\theta_l w_l} \sum_{j \in K} w_j} + \frac{\sum_{j \in L} w_j \left\{ \hat{U}_L - \hat{U}_{B'(l)} + o_p(1) \right\}}{\sum_{j \in K} w_j} \right] \times \\
& \quad I(A'_{l+1,2}) \\
& \leq 0 + o_p(1) + 0 + o_p(1) = o_p(1),
\end{aligned}$$

where the first inequality follows from Lemma S.2, the second equality from rearranging $\hat{U}_{B(l)} = (\sum_{j \in J} w_j \hat{U}_J + \sum_{j \in K} w_j \hat{U}_K + \sum_{j \in L} w_j \hat{U}_L) / \sum_{j \in B(l)} w_j$, the third equality from (S.20), $l \in J$ and step $j = l$, and the last inequality from $\bar{S}_J(t) < \bar{S}_{B(l)}(t)$ due to Lemma S.1 and $\hat{U}_L < \hat{U}_{B'(l)}$ due to Lemmas S.2 and S.4 (by a similar *decomposition* argument as above).

$A_{l+1,3}$: The proof can be done by following a similar argument as in $A_{l+1,2}$, but with the role of $B'(l)$ and $B'(l+1)$ switched with $B(l)$ and $B(l+1)$, and by using the fact that $\hat{U}_{B(j)} - \hat{U}_{B'(j)} = o_p(1)$ if and only if $Z(B(j)) - Z(B'(j)) = o_p(1)$ for $j = l, l+1$.

$A_{l+1,4}$: The proof is similar to the $j = 1$ case, but with $B(1)$ and $B'(1)$ replaced by $B(l+1)$ and $B'(l+1)$, respectively.

S.1.4 Properties of PAVA

This section collects some finite sample properties of PAVA that we appealed to in Supplement Section S.1.3. As in the previous section, we suppress dependence on t .

Lemma S.1. *Given a block $B \in \mathcal{B}$, where \mathcal{B} is determined by PAVA, any non-trivial partition $B = C_1 \cup C_2$ such that C_1 precedes C_2 satisfies $\bar{S}_{C_1} < \bar{S}_B < \bar{S}_{C_2}$.*

Proof. We use a similar argument as in proving that \bar{S}_B is strictly sandwiched between \bar{S}_{B_1} and \bar{S}_{B_2} for $B \in \mathcal{B}^{(q)}$ and $B_1, B_2 \in \mathcal{B}^{(q-1)}$ in Appendix A, except that here B belongs to the final set of blocks \mathcal{B} , and C_1, C_2 are not necessarily generated in any PAVA step. We will show that $\bar{S}_{C_1} < \bar{S}_B$; a similar argument can be used to show $\bar{S}_B < \bar{S}_{C_2}$. From the definitions of \bar{S}_{C_1} and \bar{S}_B , we have the following two strings of

equalities:

$$\bar{S}_{C_1} = a_\iota(M_\iota \lambda_\iota) = a_{\iota+1}(M_{\iota+1}(\lambda_{\iota+1} - \lambda_\iota)) = \cdots = a_j(-M_j \lambda_{j-1}),$$

and

$$\bar{S}_B = a_\iota(M_\iota \lambda'_\iota) = a_{\iota+1}(M_{\iota+1}(\lambda'_{\iota+1} - \lambda'_\iota)) = \cdots = a_j(M_j(\lambda'_j - \lambda'_{j-1})),$$

where λ_j and λ'_j are the Lagrange multipliers associated with \bar{S}_{C_1} and \bar{S}_B for $j \in C_1$, respectively, and j denotes the maximal element of C_1 ($\lambda_j \equiv 0$). We prove $\bar{S}_{C_1} < \bar{S}_B$ by contradiction. Suppose $\bar{S}_{C_1} \geq \bar{S}_B$. Then because $a_j(\cdot)$ is strictly increasing, a comparison between the two strings of equalities above leads to

$$\lambda'_\iota \leq \lambda_\iota, \tag{S.23}$$

$$\lambda'_{j+1} - \lambda'_j \leq \lambda_{j+1} - \lambda_j, \quad j = \iota, \dots, j-2, \tag{S.24}$$

$$\lambda'_j - \lambda'_{j-1} \leq -\lambda_{j-1}. \tag{S.25}$$

By (S.23) and (S.24), $\lambda'_j \leq \lambda_j$ for $j = \iota + 1, \dots, j-1$. This and (S.25) imply $\lambda'_j \leq 0$, which contradicts $\lambda'_j > 0$ shown in Appendix A. Thus we have $\bar{S}_{B_1} < \bar{S}_B$ instead. \square

Lemma S.2. *Given a block $B \in \mathcal{B}'$, where \mathcal{B}' is determined by $E_{\mathbf{w}}(\hat{\mathbf{U}}_{\mathbf{w}}(t)|\mathcal{I})$, any non-trivial partition $B = C_1 \cup C_2$ such that C_1 precedes C_2 satisfies $\hat{U}_{C_1} < \hat{U}_B < \hat{U}_{C_2}$.*

Proof. We use a similar argument to Lemma S.1. Recall that, by definition, the weighted least square projection $E_{\mathbf{w}}(\hat{\mathbf{U}}_{\mathbf{w}}|\mathcal{I})$ minimizes

$$\sum_{j=1}^k w_j \left(\frac{\hat{U}_j}{\sqrt{w_j}} - x_j \right)^2$$

over $(x_1, \dots, x_k) \in \mathbb{R}^k$, subject to the constraints $x_{j+1} - x_j \leq 0$, $j = 1, \dots, k-1$. By a similar argument as in Section 4.1, the optimal solution $(\tilde{x}_1, \dots, \tilde{x}_k)$ to the above minimization problem is unique and satisfies the KKT conditions. In particular, from the stationarity condition we have $\tilde{x}_j = \hat{U}_j/\sqrt{w_j} + (\tilde{\lambda}'_j - \tilde{\lambda}'_{j-1})/w_j$ for $j = 1, \dots, k$, where $\tilde{\lambda}'_1, \dots, \tilde{\lambda}'_{k-1}$ are the Lagrange multipliers and $\tilde{\lambda}'_0 \equiv \tilde{\lambda}'_k \equiv 0$. Then by a similar argument as in Section 2.3 and Appendix A, the solution can be obtained by a PAVA, where the pooling between adjacent blocks B_1 and B_2 involves minimizing $\sum_{j \in B_1 \cup B_2} w_j \left(\hat{U}_j/\sqrt{w_j} - x_j \right)^2$ subject to the equality constraints $x_j = x_l$ for all $j, l \in B_1 \cup B_2$. This version of PAVA results in $\hat{U}_B = \tilde{x}_j$, where j is the minimal element of the block $B \in \mathcal{B}'$. It can also be shown that $\tilde{\lambda}'_j > 0$, $j = 1, \dots, k-1$. The result then follows by replacing \bar{S}_B , \bar{S}_{C_1} , \bar{S}_{C_2} and $a_j(\Delta)$ in the proof of Lemma S.1 by \hat{U}_B , \hat{U}_{C_1} , \hat{U}_{C_2} and $\hat{U}_j/\sqrt{w_j} + \Delta/w_j$, respectively. \square

Lemma S.3. Let C_1, \dots, C_L be adjacent blocks that are arranged in increasing order (i.e., C_l precedes C_{l+1}) and satisfying $\bar{S}_{C_1} \geq \bar{S}_{C_2} \geq \dots \geq \bar{S}_{C_L}$, where $L \geq 2$. Then

$$\bar{S}_{C_1 \cup \dots \cup C_l} \geq \bar{S}_{C_1 \cup \dots \cup C_L} \geq \bar{S}_{C_{l+1} \cup \dots \cup C_L}, \quad l = 1, \dots, L-1.$$

Proof. To be clear, note that C_1, \dots, C_L need not be blocks resulting from PAVA. For future use, let j_i be the maximal element of C_i , and let λ_j be the Lagrange multipliers associated with $\bar{S}_{C_1 \cup \dots \cup C_L}$ as j ranges over $C_1 \cup \dots \cup C_L \setminus \{j_L\}$. We use induction on the number of blocks $L \geq 2$. When $L = 2$, the result will follow by similar reasoning as in Lemma S.1 and Appendix A. Again we compare the strings of equalities satisfied by \bar{S}_{C_1} , \bar{S}_{C_2} and $\bar{S}_{C_1 \cup C_2}$ in terms of $a_j(\cdot)$. We first show $\bar{S}_{C_1} \geq \bar{S}_{C_1 \cup C_2}$, arguing by contradiction. Suppose $\bar{S}_{C_1} < \bar{S}_{C_1 \cup C_2}$. Then, as $a_j(\cdot)$ is strictly increasing, a comparison of \bar{S}_{C_1} and $\bar{S}_{C_1 \cup C_2}$ leads to $\lambda_{j_1} > 0$. Also, $\bar{S}_{C_1} < \bar{S}_{C_1 \cup C_2}$ and the condition that $\bar{S}_{C_1} \geq \bar{S}_{C_2}$ lead to $\bar{S}_{C_2} < \bar{S}_{C_1 \cup C_2}$, so that a comparison of the strings of equalities satisfied by \bar{S}_{C_2} and $\bar{S}_{C_1 \cup C_2}$ leads to $\lambda_{j_1} < 0$. This contradicts $\lambda_{j_1} > 0$ obtained from $\bar{S}_{C_1} < \bar{S}_{C_1 \cup C_2}$. Thus we have $\bar{S}_{C_1} \geq \bar{S}_{C_1 \cup C_2}$, which gives $\lambda_{j_1} \leq 0$ by comparing the strings of equalities again. Then another contradiction would occur if $\bar{S}_{C_1 \cup C_2} < \bar{S}_{C_2}$, because it implies $\lambda_{j_1} > 0$. Therefore, we have that $\bar{S}_{C_1 \cup C_2} \geq \bar{S}_{C_2}$. So we have shown that the result for $L = 2$.

Now suppose the result holds for any number of blocks up to $L - 1$. We argue by contradiction. Suppose $\bar{S}_{C_1 \cup \dots \cup C_l} < \bar{S}_{C_1 \cup \dots \cup C_L}$ for some $l = 1, \dots, L - 1$. Then by the induction assumption, $\bar{S}_{C_1 \cup \dots \cup C_L} > \bar{S}_{C_1 \cup \dots \cup C_l} \geq \bar{S}_{C_1 \cup \dots \cup C_{L-1}} \geq \bar{S}_{C_{L-l} \cup \dots \cup C_L} \geq \bar{S}_{C_L}$. We can then use similar arguments as in the case $L = 2$ to show $\bar{S}_{C_1 \cup \dots \cup C_L} > \bar{S}_{C_1 \cup \dots \cup C_{L-1}}$, which leads to $\lambda_{j_{L-1}} > 0$, and also to show that $\bar{S}_{C_1 \cup \dots \cup C_L} > \bar{S}_{C_L}$, which contradicts $\lambda_{j_{L-1}} > 0$. This proves $\bar{S}_{C_1 \cup \dots \cup C_l} \geq \bar{S}_{C_1 \cup \dots \cup C_L}$. As for the second inequality, suppose $\bar{S}_{C_1 \cup \dots \cup C_L} < \bar{S}_{C_{l+1} \cup \dots \cup C_L}$. Then by the induction assumption, we have $\bar{S}_{C_1 \cup \dots \cup C_L} < \bar{S}_{C_{l+1} \cup \dots \cup C_L} \leq \bar{S}_{C_2 \cup \dots \cup C_L} \leq \bar{S}_{C_2} \leq \bar{S}_{C_1 \cup C_2} \leq \bar{S}_{C_1}$. We can then use similar arguments as in the case $L = 2$ to show $\bar{S}_{C_1 \cup \dots \cup C_L} < \bar{S}_{C_2 \cup \dots \cup C_L}$, which leads to $\lambda_{j_1} > 0$, and also to show that $\bar{S}_{C_1 \cup \dots \cup C_L} < \bar{S}_{C_1}$, which contradicts $\lambda_{j_1} > 0$. This establishes the second inequality and completes the proof. \square

The next lemma is parallel to Lemma S.3 with \bar{S}_C replaced by \hat{U}_C . The proof is much simpler, however, because an explicit expression is available for \hat{U}_C as a weighted average.

Lemma S.4. Let C_1, \dots, C_L be adjacent blocks that are arranged in increasing order (i.e., C_l precedes C_{l+1}) and satisfying $\hat{U}_{C_1} \geq \hat{U}_{C_2} \geq \dots \geq \hat{U}_{C_L}$, where $L \geq 2$. Then

$$\hat{U}_{C_1 \cup \dots \cup C_l} \geq \hat{U}_{C_1 \cup \dots \cup C_L} \geq \hat{U}_{C_{l+1} \cup \dots \cup C_L}, \quad l = 1, \dots, L-1.$$

Proof. This is a consequence of convexity and noting that

$$\hat{U}_{C_i \cup \dots \cup C_\ell} = \left(\hat{U}_{C_i} \sum_{j \in C_i} w_j + \hat{U}_{C_{i+1}} \sum_{j \in C_{i+1}} w_j + \dots + \hat{U}_{C_\ell} \sum_{j \in C_\ell} w_j \right) / \sum_{j \in C_i \cup \dots \cup C_\ell} w_j$$

is a convex combination of the \hat{U}_{C_j} , where $1 \leq i \leq \ell \leq L$. \square

S.2 Bounds on $\bar{\Delta}_j$ and $\tilde{\Delta}_j$

In this section we give bounds on $\bar{\Delta}_j$ uniformly over $t \in [t_1, t_2]$, under the conditions of Theorem 1, using the equality constraints (2.4). It can be shown using a similar argument that bounds of the same order hold for $\tilde{\Delta}_j$, since in our PAVA (see Section 2.3 and Appendix A), $\tilde{\Delta}_j$ also satisfies equality constraints, but instead within the PAVA block containing j . Again for simplicity we suppress the dependence on t .

The proof depends on the following two inequalities. When $\bar{\Delta}_j < 0$, we have

$$\begin{aligned} - \sum_{i \leq N_j(t)} \log \left(1 - \frac{d_{ij}}{r_{ij} + \bar{\Delta}_j} \right) &\geq \sum_{i \leq N_j(t)} \frac{d_{ij}}{r_{ij} + \bar{\Delta}_j} = \sum_{i \leq N_j(t)} \frac{d_{ij}}{r_{ij}} \frac{r_{ij}}{r_{ij} - |\bar{\Delta}_j|} \\ &\geq \sum_{i \leq N_j(t)} \frac{d_{ij}}{r_{ij}} \frac{n_j}{n_j - |\bar{\Delta}_j|} = \hat{A}_j(t) \frac{n_j}{n_j - |\bar{\Delta}_j|}, \end{aligned} \quad (\text{S.26})$$

where the first inequality follows from $0 \leq d_{ij}/(r_{ij} + \bar{\Delta}_j) < 1$ for all $i \leq N_j(t)$ and the fact that $\log(1 - x) + x \leq 0$ for $x \in [0, 1]$. Here $\hat{A}_j(t) \equiv \sum_{i \leq N_j(t)} d_{ij}/r_{ij}$ is the Nelson–Aalen estimator of the cumulative hazard function. Similarly, when $\bar{\Delta}_j \geq 0$, we have

$$\begin{aligned} \sum_{i \leq N_j(t)} \log \left(1 - \frac{d_{ij}}{r_{ij} + \bar{\Delta}_j} \right) &\geq \sum_{i \leq N_j(t)} \left[-\frac{d_{ij}}{r_{ij} + \bar{\Delta}_j} + \log \left(1 - \frac{d_{ij}}{r_{ij}} \right) + \frac{d_{ij}}{r_{ij}} \right] \\ &= - \sum_{i \leq N_j(t)} \frac{d_{ij}}{r_{ij}} \frac{r_{ij}}{r_{ij} + |\bar{\Delta}_j|} + \log \hat{S}_j(t) + \hat{A}_j(t) \\ &\geq - \sum_{i \leq N_j(t)} \frac{d_{ij}}{r_{ij}} \frac{n_j}{n_j + |\bar{\Delta}_j|} + \log \hat{S}_j(t) + \hat{A}_j(t) \\ &= -\hat{A}_j(t) \frac{n_j}{n_j + |\bar{\Delta}_j|} + \log \hat{S}_j(t) + \hat{A}_j(t), \end{aligned} \quad (\text{S.27})$$

where the first inequality follows from the fact that $\log(1 - x) + x$ is decreasing in $x \in [0, 1]$, $0 \leq d_{ij}/(r_{ij} + \bar{\Delta}_j) \leq d_{ij}/r_{ij} \leq 1$ for $i \leq N_j(t)$, and

$$\log \left(1 - \frac{d_{ij}}{r_{ij} + \bar{\Delta}_j} \right) + \frac{d_{ij}}{r_{ij} + \bar{\Delta}_j} \geq \log \left(1 - \frac{d_{ij}}{r_{ij}} \right) + \frac{d_{ij}}{r_{ij}}$$

for all $i \leq N_j(t)$.

We now show $\bar{\Delta}_j = O_p(\sqrt{n})$. The key is to obtain bounds based on (S.26), (S.27) and the estimating equations in (2.4), and to utilize well-known asymptotic properties of $\hat{A}_j(t)$ and $\hat{S}_j(t)$. First note that if $M_j \log \hat{S}_j(t) = M_{j'} \log \hat{S}_{j'}(t)$ for all j, j' , then $\bar{\Delta}_i = 0$ for all $i = 1, \dots, k$, which is trivially $O_p(\sqrt{n})$. On the complementary event, for a fixed j there exists $j' \neq j$ such that $M_j \log \hat{S}_j(t) \neq M_{j'} \log \hat{S}_{j'}(t)$. Then by the fact that $a_j(\Delta)$ and $a_{j'}(\Delta)$ are increasing in Δ , $\bar{\Delta}_j$ and $\bar{\Delta}_{j'}$ must have different signs. This can be understood as follows: since r_{ij} and $r_{ij'}$ are numbers of subjects at risk and $r_{ij} + \bar{\Delta}_j$ and $r_{ij'} + \bar{\Delta}_{j'}$ are adjustments to those numbers, our method essentially fine-tunes the at-risk sets to ensure the equality constraints, so there must be a gain in the size of the risk set in one sample and a loss in another. Without loss of generality suppose $\bar{\Delta}_j \geq 0, \bar{\Delta}_{j'} < 0$.

Applying (S.27) to $\log a_j(\bar{\Delta}_j)$ and (S.26) to $\log a_{j'}(\bar{\Delta}_{j'})$, we have

$$\begin{aligned} 0 &= \log a_j(\bar{\Delta}_j) - \log a_{j'}(\bar{\Delta}_{j'}) \\ &\geq M_{j'} \hat{A}_{j'}(t) \frac{n_{j'}}{n_{j'} - |\bar{\Delta}_{j'}|} - M_j \hat{A}_j(t) \frac{n_j}{n_j + |\bar{\Delta}_j|} + M_j \log \hat{S}_j(t) + M_j \hat{A}_j(t), \end{aligned}$$

where the first equality comes from (2.4). Note that $n_{j'} - |\bar{\Delta}_{j'}| = n_{j'} + \bar{\Delta}_{j'} \geq r_{ij'} + \bar{\Delta}_{j'} \geq 0$ because $\bar{h}_{i,j'} \geq 0$. Thus we can multiply both sides of the above display by $(n_{j'} - |\bar{\Delta}_{j'}|)(n_j + |\bar{\Delta}_j|) \geq 0$ without changing direction of the inequality to get

$$\begin{aligned} 0 &\geq M_{j'} \hat{A}_{j'}(t) n_{j'} (n_j + |\bar{\Delta}_j|) - M_j \hat{A}_j(t) n_j (n_{j'} - |\bar{\Delta}_{j'}|) \\ &\quad + M_j [\log \hat{S}_j(t) + \hat{A}_j(t)] (n_{j'} - |\bar{\Delta}_{j'}|) (n_j + |\bar{\Delta}_j|) \\ &= |\bar{\Delta}_j| |\bar{\Delta}_{j'}| [-M_j \log \hat{S}_j(t) - M_j \hat{A}_j(t)] + |\bar{\Delta}_{j'}| (-M_j \log \hat{S}_j(t) n_j) + \gamma_1 |\bar{\Delta}_j| + \gamma_2, \end{aligned} \tag{S.28}$$

where $\gamma_1 = M_{j'} n_{j'} \hat{A}_{j'}(t) + M_j n_{j'} (\log \hat{S}_j(t) + \hat{A}_j(t))$ and $\gamma_2 = n_{j'} n_j [M_{j'} \hat{A}_{j'}(t) + M_j \log \hat{S}_j(t)]$. Note that $-\log \hat{S}_j(t) - \hat{A}_j(t) > 0$ because

$$-\sum_{i \leq N_j(t)} \log \left(1 - \frac{d_{ij}}{r_{ij}}\right) > \sum_{i \leq N_j(t)} \frac{d_{ij}}{r_{ij}} \tag{S.29}$$

by $\log(1-x) + x < 0$ for all $x \in (0, 1]$. This, $M_j > 0, -\log \hat{S}_j(t) n_j \geq 0$ and (S.28) imply $\gamma_1 |\bar{\Delta}_j| + \gamma_2 \leq 0$. So the desired bound on $\bar{\Delta}_j$ can be obtained from bounds on γ_1 and γ_2 . As $M_j \log S_j(t) = M_{j'} \log S_{j'}(t)$ under H_0^t , we get that for any $\varepsilon > 0$,

$$\begin{aligned} \gamma_1 &= n_{j'} \left\{ M_{j'} [\hat{A}_{j'}(t) + \log S_{j'}(t)] + M_j [\log \hat{S}_j(t) - \log S_j(t)] + M_j \hat{A}_j(t) \right\} \\ &\geq \frac{n_{j'} M_j}{4} A_j(t_1) \end{aligned} \tag{S.30}$$

with probability at least $1 - \varepsilon$ and n sufficiently large, and that

$$\begin{aligned} -\gamma_2/n_{j'} &= n_j \left(M_{j'} \hat{A}_{j'}(t) + M_{j'} \log \hat{S}_{j'}(t) - M_{j'} \log S_{j'}(t) + M_j \log S_j(t) \right) \\ &= O_p(n)O_p(1/\sqrt{n}) = O_p(n^{1/2}), \end{aligned} \quad (\text{S.31})$$

by the fact that $\log \hat{S}_j - \log S_j = O_p(1/\sqrt{n})$, $\hat{A}_{j'} + \log S_{j'} = O_p(1/\sqrt{n})$ and $\hat{A}_j - A_j = O_p(1/\sqrt{n})$ using classical results on KM and Nelson–Aalen estimators (cf. Li, 1995). Since $A_j(t_1) > 0$ by the $S_0(t_1) < 1$ condition assumed in Theorem 1, (S.30) implies

$$0 < \frac{n_{j'} M_j}{4} A_j(t_1) |\bar{\Delta}_j| \leq \gamma_1 |\bar{\Delta}_j| \leq -\gamma_2$$

with probability at least $1 - \varepsilon$ and n sufficiently large, and thus by (S.31),

$$|\bar{\Delta}_j| \leq O_p(n^{1/2}). \quad (\text{S.32})$$

As for asymptotic order of $\bar{\Delta}_{j'}$, again by (S.29), we have $-M_j \log \hat{S}_j(t) n_j |\bar{\Delta}_{j'}| + \gamma_1 |\bar{\Delta}_j| + \gamma_2 \leq 0$. Since $\log \hat{S}_j - \log S_j = O_p(1/\sqrt{n})$ and the conditions in Theorem 1 imply $-\log \hat{S}_j(t) \geq -1/2 \log S_j(t_1) > 0$ with probability at least $1 - \varepsilon$ and n sufficiently large, (S.30), (S.31) and (S.32) then imply $|\bar{\Delta}_{j'}| \leq O_p(n^{1/2})$. This completes the proof.

S.3 Remarks on Theorem 1 continued

Often likelihood ratio statistics are distribution-free. Here we explain what makes this problem different from the classical setting and why we cannot obtain an asymptotically distribution-free form. The short answer is that we have $k > 2$ groups, censoring and $M_j \neq 1$. To be more specific, we will contrast the derivations of a distribution-free form in the $k = 2$ case with $k = 3$, along with the censored and $M_j \neq 1$ case with the uncensored and $M_j = 1$ case. For simplicity, we will illustrate using the omnibus test (i.e., without order restrictions) in Section 3.4, and suppress dependence on t . When $k = 2$, SSB^o in Theorem 2 can be written as

$$\{\sqrt{w_2}U_1 + \sqrt{w_1}U_2\}^2 = \frac{1}{\theta_1 + \theta_2} U_{1,2}^2,$$

where

$$U_{1,2} = \frac{\sigma_1 U_1 M_1}{\sqrt{p_1}} - \frac{\sigma_2 U_2 M_2}{\sqrt{p_2}}.$$

It can be shown that the covariance function of $U_{1,2}/\sqrt{\theta_1 + \theta_2}$ is the standardized form (i.e., it reduces to 1 for U_j at a fixed t)

$$(\theta_1 + \theta_2)(\min(s, t))/\sqrt{(\theta_1 + \theta_2)(s)(\theta_1 + \theta_2)}.$$

Such a standardized form enables us to find a suitable transformation in the time scale so that SSB^o becomes a functional of a standard Brownian bridge (see, e.g. Chang and McKeague, 2016, p.22 for the $M_1 = M_2 = 1$ case), which can lead to a distribution-free statistic.

On the other hand, when $k = 3$, SSB^o can be written as

$$\begin{aligned} & \{\sqrt{w_2}U_1 + \sqrt{w_1}U_2\}^2 + \{\sqrt{w_3}U_1 + \sqrt{w_1}U_3\}^2 \\ & + \{\sqrt{w_3}U_2 + \sqrt{w_2}U_3\}^2 \\ & = \frac{\theta_3}{\theta_1\theta_2 + \theta_1\theta_3 + \theta_2\theta_3} U_{1,2}^2 + \frac{\theta_2}{\theta_1\theta_2 + \theta_1\theta_3 + \theta_2\theta_3} U_{1,3}^2 \\ & + \frac{\theta_1}{\theta_1\theta_2 + \theta_1\theta_3 + \theta_2\theta_3} U_{2,3}^2, \end{aligned}$$

where $U_{1,3}$ and $U_{2,3}$ are defined similarly as $U_{1,2}$. Since we have shown in $k = 2$ that $U_{1,2}/\sqrt{\theta_1 + \theta_2}$ has the standardized form, the processes at hand

$$\begin{aligned} & \sqrt{\theta_l}U_{i,j}/\sqrt{\theta_1\theta_2 + \theta_1\theta_3 + \theta_2\theta_3} \\ & = \sqrt{\theta_l(\theta_i + \theta_j)}/\sqrt{\theta_1\theta_2 + \theta_1\theta_3 + \theta_2\theta_3} \times U_{i,j}/\sqrt{\theta_i + \theta_j} \end{aligned}$$

for distinct $i, j, l \in \{1, 2, 3\}$ are clearly not in the standardized form in general, due to the fact that $\sqrt{\theta_l(\theta_i + \theta_j)}/\sqrt{\theta_1\theta_2 + \theta_1\theta_3 + \theta_2\theta_3}$ may not be distribution-free. However, when $M_j = 1$ and there is no censoring, it can be shown that

$$\begin{aligned} & \frac{\sqrt{\theta_l(\theta_i + \theta_j)}}{\sqrt{\theta_1\theta_2 + \theta_1\theta_3 + \theta_2\theta_3}} \\ & = \frac{\sqrt{1/p_l(1/p_i + 1/p_j)}}{\sqrt{1/(p_1p_2) + 1/(p_2p_3) + 1/(p_1p_3)}}, \end{aligned}$$

which is distribution-free, as required.

S.4 Retention-of-effect approach

As in Section 3.3, it suffices to consider the local NPLR with equality constraint in the numerator:

$$\mathcal{R}^r(t) = \frac{\sup \{L(S_1, S_2, S_3) : S_2(t)/S_1(t) = \{S_3(t)/S_1(t)\}^M\}}{\sup \{L(S_1, S_2, S_3) : S_2(t)/S_1(t) \geq \{S_3(t)/S_1(t)\}^M\}}. \quad (\text{S.33})$$

To avoid excessive notation, in what follows we use the same notation \bar{h}_{ij} and \tilde{h}_{ij} as in Section 2.2. By the method of Lagrange multipliers, the numerator of (S.33) is the

constrained maximum $\prod_{j=1}^3 \prod_{i=1}^{m_j} \bar{h}_{ij}^{d_{ij}} (1 - \bar{h}_{ij}^{d_{ij}})$, where

$$\bar{h}_{i1} = \frac{d_{i1}}{r_{i1} + \bar{\lambda}(M-1)}, \quad \bar{h}_{i2} = \frac{d_{i2}}{r_{i2} + \bar{\lambda}}, \quad \bar{h}_{i3} = \frac{d_{i3}}{r_{i3} - M\bar{\lambda}},$$

for $i \leq N_j(t)$, $\bar{h}_{ij} = d_{ij}/r_{ij}$ for $i > N_j(t)$, and the multiplier $\bar{\lambda}$ satisfies the equality constraint

$$\prod_{i \leq N_2(t)} (1 - \bar{h}_{i,2}) / \prod_{i \leq N_1(t)} (1 - \bar{h}_{i1}) = \left\{ \prod_{i \leq N_3(t)} (1 - \bar{h}_{i,3}) / \prod_{i \leq N_1(t)} (1 - \bar{h}_{i1}) \right\}^M.$$

On the other hand, by the KKT method, the denominator of (S.33) is $\prod_{j=1}^k \prod_{i=1}^{m_j} \tilde{h}_{ij}^{d_{ij}} (1 - \tilde{h}_{ij}^{d_{ij}})$, where

$$\tilde{h}_{i1} = \frac{d_{i1}}{r_{i1} + \tilde{\lambda}(M-1)}, \quad \tilde{h}_{i2} = \frac{d_{i2}}{r_{i2} + \tilde{\lambda}}, \quad \tilde{h}_{i3} = \frac{d_{i3}}{r_{i3} - M\tilde{\lambda}},$$

for $i \leq N_j(t)$, $\tilde{h}_{ij} = d_{ij}/r_{ij}$ for $i > N_j(t)$, and the multiplier $\tilde{\lambda}$ satisfies the conditions

$$\begin{aligned} \prod_{i \leq N_2(t)} (1 - \tilde{h}_{i,2}) / \prod_{i \leq N_1(t)} (1 - \tilde{h}_{i1}) &\leq \left\{ \prod_{i \leq N_3(t)} (1 - \tilde{h}_{i,3}) / \prod_{i \leq N_1(t)} (1 - \tilde{h}_{i1}) \right\}^M, \\ \tilde{\lambda} &\geq 0, \\ \tilde{\lambda} \left[\prod_{i \leq N_2(t)} (1 - \tilde{h}_{i,2}) / \prod_{i \leq N_1(t)} (1 - \tilde{h}_{i1}) - \left\{ \prod_{i \leq N_3(t)} (1 - \tilde{h}_{i,3}) / \prod_{i \leq N_1(t)} (1 - \tilde{h}_{i1}) \right\}^M \right] &= 0. \end{aligned}$$

As in Section 2.2, the Lagrange multipliers adjust the size of the risk set in order to satisfy the equality and inequality constraints. However, now the problem becomes more complicated because each multiplier governs the adjustment between the risk sets of all the samples, whereas previously, each multiplier only governs the adjustment between the risk sets of two adjacent samples. This leads to a slightly more complicated integrated local NPLR statistic: it takes the same form as (3.1), but with $\hat{S}_0(t)$ as a consistent estimate of the survival function $S_0(t) = v_1(t)S_3^M(t) + v_2(t)S_1^{M-1}(t)S_2(t)$ instead. The limiting distribution now involves a projection onto $\mathcal{I} = \{\mathbf{z} \in \mathbb{R}^3 : z_1 = z_2 \geq z_3\}$ and a more complicated form of the weight functions:

$$\frac{\theta_1(t)\theta_3(t)}{\phi(t)(\theta_1(t) + \theta_2(t))}, \quad \frac{\theta_2(t)\theta_3(t)}{\phi(t)(\theta_1(t) + \theta_2(t))}, \quad \frac{\theta_1(t) + \theta_2(t)}{\phi(t)},$$

respectively, where $\theta_j(t) = M_j^2 \sigma_j^2(t)/p_j$, $M_1 = M-1$, $M_2 = 1$, $M_3 = M$, and $\phi(t) = \sum_{j=1}^3 \theta_j(t)$. With the above changes, the conclusion of Theorem 1 holds under the assumptions $S_3^M(t) = S_1(t)^{M-1}S_2(t)$ for $t \in [t_1, t_2]$, $S_0(t_1) < 1$ and $S_0(t_2)G_j(t_2) > 0$. A

similar bootstrap method as Section 3.1 can be used to calibrate this test.

S.5 Additional simulation results

S.5.1 Accuracy and power

Here we report results for censoring rate 50%, and unequal group sample sizes $n_1 = 135$, $n_2 = 270$, $n_3 = 260$ that roughly match the real data example. Furthermore, we consider an additional Scenario A' under the overall null H_1^c (see Figure S.1), where the standard treatment is superior to the placebo and the experimental treatment is inferior to reference, with the largest difference occurring at the margin (i.e., when $S_2^{M_2} = S_3^{M_3}$). The conclusions for Scenarios A, B and C are similar to those of Sections 6.1 and 6.2, see Table S.1. As for Scenario A', the empirical rejection rates are close to their nominal levels, except for the cases of $H_1^{S,R}$ and H_1^1 where the rejection rates represent empirical power because the standard treatment is superior to the placebo.

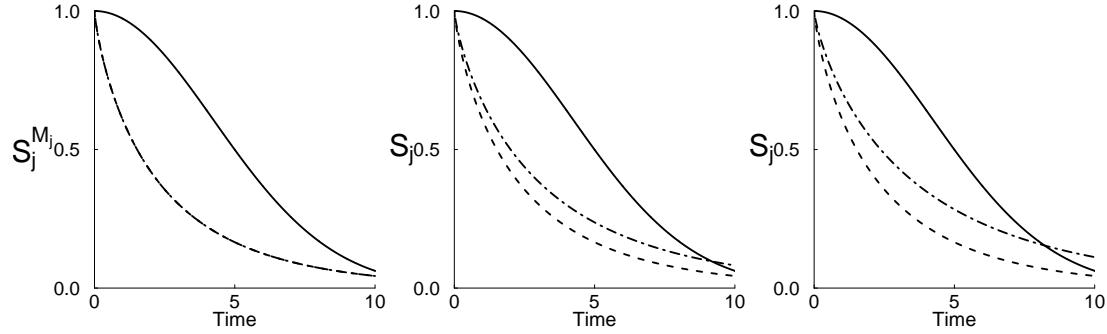


Figure S.1: Survival functions $S_j^{M_j}$ (left) and S_j (middle and right) for Scenario A' representing an example of H_1^c . Each $S_j^{M_j}$ is specified as Weibull: placebo (solid), standard therapy (dashed), experimental therapy (two-dashed). The S_j are defined from the $S_j^{M_j}$, and $(M_1, M_2, M_3) = (1, 1, 10/8)$ (middle) or $(M_1, M_2, M_3) = (1, 1, 10/7)$ (right).

S.5.2 Power/sample size calculations and optimal allocation

It is often of interest to carry out power calculations, determine sample sizes and find optimal allocation ratios prior to a clinical trial. Since our method is a two-step non-parametric testing procedure with bootstrap calibration, analytical formulas for asymptotic power would be very challenging to develop. Instead, we suggest a Monte Carlo simulation-based approach, as is often recommended (see, e.g., [Landau and Stahl, 2013](#)). Here we provide an example of such an analysis. Power versus total sample size n given a specific allocation ratio is plotted in Figure S.2. From the figure, we can see that for

Table S.1: Empirical significance levels and powers, at $\alpha = 0.05$ for the first set of H_1 columns (as in Tables 1 and 3) and at $\alpha = 0.025$ for all the other columns (as in Tables 2 and 4). Results based on 1000 replications, censoring rate 50%, and unequal allocation $n_1 = 135$, $n_2 = 270$, $n_3 = 260$. Scenario A' is indicated in Figure S.1.

(M_1, M_2, M_3)	H_1				$H_1^{S,R}$	H_1^1	H_1^N	H_1^2	$H_1^{S,R,N}$	H_1
	K_n	I_n	Cox	PW	Cox	PW	Cox	PW	Cox	PW
(1, 1, 10/8)	0.066	0.053	0	0	0.025	0.031	0.023	0.026	0	0
A (1, 1, 10/7)	0.072	0.054	0	0	0.027	0.032	0.022	0.028	0	0
(1, 1, 10/8)	0.040	0.040	0.055	0.068	1	1	0.026	0.032	0.026	0.032
A' (1, 1, 10/7)	0.053	0.053	0.052	0.059	1	1	0.026	0.028	0.026	0.028
(1, 1, 10/8)	0.909	0.929	0.363	0.334	0.392	0.370	0.710	0.656	0.222	0.192
B (1, 1, 10/7)	0.886	0.919	0.360	0.325	0.390	0.364	0.697	0.642	0.218	0.187
(1, 1, 10/8)	0.988	0.988	0.492	0.808	0.450	0.762	0.819	0.954	0.319	0.719
C (1, 1, 10/7)	0.989	0.990	0.476	0.802	0.454	0.756	0.778	0.934	0.301	0.695

equal allocation $p_1 = p_2 = p_3$, the required total sample size to achieve 80% power is $n = 120$. Also, in Table S.2 we tabulate power for various allocation ratios (for $n = 120$). Over the grid of (p_1, p_2) values in the table (note $p_3 = 1 - p_1 - p_2$), the optimal allocation is $(p_1, p_2, p_3) = (0.4, 0.1, 0.5)$. A finer grid would sometimes be more appropriate. We have provided the R code (see Supplement Section S.6) for these power calculations.

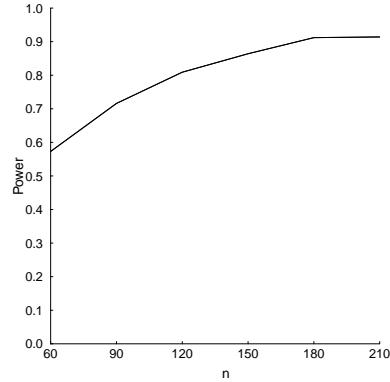


Figure S.2: Power versus total sample size (n) given equal allocation. Results based on 1000 replications, censoring rate 10%, $(M_1, M_2, M_3) = (1, 1, 10/7)$ in Scenario C.

Table S.2: Empirical powers at $\alpha = 0.05$ for different allocation ratios p_1 and p_2 ($p_3 = 1 - p_1 - p_2$). Results based on 1000 replications, censoring rate 10%, $(M_1, M_2, M_3) = (1, 1, 10/7)$ and $n = 120$ in Scenario C.

p_1	p_2							
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
0.1	0.700	0.720	0.709	0.700	0.651	0.605	0.496	0.395
0.2	0.856	0.830	0.809	0.755	0.698	0.595	0.496	
0.3	0.897	0.873	0.817	0.760	0.684	0.564		
0.4	0.901	0.864	0.814	0.740	0.632			
0.5	0.896	0.836	0.780	0.653				
0.6	0.848	0.778	0.664					
0.7	0.765	0.682						
0.8	0.637							

S.6 R code

```

1 ##### required packages
2 library(survival)
3 library(nloptr)
4 library(plyr) #array to list
5 library(Iso)
6 # for cluster computing:
7 library(parallel)
8 library(foreach)
9 library(doParallel)
10 ##### END required packages
11
12 ##### user input
13 # where R code containing the functions is
14 dir_path = paste("C:\\\\Users\\\\", Sys.getenv("USERNAME"), "\\\\Dropbox\\\\papers
15 _mine\\\\paper_ELSOC_k\\\\AOS1686", sep = "")
16 # where the results will be saved
17 dir_path2 = paste("C:\\\\Users\\\\", Sys.getenv("USERNAME"), "\\\\Dropbox\\\\
18 computing\\\\R_program\\\\out_test_20171106\\\\new", sep = "")
19 # source R code containing the functions
20 setwd(dir_path)
21 source("functions_to_be_sourced.R")
22 # set parameters for generating data:
23 n = 120
24 p1 = 0.4
25 p2 = 0.1
26 parameters = list() # a list for storing parameter values below
27 # calculate the number of cores for parallel computing
28 no_cores = detectCores()/2 - 1
29 # number of parallel tasks for computing nrep (see below) results
30 parameters$nsplit = no_cores

```

```

29 # number of datasets to be generated; should be a factor of nsplit above
30 parameters$nrep = 40 * no_cores
31 # number of datasets to be generated per parallel task
32 parameters$nrep_sub = ceiling(parameters$nrep / parameters$nsplit)
33 # number of bootstrap samples
34 parameters$nboot = 1000
35 # first entry is overall alpha level; second entry is alpha level after
   Bonferroni adjustment
36 parameters$alpha_vec = c(0.05, 0.025)
37 # the vector of margins M_1, M_2, M_3
38 parameters$M_vec = c(1, 1, 1 / 0.7)
39 # administrative censoring
40 parameters$adminC = 10
41 # ajbj_result: c(shape,scale) of Weibull for S_j^{M_j}, j=1,2,3
42 parameters$a1b1_result = c(2, 6)
43 parameters$a2b2_result = c(1.6, 5.2)
44 parameters$a3b3_result = c(1.2, 4.1)
45 # censoring rates
46 parameters$pct_censor = 0.10
47 # c(tau1,tau2): follow-up period [tau1,tau2] specified in study protocol,
   if any
48 parameters$tau1 = 0
49 parameters$tau2 = Inf
50 # starting seed for each replication of data generation
51 parameters$seedstart = 0
52 # specifying which test to perform power calculations on; current choices:
   c('two_step_sup', 'two_step_dF')
53 parameters$option_test = 'two_step_dF'
54 # specified_power: for computing power given sample sizes, set to 0. For
   computing optimal
55 # sample sizes given power, set this to the desired power.
56 parameters$specified_power = 0.2
57 ##### END user input
58
59 ##### other parameter calculations based on user input
60 a1b1 = c(parameters$a1b1_result[1], parameters$a1b1_result[2] * (
   parameters$M_vec[1] ^ (1 / parameters$a1b1_result[1])))
61 a2b2 = c(parameters$a2b2_result[1], parameters$a2b2_result[2] * (
   parameters$M_vec[2] ^ (1 / parameters$a2b2_result[1])))
62 a3b3 = c(parameters$a3b3_result[1], parameters$a3b3_result[2] * (
   parameters$M_vec[3] ^ (1 / parameters$a3b3_result[1])))
63 M_ret = parameters$M_vec[3]
64 M_ret_vec = c(M_ret - 1, 1, M_ret)
65
66 # find censoring parameters based on parameters$pct_censor given above (in
   user input)
67 prob_censor = function(c, ab, perc_censor) {

```

```

68 # Computes censoring probability - perc_censor for a given number perc_
69 # censor
70 # Args:
71 #   c: parameter for U(0,c) which generates the censoring random variable
72 #     with administrative censoring at time parameters$adminC
73 #   ab: parameters for Weibull(ab[1], ab[2]) which generates the lifetime
74 #     random variables
75 #   perc_censor: the given number perc_censor
76 # Returns:
77 #   the censoring probability - perc_censor for the given number perc_
78 #     censor
79 x = seq(0, c , by = 0.0001)
80 q = (1 - pweibull(x, shape = ab[1], scale = ab[2])) * (x < min(c,
81   parameters$adminC)) * 0.0001 / c + (1 - pweibull(x, shape = ab[1],
82   scale = ab[2])) * (x <= c & x == parameters$adminC) * (1 - min(c,
83   parameters$adminC) / c)
84 return(sum(q) - perc_censor)
85 } # END prob_censor
86 parameters$c = uniroot(prob_censor, interval = c(1.1, 700), ab = a1b1,
87   perc_censor = parameters$pct_censor)$root
88 ##### END other parameter calculations based on user input
89
90 ##### producing p-value and decision results, given a specific
91 # dataset
92 #### making an artificial dataset:
93 Yd = matrix(c(1, 2, 3, 4, 5, 6, 1.1, 2, 2.5, 5, 6, 7, 0.9, 1.5, 2.7, 4.1,
94   5.2, 6.5,
95   1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1), ncol = 2)
96 group = c(rep(1, 3), rep(2, 3), rep(3, 3))
97 data = cbind(Yd, group)
98 dat = Surv(data[, 1], data[, 2])
99 fit = survfit(dat ~ 1)
100 fit1 = survfit(dat[data[, 3] == 1] ~ 1)
101 fit2 = survfit(dat[data[, 3] == 2] ~ 1)
102 fit3 = survfit(dat[data[, 3] == 3] ~ 1)
103 #### implement the function computing p-values and decision results
104 teststatfn = teststat(parameters$tau1, parameters$tau2, fit, fit1, fit2,
105   fit3, parameters$M_vec, M_ret, data, parameters$alpha_vec, parameters$,
106   nboot)
107 #### outputs from the function:
108 teststatfn$out_dF_pval # p-value for I_n
109 teststatfn$out_sup_pval # p-value for K_n
110 teststatfn$out_dF_12_pval # p-value for the pairwise NPLR test comparing
111   group 1 and 2
112 teststatfn$out_dF_23_pval # p-value for the pairwise NPLR test comparing
113   group 2 and 3
114 teststatfn$out_dF_ret_pval # p-value for the pairwise NPLR test comparing
115   group 2 and 3 using retention-of-effect formulation

```

```

101 teststatfn$test_nocross # decision on whether  $H_{\{01\}}^c$  should be rejected
102     (1) or not (0)
103 teststatfn$test #  $K_n$ 
104 teststatfn$EL_SOcrit # critical value for  $K_n$  (corresponding to alpha_vec
105     levels)
106 teststatfn$inttest_dF #  $I_n$ 
107 teststatfn$int_dFEL_SOcrit # critical value for  $I_n$  (corresponding to
108     alpha_vec levels)
109 ##### use outputs from the function to implement the proposed two-step
110     procedure:
111 test_SO_sup = 0
112 test_SO_dF = 0
113 if (teststatfn$test_nocross == 1) {
114     if (teststatfn$test > teststatfn$EL_SOcrit[1]) {
115         test_SO_sup = 1
116     } # END if
117     if (teststatfn$inttest_dF > teststatfn$int_dFEL_SOcrit[1]) {
118         test_SO_dF = 1
119     } # END if
120 } else {
121     test_SO_sup = -1
122     test_SO_dF = -1
123 } # END if/else
124 ##### results of the proposed two-step procedure
125 test_SO_sup
126 test_SO_dF
127 # based on  $K_n$ :
128 # test_SO_sup == -1:  $H_{\{01\}}^c$  not rejected
129 # test_SO_sup == 0:  $H_{\{01\}}^c$  rejected,  $H_0$  not rejected
130 # test_SO_sup == 1:  $H_{\{01\}}^c$  rejected,  $H_0$  rejected (i.e. supporting  $H_1$ )
131 ##### END producing p-value and decision results, given a
132     specific dataset
133 ###### power function
134 powerfn = function(n, parameters, p1, p2, split) {
135 # Computes the empirical rejection rate of our composite procedure
136 # Args:
137 # n: total sample size
138 # parameters: the parameters input/calculated in the codes above (up to
139 #     # END other parameter calculations based on user input)
140 # p1: proportion of data for the first group
141 # p2: proportion of data for the second group
142 # split: number of parallel tasks for computing nrep (see parameters$  

143     nrep above) results

```

```

142 # Returns:
143 #   the empirical rejection rate of our composite procedure
144 #### parameter organization
145 c = rep(parameters$c, 3)
146 n1 = ceiling(n * p1)
147 n2 = ceiling(n * p2)
148 n3 = n - n1 - n2
149 nrep_sub = parameters$nrep_sub
150 #### define output
151 test_nocross = 1:nrep_sub * 0
152 test = 1:nrep_sub * 0
153 crit_boot = matrix(0, nrow = nrep_sub, ncol = length(parameters$alpha_vec))
154 inttest_dF = 1:nrep_sub * 0
155 crit_boot_int_dF = matrix(0, nrow = nrep_sub, ncol = length(parameters$alpha_vec))
156 test_SO_sup = 1:nrep_sub * 0
157 test_SO_dF = 1:nrep_sub * 0
158 #### for loop generating nrep_sub number of datasets and computing
159 # statistics accordingly
160 for (i in 1:nrep_sub) {
161   set.seed(parameters$seedstart + nrep_sub * (split - 1) + i)
162   print(parameters$seedstart + nrep_sub * (split - 1) + i)
163   ## data generation
164   X1 = rweibull(n = n1, shape = a1b1[1], scale = a1b1[2])
165   X2 = rweibull(n = n2, shape = a2b2[1], scale = a2b2[2])
166   X3 = rweibull(n = n3, shape = a3b3[1], scale = a3b3[2])
167   C1 = pmin(runif(n1, min = 0, max = c[1]), parameters$adminC)
168   C2 = pmin(runif(n2, min = 0, max = c[2]), parameters$adminC)
169   C3 = pmin(runif(n3, min = 0, max = c[3]), parameters$adminC)
170   ## make censoring data:
171   Y1 = apply(cbind(X1, C1), 1, min)
172   Y2 = apply(cbind(X2, C2), 1, min)
173   Y3 = apply(cbind(X3, C3), 1, min)
174   delta1 = (X1 <= C1)
175   delta2 = (X2 <= C2)
176   delta3 = (X3 <= C3)
177   sort1 = sort(Y1, index.return = TRUE)$ix
178   sort2 = sort(Y2, index.return = TRUE)$ix
179   sort3 = sort(Y3, index.return = TRUE)$ix
180   data = matrix(c(Y1[sort1], Y2[sort2], Y3[sort3], delta1[sort1], delta2[sort2], delta3[sort3], rep(1, times = n1), rep(2, times = n2), rep(3, times = n3)), ncol = 3)
181   dat = Surv(data[, 1], data[, 2])
182   ## survival analysis
183   fit = survfit(dat ~ 1)
184   fit1 = survfit(dat[data[, 3] == 1] ~ 1)
185   fit2 = survfit(dat[data[, 3] == 2] ~ 1)

```

```

185 fit3 = survfit(dat[data[, 3] == 3] ~ 1)
186 ## test stat fn apply
187 print(Sys.time())
188 teststatfn = teststat(parameters$tau1, parameters$tau2, fit, fit1,
189   fit2, fit3, parameters$M_vec, M_ret, data, parameters$alpha_vec,
190   parameters$nboot)
191 print(Sys.time())
192 ## the proposed two-step procedure
193 test_nocross[i] = teststatfn$test_nocross
194 test[i] = teststatfn$test
195 crit_boot[i, ] = teststatfn$EL_SOcrit # [, j] corresponds to j-th
196   element of alpha_vec
197 inttest_dF[i] = teststatfn$inttest_dF
198 crit_boot_int_dF[i, ] = teststatfn$int_dFEL_SOcrit # [, j]
199   corresponds to j-th element of alpha_vec
200 if (test_nocross[i] == 1) {
201   if (test[i] > crit_boot[i, 1]) {
202     test_SO_sup[i] = 1
203   } # END if
204   if (inttest_dF[i] > crit_boot_int_dF[i, 1]) {
205     test_SO_dF[i] = 1
206   } # END if
207 } else {
208   test_SO_sup[i] = -1
209   test_SO_dF[i] = -1
210 } # END if/else
211 } # END for
212
213 # finally get rej rate
214 if (parameters$option_test == 'two_step_sup') {
215   rp = sum(test_SO_sup[test_nocross == 1] == 1) / nrep_sub
216 } else if (parameters$option_test == 'two_step_dF') {
217   rp = sum(test_SO_dF[test_nocross == 1] == 1) / nrep_sub
218 } #END if/else if
219 return(rp)
220 } # END powerfn
221 ##### END power function
222
223 ##### calculating power given n, p1, p2 (utilizing R parallel
224   computing)
225 # initiate cluster
226 cl = makeCluster(no_cores)
227 registerDoParallel(cl)
228 #### compute the results:
229 time1 = Sys.time()
230 parameters$nrep
231 foreach(split = 1:parameters$nsplit ,
232   .combine = c,

```

```

228 .packages = c("survival", "nloptr", "plyr", "Iso")
229 ) %dopar% {
230   out = powerfn(n = n, parameters, p1 = p1, p2 = p2, split)
231   setwd(dir_path2)
232   save(out, file = paste("n_", n, "_split_", split, '_nrep_', parameters$nrep,
233                           "_p1_", p1, "_p2_", p2, ".Rdata", sep = ""))
233 } # END foreach
234 time2 = Sys.time()
235 time2-time1
236 stopCluster(cl)
237 ### read the results:
238 rp_vec = 1:parameters$nsplit * 0
239 setwd(dir_path2)
240 for (split in 1:parameters$nsplit) {
241   load(paste("n_", n, "_split_", split, '_nrep_', parameters$nrep, "_p1_",
242             p1, "_p2_", p2, ".Rdata", sep = ""))
243   rp_vec[split] = out
244 } # END for
245 mean(rp_vec) # the desired power
245 #####END calculating power given n, p1, p2

```

power_calculations.R

```

1 ##### preliminary functions
2 division00 = function(x, y) {
3 # Computes the fraction of two vectors following the 0 / 0 = 1 convention.
4 # (can't deal with Inf / Inf though)
5 # Args:
6 #   x: numerator
7 #   y: denominator
8 #   x, y should be vectors of the same length
9 # Returns:
10 #   out: x / y; when x[i] = 0 and y[i] = 0, out[i] = 1
11   out = x / y
12   out[as.logical((x == 0) * (y == 0))] = 1
13   return(out)
14 } # END division00
15 product = function(x, y) {
16 # Computes the product of two vectors following the 0 * Inf = 0 convention
17 .
18 # Args:
19 #   x, y are vectors of the same length
20 # Returns:
21 #   out: x * y; when x[i] = 0 or y[i] = 0, out[i] = 0
22   out = x * y
23   out[(x == 0 | y == 0)] = 0
24   return(out)
25 } # END product
26 product_mat = function(x, y) {

```

```

25 # Computes the product of two matrices following the 0 * Inf = 0
26   convention .
27 # Args:
28 #   x, y are matrices of the same dimensions
29 # Returns:
30 #   out: x * y; when x[i, ] = 0 or y[i, ] = 0, out[i, ] = 0
31 out = x * y
32 for (i in 1:dim(x)[1]) {
33   out[i, (x[i, ] == 0 | y[i, ] == 0)] = 0
34 }
35 return(out)
36 } # END product_mat
37 ###### END preliminary functions
38
39 ###### EL test related functions
40 neg_log_likelihood_elt_eq_h = function(t, fit1, fit2, fit3, M_vec, init_
41   lambdas = c(0,0), maxeval = 10000) {
42 # Computes the negative log likelihood at a given time t
43 # when subject to equality constraint S_1 ^ {M_1} \succ S_2 ^ {M_2} \succ
44 # S_3 ^ {M_3}.
45 # (\succ is the latex notation we used to denote pointwise greater than)
46 # Args:
47 #   t: the given t>=0 in localized EL statistic
48 #   fit1: survfit applied to data from the 1st treatment group (see power_
49 #         calculations.R)
50 #   fit2: survfit applied to data from the 2nd treatment group
51 #   fit3: survfit applied to data from the 3rd treatment group
52 #   M_vec: the vector of margins M_1, M_2, M_3
53 #   init_lambdas: the vector of initial values for the Lagrange
54 #                 multipliers
55 #   maxeval: maximum number of function evaluations
56 # Returns:
57 #   the negative log likelihood at a given time t
58 #   when subject to equality constraint S_1 ^ {M_1} \succ S_2 ^ {M_2} \
59 #     succ S_3 ^ {M_3}
60 d1 = fit1$n.event[fit1$time <= t & fit1$n.event != 0]
61 r1 = fit1$n.risk[fit1$time <= t & fit1$n.event != 0]
62 d2 = fit2$n.event[fit2$time <= t & fit2$n.event != 0]
63 r2 = fit2$n.risk[fit2$time <= t & fit2$n.event != 0]
64 d3 = fit3$n.event[fit3$time <= t & fit3$n.event != 0]
65 r3 = fit3$n.risk[fit3$time <= t & fit3$n.event != 0]
66 A1 = r1 - d1
67 A2 = r2 - d2
68 A3 = r3 - d3
69 D1 = max(d1 - r1) / M_vec[1] + 0.0001
70 D2 = max(d2 - r2) / M_vec[2] + 0.0001
71 D3 = max(d3 - r3) / M_vec[3] + 0.0001

```

```

67  init_weights = c(d1 / (r1 + M_vec[1] * init_lambdas[1]), d2 / (r2 + M_
68    vec[2] * (init_lambdas[2] - init_lambdas[1])), d3 / (r3 - M_vec[3] *
69      init_lambdas[2]))
70  fnc_objective_h = function(h) {
71    # Computes the negative log likelihood at a given time t for a given
72    # vector of h
73    # Args:
74    #   h: the vector of hazard probabilities. At init_lambdas, h == c(d1/r1
75    #     , d2/r2, d3/r3)
76    # Returns:
77    #   the negative log likelihood at a given time t
78    h1 = h[1:length(d1)]
79    h2 = h[(length(d1) + 1):(length(d1) + length(d2))]
80    h3 = h[(length(d1) + length(d2) + 1):(length(d1) + length(d2) + length
81      (d3))]
82    out = sum(d1 * log(h1)) + sum(d2 * log(h2)) + sum(d3 * log(h3)) + sum(
83      product(A1, log(1 - h1))) + sum(product(A2, log(1 - h2))) + sum(
84      product(A3, log(1 - h3)))
85    return (-out)
86  } # END fnc_objective_h
87  gnc_objective_h = function(h) {
88    # Computes the gradient of the negative log likelihood at a given time t
89    # for a given vector of h
90    # Args:
91    #   h: the vector of hazard probabilities. At init_lambdas, h == c(d1/r1
92    #     , d2/r2, d3/r3)
93    # Returns:
94    #   the gradient of the negative log likelihood at a given time t
95    h1 = h[1:length(d1)]
96    h2 = h[(length(d1) + 1):(length(d1) + length(d2))]
97    h3 = h[(length(d1) + length(d2) + 1):(length(d1) + length(d2) + length
98      (d3))]
99    out = c(division00(d1, h1) - division00(r1 - d1, 1 - h1), division00(
100        d2, h2) - division00(r2 - d2, 1 - h2), division00(d3, h3) -
      division00(r3 - d3, 1 - h3))
101   return(-out)
102 } # END gnc_objective_h
103 constraints_h = function(h) {
104   # Computes the equality constraints in the form of a vector that is == 0
105   # Args:
106   #   h: the vector of hazard probabilities. At init_lambdas, h == c(d1/r1
107   #     , d2/r2, d3/r3)
108   # Returns:
109   #   the vector that is == 0
110   h1 = h[1:length(d1)]
111   h2 = h[(length(d1) + 1):(length(d1) + length(d2))]
112   h3 = h[(length(d1) + length(d2) + 1):(length(d1) + length(d2) + length
113     (d3))]

```

```

101  return(c(log(division00(prod((1 - h2) ^ M_vec[2]), prod((1 - h1) ^ M_
102    vec[1]))), log(division00(prod((1 - h3) ^ M_vec[3]), prod((1 - h2)
103      ^ M_vec[2])))))
} # END constraints_h
104 jac_constraints_h = function(h) {
# Computes the Jacobian of the output of constraints_h above as a
# function of h
105 # Args:
#   h: the vector of hazard probabilities. At init_lambdas, h == c(d1/r1
#     , d2/r2, d3/r3)
106 # Returns:
#   the Jacobian of the output of constraints_h above as a function of h
107 h1 = h[1:length(d1)]
108 h2 = h[(length(d1) + 1):(length(d1) + length(d2))]
109 h3 = h[(length(d1) + length(d2) + 1):(length(d1) + length(d2) + length
#   (d3))]
110 out1 = c(division00(M_vec[1], 1 - h1), division00(-M_vec[2], 1 - h2),
111   rep(0, length = length(d3)))
112 out2 = c(rep(0, length = length(d1)), division00(M_vec[2], 1 - h2),
113   division00(-M_vec[3], 1 - h3))
114 return(rbind(out1, out2))
} # END jac_constraints_h
115 local_opts = list("algorithm" = "NLOPT_LD_MMA", "xtol_rel" = 1.0e-10)
116 opts_used = list("algorithm" = "NLOPT_LD_AUGLAG", "xtol_rel" = 1.0e-10,
117   "maxeval" = maxeval, "local_opts" = local_opts)
118 nlopt = nloptr(x0 = init_weights, eval_f = fnc_objective_h, eval_grad_f
= gnc_objective_h,
119   lb = rep(0.0, times = length(d1) + length(d2) + length(d3
#   )),
120   ub = rep(1.0, times = length(d1) + length(d2) + length(d3
#   )),
121   eval_g_eq = constraints_h, eval_jac_g_eq = jac_
#   constraints_h, opts = opts_used)
122 nlopt$Ds = c(D1,D2,D3)
123 nlopt$resultEEs = constraints_h(nlopt$solution)
124 nlopt$lambdas = c((d1[1] / nlopt$solution[1] - r1[1]) / M_vec[1], (r3[1]
#   - d3[1] / nlopt$solution[(length(d1) + length(d2) + 1)]) / M_vec[3])
125 return(nlopt)
} # END neg_log_likelihood_le_t_eq_h
126
127
128 neg_log_likelihood_le_t_ineq_h = function(t, fit1, fit2, fit3, M_vec, init
#   _lambdas=c(0,0), maxeval = 10000) {
129 # Computes the negative log likelihood at a given time t
130 # when subject to inequality constraint S_1 ^ {M_1} \succ S_2 ^ {M_2} \
#   succ S_3 ^ {M_3}.
131 # (\succ is the latex notation we used to denote pointwise greater than)
132 # Args:
133 #   t: the given t>=0 in localized EL statistic

```

```

134 # fit1: survfit applied to data from the 1st treatment group (see power_
135 # calculations.R)
136 # fit2: survfit applied to data from the 2nd treatment group
137 # fit3: survfit applied to data from the 3rd treatment group
138 # M_vec: the vector of margins M_1, M_2, M_3
139 # init_lambdas: the vector of initial values for the Lagrange
140 # multipliers
141 # maxeval: maximum number of function evaluations
142 # Returns:
143 #   the negative log likelihood at a given time t
144 #   when subject to inequality constraint S_1 ^ {M_1} \succ S_2 ^ {M_2} \
145 #     succ S_3 ^ {M_3}
146 d1 = fit1$n.event[fit1$time <= t & fit1$n.event != 0]
147 r1 = fit1$n.risk[fit1$time <= t & fit1$n.event != 0]
148 d2 = fit2$n.event[fit2$time <= t & fit2$n.event != 0]
149 r2 = fit2$n.risk[fit2$time <= t & fit2$n.event != 0]
150 d3 = fit3$n.event[fit3$time <= t & fit3$n.event != 0]
151 r3 = fit3$n.risk[fit3$time <= t & fit3$n.event != 0]
152 A1 = r1 - d1
153 A2 = r2 - d2
154 A3 = r3 - d3
155 D1 = max(d1 - r1) / M_vec[1] + 0.0001
156 D2 = max(d2 - r2) / M_vec[2] + 0.0001
157 D3 = max(d3 - r3) / M_vec[3] + 0.0001
158 init_weights = c(d1 / (r1 + M_vec[1] * init_lambdas[1]), d2 / (r2 + M_
159   vec[2] * (init_lambdas[2] - init_lambdas[1])), d3 / (r3 - M_vec[3] *
160   init_lambdas[2]))
161 fnc_objective_h = function(h) {
162   # Computes the negative log likelihood at a given time t for a given
163   # vector of h
164   # Args:
165   #   h: the vector of hazard probabilities. At init_lambdas, h == c(d1/r1
166   #     , d2/r2, d3/r3)
167   # Returns:
168   #   the negative log likelihood at a given time t
169   h1 = h[1:length(d1)]
170   h2 = h[(length(d1) + 1):(length(d1) + length(d2))]
171   h3 = h[(length(d1) + length(d2) + 1):(length(d1) + length(d2) + length
172     (d3))]
173   out = sum(d1 * log(h1)) + sum(d2 * log(h2)) + sum(d3 * log(h3)) + sum(
174     product(A1, log(1 - h1))) + sum(product(A2, log(1 - h2))) + sum(
175     product(A3, log(1 - h3)))
176   return (-out)
177 } # END fnc_objective_h
178 gnc_objective_h = function(h) {
179   # Computes the gradient of the negative log likelihood at a given time t
180   # for a given vector of h
181   # Args:

```

```

171  #   h: the vector of hazard probabilities. At init_lambdas, h == c(d1/r1
172  #           , d2/r2, d3/r3)
173  # Returns:
174  #   the gradient of the negative log likelihood at a given time t
175  h1 = h[1:length(d1)]
176  h2 = h[(length(d1) + 1):(length(d1) + length(d2))]
177  h3 = h[(length(d1) + length(d2) + 1):(length(d1) + length(d2) + length
178  #           (d3))]
179  out = c(dvision00(d1, h1) - division00(r1 - d1, 1 - h1), division00(
180  #           d2, h2) - division00(r2 - d2, 1 - h2), division00(d3, h3) -
181  #           division00(r3 - d3, 1 - h3))
182  return (-out)
183 } # END gnc_objective_h
184 constraints_h = function(h) {
185 # Computes the inequality constraints in the form of a vector that is <=
186 #           0
187 # Args:
188 #   h: the vector of hazard probabilities. At init_lambdas, h == c(d1/r1
189 #           , d2/r2, d3/r3)
190 # Returns:
191 #   the vector that is <= 0
192 h1 = h[1:length(d1)]
193 h2 = h[(length(d1) + 1):(length(d1) + length(d2))]
194 h3 = h[(length(d1) + length(d2) + 1):(length(d1) + length(d2) + length
195 #           (d3))]
196 return(c(log(dvision00(prod((1 - h2) ^ M_vec[2]), prod((1 - h1) ^ M_
197 #           vec[1])), log(dvision00(prod((1 - h3) ^ M_vec[3]), prod((1 - h2)
198 #           ^ M_vec[2])))))
199 } # END constraints_h
200 jac_constraints_h = function(h) {
201 # Computes the Jacobian of the output of constraints_h above as a
202 #   function of h
203 # Args:
204 #   h: the vector of hazard probabilities. At init_lambdas, h == c(d1/r1
205 #           , d2/r2, d3/r3)
206 # Returns:
207 #   the Jacobian of the output of constraints_h above as a function of h
208 h1 = h[1:length(d1)]
209 h2 = h[(length(d1) + 1):(length(d1) + length(d2))]
210 h3 = h[(length(d1) + length(d2) + 1):(length(d1) + length(d2) + length
211 #           (d3))]
212 out1 = c(dvision00(M_vec[1], 1 - h1), division00(-M_vec[2], 1 - h2),
213 #           rep(0, length = length(d3)))
214 out2 = c(rep(0, length = length(d1)), division00(M_vec[2], 1 - h2),
215 #           division00(-M_vec[3], 1 - h3))
216 return(rbind(out1, out2))
217 } # END jac_constraints_h
218 local_opts = list("algorithm" = "NLOPT_LD_MMA", "xtol_rel" = 1.0e-10)

```

```

205 opts_used = list("algorithm" = "NLOPT_LD_AUGLAG", "xtol_rel" = 1.0e-10,
206   "maxeval" = maxeval, "local_opts" = local_opts)
207 nlopt = nloptr(x0 = init_weights, eval_f = fnc_objective_h, eval_grad_f
208   = gnc_objective_h,
209   lb = rep(0.0, times = length(d1) + length(d2) + length(d3)
210     ), ub = rep(1.0, times = length(d1) + length(d2) +
211       length(d3)),
212   eval_g_ineq = constraints_h, eval_jac_g_ineq = jac_
213     constraints_h, opts = opts_used)
214 nlopt$Ds = c(D1, D2, D3)
215 nlopt$resultEEs = constraints_h(nlopt$solution) # all elements need to
216   be <=0
217 nlopt$lambda = c((d1[1] / nlopt$solution[1] - r1[1]) / M_vec[1], (r3[1]
218   - d3[1] / nlopt$solution[(length(d1) + length(d2) + 1)]) / M_vec[3])
219 return (nlopt)
220 } # END neg_log_likelihood_le_t_ineq_h
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236 a_1 = function(lambda, fit1, fit2, t, tilde_theta = 1, M_vec2) {
237 # Computes a_j(lambda) - tilde_theta, where a_j(lambda) corresponds to
238 # equation (3.4) in the paper, for a given number tilde_theta
239 # This is to solve for lambda0 later, for the pairwise NPLR tests, for a
240 # given number t >= 0
241 # Args:
242 # lambda: the Lagrange multiplier in the numerator of equation (3.4) in
243 # the paper
244 # fit1: survfit applied to data from the j-th treatment group in
245 # equation (3.4) in the paper
246 # fit2: survfit applied on data from the (j+1)-th treatment group in
247 # equation (3.4) in the paper
248 # t: the given t >= 0 in localized EL statistic
249 # tilde_theta: the given number tilde_theta
250 # M_vec2: the vector of margins M_j, M_{j+1}
251 # Returns:
252 # a_j(lambda)-tilde_theta, where a_j(lambda) corresponds to equation
253 # (3.4) in the paper, for the given number tilde_theta
254 if (sum(fit1$n.risk == fit1$n.event) == 0) {
255   h1 = division00(fit1$n.event, (fit1$n.risk + M_vec2[1] * lambda))
256 } else {
257   h1 = division00(fit1$n.event, (fit1$n.risk + M_vec2[1] * lambda)) * as
258     .numeric(fit1$n.risk != fit1$n.event) + division00(fit1$n.event, (
259       fit1$n.event + M_vec2[1] * lambda)) * as.numeric(fit1$n.risk ==
260         fit1$n.event)
261 } # END if/else
262 if (sum(fit2$n.risk == fit2$n.event) == 0) {
263   h2 = division00(fit2$n.event, (fit2$n.risk - M_vec2[2] * lambda))
264 } else {

```

```

236 h2 = division00(fit2$n.event, (fit2$n.risk - M_vec2[2] * lambda)) * as
237   .numeric(fit2$n.risk != fit2$n.event) + division00(fit2$n.event, (
238     fit2$n.event - M_vec2[2] * lambda)) * as.numeric(fit2$n.risk ==
239     fit2$n.event)
240 } # END if/else
241 num = ((1 - h1) ^ M_vec2[1])[fit1$time <= t]
242 denom = ((1 - h2) ^ M_vec2[2])[fit2$time <= t]
243 return(division00(prod(num), prod(denom)) - tilde_theta)
244 } # END a_1
245
246 a_123_ret = function(lambda, fit1, fit2, fit3, t, tilde_theta = 1, M_ret)
247 {
248 # Computes a(lambda) - tilde_theta, where a(lambda) = the RHS / LHS of the
249 # 2nd display in p. 17 of the supplementary material,
250 # for a given number tilde_theta
251 # This is to solve for lambda0_ret later, for the retention-of-effect
252 # approach, for a given number t >= 0
253 # Args:
254 #   lambda: the Lagrange multiplier in the first display in p. 17 of the
255 #   supplementary material
256 #   fit1: survfit applied to data from the 1st treatment group (see power-
257 #   calculations.R)
258 #   fit2: survfit applied to data from the 2nd treatment group
259 #   fit3: survfit applied to data from the 3rd treatment group
260 #   t: the given t >= 0 in localized EL statistic
261 #   tilde_theta: the given number tilde_theta
262 #   M_ret: the margin M in (5.1) of the paper
263 # Returns:
264 #   a(lambda) - tilde_theta, where a(lambda) = the RHS / LHS of the 2nd
265 #   display in p. 17 of the supplementary material,
266 #   for the given number tilde_theta
267 if (sum(fit1$n.risk == fit1$n.event) == 0) {
268   h1 = division00(fit1$n.event, (fit1$n.risk + (M_ret - 1) * lambda))
269 } else {
270   h1 = division00(fit1$n.event, (fit1$n.risk + (M_ret - 1) * lambda)) *
271     as.numeric(fit1$n.risk != fit1$n.event) + division00(fit1$n.event,
272     (fit1$n.event + (M_ret - 1) * lambda)) * as.numeric(fit1$n.risk ==
273     fit1$n.event)
274 } # END if/else
275 if (sum(fit2$n.risk == fit2$n.event) == 0) {
276   h2 = division00(fit2$n.event, (fit2$n.risk + lambda))
277 } else {
278   h2 = division00(fit2$n.event, (fit2$n.risk + lambda)) * as.numeric(
279     fit2$n.risk != fit2$n.event) + division00(fit2$n.event, (fit2$n.
280     event + lambda)) * as.numeric(fit2$n.risk == fit2$n.event)
281 } # END if/else
282 if (sum(fit3$n.risk == fit3$n.event) == 0) {
283   h3 = division00(fit3$n.event, (fit3$n.risk - M_ret * lambda))

```

```

270 } else {
271   h3 = division00(fit3$n.event, (fit3$n.risk - M.ret * lambda)) * as.numeric(fit3$n.risk != fit3$n.event) + division00(fit3$n.event, (fit3$n.event - M.ret * lambda)) * as.numeric(fit3$n.risk == fit3$n.event)
272 } # END if/else
273 num = ((1 - h3) ^ M.ret)[fit3$time <= t]
274 denom1 = ((1 - h1) ^ (M.ret - 1))[fit1$time <= t]
275 denom2 = (1 - h2)[fit2$time <= t]
276 return(division00(prod(num), prod(denom1) * prod(denom2)) - tilde_theta)
277 } # END a_123_ret
278
279 lambda0_ret = function(t, fit1, fit2, fit3, tilde_theta = 1, M.ret) {
280 # Computes the root of a(lambda) - tilde_theta = 0 (see a_123_ret above),
281 # for the retention-of-effect approach, for a given number t >= 0
282 # where a(lambda) = the RHS / LHS of the 2nd display in p. 17 of the
283 # supplementary material, for a given number tilde_theta
284 # Args:
285 #   t: the given t >= 0 in localized EL statistic
286 #   fit1: survfit applied to data from the 1st treatment group (see power_
287 #         calculations.R)
288 #   fit2: survfit applied to data from the 2nd treatment group
289 #   fit3: survfit applied to data from the 3rd treatment group
290 #   tilde_theta: the given number tilde_theta
291 #   M.ret: the margin M in (5.1) of the paper
292 # Returns:
293 #   the root of a(lambda) - tilde_theta = 0 for the retention-of-effect
294 #   approach
295 #   for the given number tilde_theta and the given number t >= 0
296 out = 1:length(t) * 0
297 for (i in 1:length(t)) {
298   if (sum(fit1$time[fit1$n.event != 0] <= t[i]) == 0 | sum(fit2$time[
299     fit2$n.event != 0] <= t[i]) == 0 | sum(fit3$time[fit3$n.event != 0]
300     <= t[i]) == 0) {
301     out[i] = NA
302   } else {
303     d1 = fit1$n.event[fit1$time <= t[i] & fit1$n.event != 0]
304     r1 = fit1$n.risk[fit1$time <= t[i] & fit1$n.event != 0]
305     d2 = fit2$n.event[fit2$time <= t[i] & fit2$n.event != 0]
306     r2 = fit2$n.risk[fit2$time <= t[i] & fit2$n.event != 0]
307     d3 = fit3$n.event[fit3$time <= t[i] & fit3$n.event != 0]
308     r3 = fit3$n.risk[fit3$time <= t[i] & fit3$n.event != 0]
309     D1 = max(d1 - r1) / (M.ret - 1) + 0.0001
310     D2 = max(d2 - r2) + 0.0001
311     D3 = max(d3 - r3) / M.ret + 0.0001
312     if (max(D1,D2) != (-D3)) {
313       out[i] = uniroot(a_123_ret, interval = c(max(D1, D2), -D3), tol =
314         0.0001, fit1 = fit1, fit2 = fit2, fit3 = fit3, t = t[i], tilde_theta)
315     }
316   }
317 }

```

```

            theta = tilde_theta, M_ret = M_ret )$root
308     } else {
309         out[i] = max(D1, D2)
310     } # END if/else
311 } # END if/else
312 } # END for
313 return(out)
314 } # END lambda0_ret

315
316 lambda0 = function(t, fit1, fit2, tilde_theta = 1, M_vec2) {
317 # Computes the root of a_j(lambda) - tilde_theta = 0 (see a_1 above), for
# the pairwise NPLR tests, for a given number t >= 0
318 # where a_j(lambda) corresponds to equation (3.4) in the paper, for a
# given number tilde_theta
319 # Args:
320 #   t: the given t >= 0 in localized EL statistic
321 #   fit1: survfit applied to data from the j-th treatment group in
#         equation (3.4) in the paper
322 #   fit2: survfit applied on data from the (j+1)-th treatment group in
#         equation (3.4) in the paper
323 #   tilde_theta: the given number tilde_theta
324 #   M_vec2: the vector of margins M_j, M_{j+1}
325 # Returns:
326 #   the root of a_j(lambda) - tilde_theta = 0, where a_j(lambda)
# corresponds to equation (3.4) in the paper
327 #   for the given number tilde_theta and the given number t >= 0
328 out = 1:length(t) * 0
329 for (i in 1:length(t)) {
330     if (sum(fit1$time[fit1$n.event != 0] <= t[i]) == 0 | sum(fit2$time[fit2$n.
# event != 0] <= t[i]) == 0) {
331         out[i] = NA
332     } else {
333         D1 = max((fit1$n.event - fit1$n.risk)[fit1$time <= t[i] & fit1$n.
# event != 0]) / M_vec2[1]
334         D2 = max((fit2$n.event - fit2$n.risk)[fit2$time <= t[i] & fit2$n.
# event != 0]) / M_vec2[2]
335         if (D1 != (-D2)) {
336             out[i] = uniroot(a_1, interval = c(D1 + 0.0001, -D2 - 0.0001), tol
# = 0.0001, fit1 = fit1, fit2 = fit2, t = t[i], tilde_theta =
tilde_theta, M_vec2 = M_vec2)$root
337         } else {
338             out[i] = D1
339         } # END if/else
340     } # END if/else
341 } # END for
342 return(out)
343 } # END lambda0
344
```

```

345 sigma2_hat_overpj = function(t, fit, fit1, fit2, fit3, M_vec) {
346 # Computes the vector of  $\hat{\sigma}_j^2(t) / (n_j / n)$ , j = 1, 2, 3,
347 # as in Section 2.1 of the paper for a given number t >= 0
348 # Args:
349 #   t: the given t >= 0 in localized EL statistic
350 #   fit: survfit applied to the pooled data from all groups (see power_
351 #         calculations.R)
352 #   fit1: survfit applied to data from the 1st treatment group
353 #   fit2: survfit applied to data from the 2nd treatment group
354 #   fit3: survfit applied to data from the 3rd treatment group
355 #   M_vec: the vector of margins M_1, M_2, M_3
356 # Returns:
357 #   the  $\hat{\sigma}_j^2(t) / (n_j / n)$  vector (j = 1, 2, 3) as in
358 #   Section 2.1 of the paper for a given number t >= 0
359 n = fit$n
360 out = matrix(0, nrow = length(t), ncol = 3)
361 for (i in 1:length(t)) {
362   d1 = fit1$n.event[fit1$time <= t[i] & fit1$n.event != 0]
363   r1 = fit1$n.risk[fit1$time <= t[i] & fit1$n.event != 0]
364   d2 = fit2$n.event[fit2$time <= t[i] & fit2$n.event != 0]
365   r2 = fit2$n.risk[fit2$time <= t[i] & fit2$n.event != 0]
366   d3 = fit3$n.event[fit3$time <= t[i] & fit3$n.event != 0]
367   r3 = fit3$n.risk[fit3$time <= t[i] & fit3$n.event != 0]
368   out[i,1] = n * sum(division00(d1, r1 * (r1 - d1)))
369   out[i,2] = n * sum(division00(d2, r2 * (r2 - d2)))
370   out[i,3] = n * sum(division00(d3, r3 * (r3 - d3)))
371 } # END for
372 return(out)
373 } # END sigma2_hat_overpj
374
375 theta_hat_j = function(t, fit, fit1, fit2, fit3, M_vec) {
376 # Computes the vector of  $\hat{\theta}_j(t)$ , j = 1, 2, 3, as in Section 2.1
377 #   of the paper for a given number t >= 0
378 # Args:
379 #   t: the given t >= 0 in localized EL statistic
380 #   fit: survfit applied to the pooled data from all groups (see power_
381 #         calculations.R)
382 #   fit1: survfit applied to data from the 1st treatment group
383 #   fit2: survfit applied to data from the 2nd treatment group
384 #   fit3: survfit applied to data from the 3rd treatment group
385 #   M_vec: the vector of margins M_1, M_2, M_3
386 # Returns:
387 #   the  $\hat{\theta}_j(t)$  vector (j = 1, 2, 3) as in Section 2.1 of the
388 #   paper for a given number t >= 0
389 n = fit$n
390 out = matrix(0, nrow = length(t), ncol = 3)
391 for (i in 1:length(t)) {
392   d1 = fit1$n.event[fit1$time <= t[i] & fit1$n.event != 0]

```

```

388 r1 = fit1$n.risk[fit1$time <= t[i] & fit1$n.event != 0]
389 d2 = fit2$n.event[fit2$time <= t[i] & fit2$n.event != 0]
390 r2 = fit2$n.risk[fit2$time <= t[i] & fit2$n.event != 0]
391 d3 = fit3$n.event[fit3$time <= t[i] & fit3$n.event != 0]
392 r3 = fit3$n.risk[fit3$time <= t[i] & fit3$n.event != 0]
393 out[i, 1] = (M_vec[1]^2) * n * sum(division00(d1, r1 * (r1 - d1)))
394 out[i, 2] = (M_vec[2]^2) * n * sum(division00(d2, r2 * (r2 - d2)))
395 out[i, 3] = (M_vec[3]^2) * n * sum(division00(d3, r3 * (r3 - d3)))
396 } # END for
397 return(out)
398 } # END theta_hat_j
399
400 psi_hat_j = function(t, fit, fit1, fit2, fit3, M_vec) {
# Computes the vector of the asymptotic standard deviation of \hat{S}_j ^ {M_j}(t) for a given number t >= 0,
401 # as in line 8 of p. 9 in the paper
402 # Args:
403 #   t: the given t >= 0 in localized EL statistic
404 #   fit: survfit applied to the pooled data from all groups (see power_
405 #         calculations.R)
406 #   fit1: survfit applied to data from the 1st treatment group
407 #   fit2: survfit applied to data from the 2nd treatment group
408 #   fit3: survfit applied to data from the 3rd treatment group
409 #   M_vec: the vector of margins M_1, M_2, M_3
410 # Returns:
411 #   the vector of the asymptotic standard deviation of \hat{S}_j ^ {M_j}(t)
412 #       ) for a given number t >= 0
412 n = fit$n
413 out = matrix(0, nrow = length(t), ncol = 3)
414 for (i in 1:length(t)) {
415   d1 = fit1$n.event[fit1$time <= t[i] & fit1$n.event != 0]
416   r1 = fit1$n.risk[fit1$time <= t[i] & fit1$n.event != 0]
417   d2 = fit2$n.event[fit2$time <= t[i] & fit2$n.event != 0]
418   r2 = fit2$n.risk[fit2$time <= t[i] & fit2$n.event != 0]
419   d3 = fit3$n.event[fit3$time <= t[i] & fit3$n.event != 0]
420   r3 = fit3$n.risk[fit3$time <= t[i] & fit3$n.event != 0]
421   tcalc1 = t[i] - fit1$time
422   tcalc1[tcalc1 < 0] = Inf
423   tcalc2 = t[i] - fit2$time
424   tcalc2[tcalc2 < 0] = Inf
425   tcalc3 = t[i] - fit3$time
426   tcalc3[tcalc3 < 0] = Inf
427   S1 = fit1$surv[which.min(tcalc1)]
428   S2 = fit2$surv[which.min(tcalc2)]
429   S3 = fit3$surv[which.min(tcalc3)]
430   if(t[i] < min(fit1$time)){
431     S1 = 1
432   } # END if

```

```

433 if( t [ i ] < min( fit2$time) ){
434   S2 = 1
435 } # END if
436 if( t [ i ] < min( fit3$time) ){
437   S3 = 1
438 } # END if
439 out [ i , 1 ] = product(S1 ^ M_vec [ 1 ] , M_vec [ 1 ] * sqrt(n * sum(division00(
440   d1 , r1 * (r1 - d1)))) )
440 out [ i , 2 ] = product(S2 ^ M_vec [ 2 ] , M_vec [ 2 ] * sqrt(n * sum(division00(
441   d2 , r2 * (r2 - d2)))) )
441 out [ i , 3 ] = product(S3 ^ M_vec [ 3 ] , M_vec [ 3 ] * sqrt(n * sum(division00(
442   d3 , r3 * (r3 - d3)))) )
443 } # END for
444 return(out)
445 } # END psi_hat_j
446
447 psi_hat_ret = function(t , fit , fit1 , fit2 , M_ret_vec) {
448 # Computes the asymptotic standard deviation of \hat{S}_1 ^ {M-1}(t) \hat{S}_2(t) for a given number t >= 0,
449 # as in p. 17 of the supplementary material
450 # Args:
451 #   t: the given t >= 0 in localized EL statistic
452 #   fit: survfit applied to the pooled data from all groups (see power_calculations.R)
453 #   fit1: survfit applied to data from the 1st treatment group
454 #   fit2: survfit applied to data from the 2nd treatment group
455 #   M_ret_vec: the vector of margins c(M_ret - 1, 1, M_ret), where M_ret is the margin M in (5.1) of the paper
456 # Returns:
457 #   the asymptotic standard deviation of \hat{S}_1 ^ {M-1}(t) \hat{S}_2(t) for a given number t >= 0
458 n = fit$n
459 out = 1:length(t) * 0
460 for (i in 1:length(t)) {
461   d1 = fit1$n.event[fit1$time <= t[i] & fit1$n.event != 0]
462   r1 = fit1$n.risk[fit1$time <= t[i] & fit1$n.event != 0]
463   d2 = fit2$n.event[fit2$time <= t[i] & fit2$n.event != 0]
464   r2 = fit2$n.risk[fit2$time <= t[i] & fit2$n.event != 0]
465   tcalc1 = t[i] - fit1$time
466   tcalc1[tcalc1<0] = Inf
467   tcalc2 = t[i] - fit2$time
468   tcalc2[tcalc2 < 0] = Inf
469   S1 = fit1$surv[which.min(tcalc1)]
470   S2 = fit2$surv[which.min(tcalc2)]
471   if(t[i] < min(fit1$time)){
472     S1=1
473   } # END if
474   if(t[i] < min(fit2$time)){

```

```

474     S2=1
475   } # END if
476   out21 = (M_ret_vec[1] ^ 2) * n * sum(division00(d1, r1 * (r1 - d1)))
477   out22 = (M_ret_vec[2] ^ 2) * n * sum(division00(d2, r2 * (r2 - d2)))
478   out[i] = product((S1 ^ M_ret_vec[1]) * (S2 ^ M_ret_vec[2]), sqrt(out21
479     + out22))
480 } # END for
481 return(out)
482 } # END psi_hat_ret
483 teststat = function(tau1, tau2, fit, fit1, fit2, fit3, M_vec, M_ret, data,
484   alpha_vec, nboot) {
# Computes p-values, test statistics, critical values and decisions for I_n, K_n and pairwise NPLR tests
485 # Args:
486 # tau1, tau2: follow-up period [tau1,tau2] specified in study protocol, if any
487 # fit: survfit applied to the pooled data from all groups (see power_calculations.R)
488 # fit1: survfit applied to data from the 1st treatment group
489 # fit2: survfit applied to data from the 2nd treatment group
490 # fit3: survfit applied to data from the 3rd treatment group
491 # M_vec: the vector of margins M_1, M_2, M_3
492 # M_ret: the margin M in (5.1) of the paper
493 # data: an n x 3 matrix with the first column being X_{ij}, i = 1, ..., n_j, j = 1, 2, 3,
494 #       the second column being the corresponding censoring indicators,
495 #       and the third column being the corresponding group number (j for the j-th treatment group)
496 # alpha_vec: c(overall alpha level, alpha level after Bonferroni adjustment)
497 # nboot: number of bootstrap samples
498 # Returns:
499 # test_nocross: decision on whether H_{01}^c should be rejected (1) or not (0), in first step of our composite procedure
500 # test: K_n in (3.1) of the paper
501 # EL_SOcrit: critical value for K_n (corresponding to alpha_vec levels)
502 # out_sup_pval: p-value for K_n
503 # inttest_dF: I_n in (3.1) of the paper
504 # int_dFEL_SOcrit: critical value for I_n (corresponding to alpha_vec levels)
505 # out_dF_pval: p-value for I_n
506 # inttest_dF_12: I_{1n} in Section 3.3 of the paper
507 # inttest_dF_23: I_{2n} in Section 3.3 of the paper
508 # int_dFEL_SOcrit_12: quantiles based on bootstrapped values of pairwise NPLR test comparing group 1 and 2
509 # int_dFEL_SOcrit_23: quantiles based on bootstrapped values of pairwise NPLR test comparing group 1 and 2

```

```

510 #   out_dF_12$pval: p-value for the pairwise NPLR test comparing group 1
511 #   and 2
512 #   out_dF_23$pval: p-value for the pairwise NPLR test comparing group 2
513 #   and 3
514 #   inttest_dF_ret: the pairwise NPLR statistic comparing group 2 and 3
515 #   using retention-of-effect formulation
516 #   int_dFEL_SOcrit_ret: critical value for pairwise NPLR test comparing
517 #   group 2 and 3 using retention-of-effect approach
518 #   out_dF_ret_pval: p-value for the pairwise NPLR test comparing group 2
519 #   and 3 using retention-of-effect formulation
520 nn = c(fit1$n, fit2$n, fit3$n)
521 ##### finding t \in [t_1, t_2] that are ordered observed uncensored times
522 T_11 = min(fit1$time[fit1$n.event != 0])
523 T_21 = min(fit2$time[fit2$n.event != 0])
524 T_31 = min(fit3$time[fit3$n.event != 0])
525 T_ms = 1:3 * 0
526 T_ms_all = 1:3 * 0
527 T_ms[1] = max(fit1$time[fit1$n.event != 0])
528 T_ms_all[1] = max(fit1$time)
529 T_ms[2] = max(fit2$time[fit2$n.event != 0])
530 T_ms_all[2] = max(fit2$time)
531 T_ms[3] = max(fit3$time[fit3$n.event != 0])
532 T_ms_all[3] = max(fit3$time)
533 fit_time_restrict_boot = (fit$n.event != 0 & fit$time <= tau2 & fit$time
534     >= tau1) # ordered observed uncensored times
535 mm = length(fit$time[fit_time_restrict_boot])
536 lowerb = max(T_11, T_21, T_31) # data driven t_1, as in Remark 3 of
537     Theorem 1
538 upperb = min(T_ms) # data driven t_2, as in Remark 3 of Theorem 1
539 upperb_which = which.min(T_ms)
540 lowerbindx_boot = which.min(abs(fit$time[fit_time_restrict_boot] -
541     lowerb))
542 upperbindx_boot = which.min(abs(fit$time[fit_time_restrict_boot] -
543     upperb))
544 if (T_ms[upperb_which] == T_ms_all[upperb_which]) {
545     upperbindx_boot = upperbindx_boot - 1
546 } # END if
547 Td_sort_boot = (fit$time[fit_time_restrict_boot])[lowerbindx_boot:
548     upperbindx_boot] # t \in [t_1, t_2] that are ordered observed
549     uncensored times
550 ##### Kaplan—Meier estimates for each treatment group and related
551     quantities for the first step of our composite procedure:
552 S1_hat = (((c(1, fit1$surv)[cumsum(c(0, fit$time) %in% c(0, fit1$time))-
553     ])[-1])[fit_time_restrict_boot])[lowerbindx_boot:upperbindx_boot]
554 S2_hat = (((c(1, fit2$surv)[cumsum(c(0, fit$time) %in% c(0, fit2$time))-
555     ])[-1])[fit_time_restrict_boot])[lowerbindx_boot:upperbindx_boot]
556 S3_hat = (((c(1, fit3$surv)[cumsum(c(0, fit$time) %in% c(0, fit3$time))-
557     ])[-1])[fit_time_restrict_boot])[lowerbindx_boot:upperbindx_boot]

```

```

543 diff_12 = M_vec[1] * log(S1_hat) - M_vec[2] * log(S2_hat)
544 diff_23 = M_vec[2] * log(S2_hat) - M_vec[3] * log(S3_hat)
545 ### computing bootstrap related quantities for the first treatment group:
546 nobd1 = length(rep(fit1$time, times = fit1$n.event))
547 data1_big = matrix(rep(rep(fit1$time, times = fit1$n.event), times = mm),
548 , byrow = TRUE, nrow = mm)
549 fit_time_1_big = matrix(rep(fit$time[fit_time_restrict_boot], times =
550 nobd1), byrow = FALSE, ncol = nobd1)
551 Ind1t_big = (data1_big <= fit_time_1_big)
552 Ind1x = rep(fit1$n.risk, times = fit1$n.event)
553 Ind1x_big = matrix(rep(Ind1x, times = mm), byrow = TRUE, nrow = mm)
554 Gs_1 = matrix(rnorm(nobd1 * nboot), nrow = nboot, ncol = nobd1)
555 Imuw_BIG_1 = array(rep(Ind1t_big / Ind1x_big, each = nboot), c(nboot, mm
556 , nobd1))
557 Gs_1_BIG = array(matrix(rep(t(Gs_1), each = mm), byrow = TRUE, nrow =
558 nboot), c(nboot, mm, nobd1))
559 sum_DWknGImuw_1_big = sqrt(sum(nn)) * apply(Imuw_BIG_1 * Gs_1_BIG, c(1,
560 2), sum)
561 Imuw_centered_BIG_1 = sqrt(sum(nn)) * Imuw_BIG_1 * Gs_1_BIG - array(rep(
562 sum_DWknGImuw_1_big / nn[1], times = nobd1), c(nboot, mm, nobd1))
563 sum_WknImuw2_1_big = apply(Imuw_centered_BIG_1 ^ 2, c(1, 2), sum)
564 ### computing bootstrap related quantities for the second treatment group:
565 nobd2 = length(rep(fit2$time, times = fit2$n.event))
566 data2_big = matrix(rep(rep(fit2$time, times = fit2$n.event), times = mm),
567 , byrow = TRUE, nrow = mm)
568 fit_time_2_big = matrix(rep(fit$time[fit_time_restrict_boot], times =
569 nobd2), byrow = FALSE, ncol = nobd2)
570 Ind2t_big = (data2_big <= fit_time_2_big)
571 Ind2x = rep(fit2$n.risk, times = fit2$n.event)
572 Ind2x_big = matrix(rep(Ind2x, times = mm), byrow = TRUE, nrow = mm)
573 Gs_2 = matrix(rnorm(nobd2 * nboot), nrow = nboot, ncol = nobd2)
574 Imuw_BIG_2 = array(rep(Ind2t_big / Ind2x_big, each = nboot), c(nboot, mm
575 , nobd2))
576 Gs_2_BIG = array(matrix(rep(t(Gs_2), each = mm), byrow = TRUE, nrow =
577 nboot), c(nboot, mm, nobd2))
578 sum_DWknGImuw_2_big = sqrt(sum(nn)) * apply(Imuw_BIG_2 * Gs_2_BIG, c(1,
579 2), sum)
580 Imuw_centered_BIG_2 = sqrt(sum(nn)) * Imuw_BIG_2 * Gs_2_BIG - array(rep(
581 sum_DWknGImuw_2_big / nn[2], times = nobd2), c(nboot, mm, nobd2))
582 sum_WknImuw2_2_big = apply(Imuw_centered_BIG_2 ^ 2, c(1, 2), sum)
583 nobd3 = length(rep(fit3$time, times = fit3$n.event))
584 ### computing bootstrap related quantities for the third treatment group:
585 data3_big = matrix(rep(rep(fit3$time, times = fit3$n.event), times = mm),
586 , byrow = TRUE, nrow = mm)
587 fit_time_3_big = matrix(rep(fit$time[fit_time_restrict_boot], times =
588 nobd3), byrow = FALSE, ncol = nobd3)
589 Ind3t_big = (data3_big <= fit_time_3_big)
590 Ind3x = rep(fit3$n.risk, times = fit3$n.event)

```

```

577 Ind3x_big = matrix(rep(Ind3x, times = mm), byrow = TRUE, nrow = mm)
578 Gs_3 = matrix(rnorm(nobd3 * nboot), nrow = nboot, ncol = nobd3)
579 Imuw_BIG_3 = array(rep(Ind3t_big / Ind3x_big, each = nboot), c(nboot, mm
580 , nobd3))
580 Gs_3_BIG = array(matrix(rep(t(Gs_3), each = mm), byrow = TRUE, nrow =
581 nboot), c(nboot, mm, nobd3))
581 sum_DWknGImuw_3_big = sqrt(sum(nn)) * apply(Imuw_BIG_3 * Gs_3_BIG, c(1,
582 2), sum)
582 Imuw_centered_BIG_3 = sqrt(sum(nn)) * Imuw_BIG_3 * Gs_3_BIG - array(rep(
583 sum_DWknGImuw_3_big / nn[3], times = nobd3), c(nboot, mm, nobd3))
583 sum_WknImuw2_3_big = apply(Imuw_centered_BIG_3 ^ 2, c(1, 2), sum)
584 ### bootstrap components in the limiting distribution in Theorem 1:
585 theta_hat_js = theta_hat_j(Td_sort_boot, fit, fit1, fit2, fit3, M_vec)
586 sigma2_hat_overpjs = sigma2_hat_overpj(Td_sort_boot, fit, fit1, fit2,
587 fit3, M_vec)
587 Dsqtheta1_big = matrix(rep(1 / sqrt(theta_hat_js[, 1])), times = nboot),
588 byrow = TRUE, nrow = nboot)
588 Dsqsigmadp1_big = matrix(rep(1 / sqrt(sigma2_hat_overpjs[, 1])), times =
589 nboot), byrow = TRUE, nrow = nboot)
589 Ujps = array(0, c(3, nboot, upperbindx_boot - lowerbindx_boot + 1))
590 Ujps[1, , ] = product_mat(as.matrix(-sum_DWknGImuw_1_big[, lowerbindx_
591 boot:upperbindx_boot]), Dsqsigmadp1_big)
591 Dsqtheta2_big = matrix(rep(1 / sqrt(theta_hat_js[, 2])), times = nboot),
592 byrow = TRUE, nrow = nboot)
592 Dsqsigmadp2_big = matrix(rep(1 / sqrt(sigma2_hat_overpjs[, 2])), times =
593 nboot), byrow = TRUE, nrow = nboot)
593 Ujps[2, , ] = product_mat(as.matrix(-sum_DWknGImuw_2_big[, lowerbindx_boot
594 :upperbindx_boot]), Dsqsigmadp2_big)
594 Dsqtheta3_big = matrix(rep(1 / sqrt(theta_hat_js[, 3])), times = nboot),
595 byrow = TRUE, nrow = nboot)
595 Dsqsigmadp3_big = matrix(rep(1 / sqrt(sigma2_hat_overpjs[, 3])), times =
596 nboot), byrow = TRUE, nrow = nboot)
596 Ujps[3, , ] = product_mat(as.matrix(-sum_DWknGImuw_3_big[, lowerbindx_
597 boot:upperbindx_boot]), Dsqsigmadp3_big)
597 wjs = array(0, c(3, nboot, upperbindx_boot - lowerbindx_boot + 1))
598 wjs_denom = theta_hat_js[, 1] * theta_hat_js[, 2] + theta_hat_js[, 1] *
599 theta_hat_js[, 3] + theta_hat_js[, 2] * theta_hat_js[, 3]
599 wjs[1, , ] = matrix(rep(division00(theta_hat_js[, 2] * theta_hat_js[, 3],
600 wjs_denom), times = nboot), byrow = TRUE, nrow = nboot)
600 wjs[2, , ] = matrix(rep(division00(theta_hat_js[, 1] * theta_hat_js[, 3],
601 wjs_denom), times = nboot), byrow = TRUE, nrow = nboot)
601 wjs[3, , ] = matrix(rep(division00(theta_hat_js[, 1] * theta_hat_js[, 2],
602 wjs_denom), times = nboot), byrow = TRUE, nrow = nboot)
602 pava_time1 = Sys.time()
603 pava_result = array(unlist(mapply(pava, alply(division00(Ujps, sqrt(wjs)
604 ), c(2, 3)), alply(wjs, c(2, 3)), decreasing = TRUE)), c(3, nboot,
upperbindx_boot - lowerbindx_boot + 1))
pava_time2 = Sys.time()

```

```

605 avg_Ujps = apply(sqrt(wjs) * Ujps, c(2, 3), sum)
606 avg_Ujps_karray = array(rep(avg_Ujps, each = 3), c(3, nboot, upperbindx_
607 boot - lowerbindx_boot + 1))
608 Up2_boot_H1_1sided = apply(wjs * (pava_result - avg_Ujps_karray)^2, c
609 (2, 3), sum) # bootstrap SSB(t) in Theorem 1
610 #### bootstrap sup_{t \in [t_1, t_2]} SSB(t) in Theorem 1:
611 sup_boot_H1 = apply(as.matrix(Up2_boot_H1_1sided), 1, max) # bootstrap
612 sup_{t \in [t_1, t_2]} SSB(t) in Theorem 1
613 EL_SOcrit = as.vector(quantile(sup_boot_H1, probs = 1 - alpha_vec)) #
614 quantiles based on bootstrapped values of sup_{t \in [t_1, t_2]} SSB(
615 t)
616 #### quantities related to bootstrap \int_{t_1}^{t_2} SSB(t) dF_0(t) in
617 Theorem 1:
618 S1_hat_Wei = ((c(1, fit1$surv)[cumsum(c(0, fit$time) %in% c(0, fit1$time
619 ))])[-1])[fit_time_restrict_boot]
620 S2_hat_Wei = ((c(1, fit2$surv)[cumsum(c(0, fit$time) %in% c(0, fit2$time
621 ))])[-1])[fit_time_restrict_boot]
622 S3_hat_Wei = ((c(1, fit3$surv)[cumsum(c(0, fit$time) %in% c(0, fit3$time
623 ))])[-1])[fit_time_restrict_boot]
624 S1_hat_M1 = S1_hat_Wei ^ M_vec[1]
625 S2_hat_M2 = S2_hat_Wei ^ M_vec[2]
626 S3_hat_M3 = S3_hat_Wei ^ M_vec[3]
627 S1_hat_M1_big = matrix(rep(S1_hat_M1, time = nboot), byrow = TRUE, nrow
628 = nboot)
629 S2_hat_M2_big = matrix(rep(S2_hat_M2, time = nboot), byrow = TRUE, nrow
630 = nboot)
631 S3_hat_M3_big = matrix(rep(S3_hat_M3, time = nboot), byrow = TRUE, nrow
632 = nboot)
633 psi_hat_js = psi_hat_j(fit$time[fit_time_restrict_boot], fit, fit1, fit2
634 , fit3, M_vec)
635 vjs = array(0, c(3, nboot, mn))
636 vjs_denom = psi_hat_js[, 1] * psi_hat_js[, 2] + psi_hat_js[, 1] * psi_
637 hat_js[, 3] + psi_hat_js[, 2] * psi_hat_js[, 3]
638 vjs[1, , ] = matrix(rep(division00(psi_hat_js[, 2] * psi_hat_js[, 3],
639 vjs_denom), times = nboot), byrow = TRUE, nrow = nboot)
640 vjs[2, , ] = matrix(rep(division00(psi_hat_js[, 1] * psi_hat_js[, 3],
641 vjs_denom), times = nboot), byrow = TRUE, nrow = nboot)
642 vjs[3, , ] = matrix(rep(division00(psi_hat_js[, 1] * psi_hat_js[, 2],
643 vjs_denom), times = nboot), byrow = TRUE, nrow = nboot)
644 F_123_big = 1 - (vjs[1, , ] * S1_hat_M1_big + vjs[2, , ] * S2_hat_M2_big
645 + vjs[3, , ] * S3_hat_M3_big)
646 Up2_boot_H1_1sided_times_dF = Up2_boot_H1_1sided * t(apply(cbind(0, F_
647 123_big), 1, diff))[, lowerbindx_boot:upperbindx_boot]
648 int_dF_boot_H1 = apply(as.matrix(Up2_boot_H1_1sided_times_dF), 1, sum)
649 # bootstrap \int_{t_1}^{t_2} SSB(t) dF_0(t) in Theorem 1
650 int_dFEL_SOcrit = as.vector(quantile(int_dF_boot_H1, probs = 1 - alpha_
651 vec)) # quantiles based on bootstrapped values of \int_{t_1}^{t_2}
652 SSB(t) dF_0(t) in Theorem 1

```

```

631 ##### quantities related to bootstrap \int_{t_1}^{t_2} SSB(t) dF_0(t) in
632 Theorem 1 using retention-of-effect approach in supplementary material
633 Section S.4:
634
635 M_ret_vec = c(M_ret - 1, 1, M_ret)
636 S1_hat_M1_ret = S1_hat_Wei ^ M_ret_vec[1]
637 S2_hat_M2_ret = S2_hat_Wei ^ M_ret_vec[2]
638 S3_hat_M3_ret = S3_hat_Wei ^ M_ret_vec[3]
639 S1_hat_M1_ret_big = matrix(rep(S1_hat_M1_ret, time = nboot), byrow =
640 TRUE, nrow = nboot)
641 S2_hat_M2_ret_big = matrix(rep(S2_hat_M2_ret, time = nboot), byrow =
642 TRUE, nrow = nboot)
643 S3_hat_M3_ret_big = matrix(rep(S3_hat_M3_ret, time = nboot), byrow =
644 TRUE, nrow = nboot)
645 psi_hat_12 = psi_hat_ret(fit$time[fit_time_restrict_boot], fit, fit1,
646 fit2, M_ret_vec)
647 vjs_ret = array(0, c(2, nboot, mm))
648 vjs_ret_denom = psi_hat_12 + psi_hat_js[, 3]
649 vjs_ret[1, , ] = matrix(rep(dvision00(psi_hat_js[, 3], vjs_ret_denom),
650 times = nboot), byrow = TRUE, nrow = nboot)
651 vjs_ret[2, , ] = matrix(rep(dvision00(psi_hat_12, vjs_ret_denom), times
652 = nboot), byrow = TRUE, nrow = nboot)
653 F_123_ret_big = 1 - (vjs_ret[1, , ] * S1_hat_M1_ret_big * S2_hat_M2_ret_
654 big + vjs_ret[2, , ] * S3_hat_M3_ret_big)
655 theta_hat_js_ret = theta_hat_j(Td_sort_boot, fit, fit1, fit2, fit3, M_
656 ret_vec)
657 theta1_big = matrix(rep(theta_hat_js_ret[, 1], times = nboot), byrow =
658 TRUE, nrow = nboot)
659 theta2_big = matrix(rep(theta_hat_js_ret[, 2], times = nboot), byrow =
660 TRUE, nrow = nboot)
661 theta3_big = matrix(rep(theta_hat_js_ret[, 3], times = nboot), byrow =
662 TRUE, nrow = nboot)
663 phi_big = theta1_big + theta2_big + theta3_big
664 localstat_limit_1_ret = ((theta1_big * Ujps[1, , ] + sqrt(theta1_big *
665 theta2_big) * Ujps[2, , ] - sqrt(theta1_big * theta3_big) * Ujps[3, ,
666 ]) ^ 2) / (phi_big ^ 2)
667 localstat_limit_2_ret = ((sqrt(theta1_big * theta2_big) * Ujps[1, , ] +
668 theta2_big * Ujps[2, , ] - sqrt(theta2_big * theta3_big) * Ujps[3, ,
669 ]) ^ 2) / (phi_big ^ 2)
670 localstat_limit_3_ret = ((sqrt(theta1_big * theta3_big) * Ujps[1, , ] +
671 sqrt(theta2_big * theta3_big) * Ujps[2, , ] - theta3_big * Ujps[3, ,
672 ]) ^ 2) / (phi_big ^ 2)
673 localstat_limit_2_ret_unsq = -sqrt(sum(nn)) * (M_ret_vec[2] ^ 2) * (sqrt(
674 (theta1_big) * Ujps[1, , ] + sqrt(theta2_big) * Ujps[2, , ] - sqrt(
675 theta3_big) * Ujps[3, , ]) / phi_big
676 localstat_limit_ret_times_dF = (localstat_limit_2_ret_unsq <= 0) * (
677 localstat_limit_1_ret + localstat_limit_2_ret + localstat_limit_3_ret
678 ) * t(apply(cbind(0, F_123_ret_big), 1, diff))[, lowerbindx_boot:
679 upperbindx_boot]

```

```

655 int_dF_boot_ret = apply(as.matrix(localstat_limit_ret_times_dF), 1, sum)
656     # bootstrap \int_{t_1}^{t_2} SSB(t) dF_0(t) in Theorem 1 using
657     retention-of-effect approach
658 int_dFEL_SOcrit_ret = as.vector(quantile(int_dF_boot_ret, probs=1 -
659     alpha_vec)) # quantiles based on bootstrapped values of \int_{t_1}^{t_2} SSB(t) dF_0(t) in Theorem 1 using retention-of-effect approach
660 #### quantities related to bootstrap combined pairwise NPLR test in Section
661     3.3 of the paper
662 theta12_big = matrix(rep(theta_hat_js[, 1] + theta_hat_js[, 2], times =
663     nboot), byrow = TRUE, nrow = nboot)
664 theta23_big = matrix(rep(theta_hat_js[, 2] + theta_hat_js[, 3], times =
665     nboot), byrow = TRUE, nrow = nboot)
666 theta13_big = matrix(rep(theta_hat_js[, 1] + theta_hat_js[, 3], times =
667     nboot), byrow = TRUE, nrow = nboot)
668 localstat_limit_12 = as.matrix((pmax(M_vec[1] * sum_DWknGImuw_1_big - M_
669     vec[2] * sum_DWknGImuw_2_big, 0) ^ 2)[, lowerbindx_boot:upperbindx_
670     boot]) / theta12_big
671 localstat_limit_23 = as.matrix((pmax(M_vec[2] * sum_DWknGImuw_2_big - M_
672     vec[3] * sum_DWknGImuw_3_big, 0) ^ 2)[, lowerbindx_boot:upperbindx_
673     boot]) / theta23_big
674 localstat_limit_13 = as.matrix((pmax(M_vec[1] * sum_DWknGImuw_1_big - M_
675     vec[3] * sum_DWknGImuw_3_big, 0) ^ 2)[, lowerbindx_boot:upperbindx_
676     boot]) / theta13_big
677 localstat_limit_12_times_dF = localstat_limit_12 * t(apply(cbind(0, F_
678     123_big), 1, diff))[, lowerbindx_boot:upperbindx_boot]
679 localstat_limit_23_times_dF = localstat_limit_23 * t(apply(cbind(0, F_
680     123_big), 1, diff))[, lowerbindx_boot:upperbindx_boot]
681 localstat_limit_13_times_dF = localstat_limit_13 * t(apply(cbind(0, F_
682     123_big), 1, diff))[, lowerbindx_boot:upperbindx_boot]
683 int_dF_boot_12 = apply(as.matrix(localstat_limit_12_times_dF), 1, sum)
684 int_dF_boot_23 = apply(as.matrix(localstat_limit_23_times_dF), 1, sum)
685 int_dF_boot_13 = apply(as.matrix(localstat_limit_13_times_dF), 1, sum)
686 int_dFEL_SOcrit_12 = as.vector(quantile(int_dF_boot_12, probs = 1 -
687     alpha_vec)) # quantiles based on bootstrapped values of pairwise
688     NPLR test comparing group 1 and 2
689 int_dFEL_SOcrit_23 = as.vector(quantile(int_dF_boot_23, probs = 1 -
690     alpha_vec)) # quantiles based on bootstrapped values of pairwise
691     NPLR test comparing group 2 and 3
692 int_dFEL_SOcrit_13 = as.vector(quantile(int_dF_boot_13, probs = 1 -
693     alpha_vec)) # quantiles based on bootstrapped values of pairwise
694     NPLR test comparing group 1 and 3
695 int_dFEL_SOcrit_infadj = as.vector(quantile(pmin(int_dF_boot_12, int_dF_
696     boot_23), probs = 1 - alpha_vec))
697 #### functions related to the nonparametric likelihood ratio statistics:
698 neg2logR = function(t, fit1, fit2, fit3, M_vec, EL_CBCrit = 0) {
699 # Computes negative two log nonparametric likelihood ratio (in Section
700     2.2 of the paper)
701 # - EL_CBCrit, for a given number EL_CBCrit at a given time t

```

```

678 #   Args:
679 #     t: the given t >= 0 in localized EL statistic
680 #     fit1: survfit applied to data from the 1st treatment group (see
681 #           power_calculations.R)
682 #     fit2: survfit applied to data from the 2nd treatment group
683 #     fit3: survfit applied to data from the 3rd treatment group
684 #     M_vec: the vector of margins M_1, M_2, M_3
685 #     EL_CBcrit: the given number EL_CBcrit (default is 0)
686 #
687 #   Returns:
688 #     the negative two log nonparametric likelihood ratio (in Section 2.2
689 #     of the paper)
690 #
691 #     - EL_CBcrit, for a given number EL_CBcrit at a given time t
692 d1 = fit1$n.event[fit1$time <= t & fit1$n.event != 0]
693 r1 = fit1$n.risk[fit1$time <= t & fit1$n.event != 0]
694 d2 = fit2$n.event[fit2$time <= t & fit2$n.event != 0]
695 r2 = fit2$n.risk[fit2$time <= t & fit2$n.event != 0]
696 d3 = fit3$n.event[fit3$time <= t & fit3$n.event != 0]
697 r3 = fit3$n.risk[fit3$time <= t & fit3$n.event != 0]
698 A1 = r1 - d1
699 A2 = r2 - d2
700 A3 = r3 - d3
701 D1 = max(d1 - r1) / M_vec[1] + 0.0001
702 D2 = max(d2 - r2) / M_vec[2] + 0.0001
703 D3 = max(d3 - r3) / M_vec[3] + 0.0001
704 init_lambda1 = median(c(D1, -D2 - D3))
705 init_lambda2 = median(c(D2 + init_lambda1, -D3))
706 num_neg_loglik = neg_log_likelihood_le_t_eq_h(t, fit1, fit2, fit3, M_
707     vec, init_lambdas = c(init_lambda1, init_lambda2))
708 denom_neg_loglik = neg_log_likelihood_le_t_ineq_h(t, fit1, fit2, fit3,
709     M_vec, init_lambdas = c(init_lambda1, init_lambda2))
710 num_warn <- tryCatch(neg_log_likelihood_le_t_eq_h(t, fit1, fit2, fit3,
711     M_vec, init_lambdas = c(init_lambda1, init_lambda2)), error =
712     function(e) e, warning = function(w) w)
713 denom_warn <- tryCatch(neg_log_likelihood_le_t_ineq_h(t, fit1, fit2,
714     fit3, M_vec, init_lambdas = c(init_lambda1, init_lambda2)), error =
715     function(e) e, warning = function(w) w)
716 init_lambda1_grid = seq(D1 + 0.0001, -D2 - D3 - 0.0001, length = 100)
717 init_dir1 = 1
718 init_wid1 = ((-D2 - D3 - 0.0001) - (D1 + 0.0001)) / 10
719 while (sum(abs(num_neg_loglik$resultEES) >= 0.0001) != 0 | sum(denom_
720     neg_loglik$resultEES >= 10 ^ -8) != 0 | is(num_warn, "warning") * is
721     (denom_warn, "warning") != 0) {
722     init_lambda1 = init_lambda1 + init_wid1 * init_dir1 * ((-1) ^ init_
723         dir1)
724     if (init_lambda1 > -D2 - D3 - 0.0001 | init_lambda1 < D1 + 0.0001)
725         break
726     init_lambda2 = median(c(D2 + init_lambda1, -D3))

```

```

713 num_neg_loglik = neg_log_likelihood_le_t_eq_h(t, fit1, fit2, fit3, M_
714   _vec, init_lambdas = c(init_lambda1, init_lambda2))
715 denom_neg_loglik = neg_log_likelihood_le_t_ineq_h(t, fit1, fit2,
716   fit3, M_vec, init_lambdas = c(init_lambda1, init_lambda2))
717 num_warn <- tryCatch(neg_log_likelihood_le_t_eq_h(t, fit1, fit2,
718   fit3, M_vec, init_lambdas = c(init_lambda1, init_lambda2)), error
719   = function(e) e, warning = function(w) w)
720 denom_warn <- tryCatch(neg_log_likelihood_le_t_ineq_h(t, fit1, fit2,
721   fit3, M_vec, init_lambdas = c(init_lambda1, init_lambda2)),
722   error = function(e) e, warning = function(w) w)
723 init_dir1 = init_dir1 + 1
724 }
725 } # END while
726 still_error = (sum(abs(num_neg_loglik$resultEEs) >= 0.0001) != 0 | sum
727   (denom_neg_loglik$resultEEs >= 10^-8) != 0 | is(num_warn, "warning"
728     ) * is(denom_warn, "warning") != 0)
729 test1t = 2 * num_neg_loglik$objective - 2 * denom_neg_loglik$objective
730 return(list(out = test1t - EL_CBcrit,
731             still_error = still_error,
732             num_lambdas = num_neg_loglik$lambdas,
733             denom_lambdas = denom_neg_loglik$lambdas
734           ))
735 } # END neg2logR
736 neg2logR_adjpair = function(lambda0_hat, t, fit1, fit2, EL_CBcrit, M_
737   vec2) {
738 # Computes negative two log nonparametric likelihood ratio (in Section
739 # 3.3 of the paper)
740 # - EL_CBcrit, for the j-th and (j+1)-th treatment groups,
741 # for a given number EL_CBcrit at a given time t
742 # Args:
743 #   lambda0_hat: the Lagrange multiplier in the numerator of equation
744 #     (3.4) in the paper
745 #   t: the given t >= 0 in localized EL statistic
746 #   fit1: survfit applied to data from the j-th treatment group in
747 #     equation (3.4) in the paper
748 #   fit2: survfit applied on data from the (j+1)-th treatment group in
749 #     equation (3.4) in the paper
750 #   EL_CBcrit: the given number EL_CBcrit (default is 0)
751 #   M_vec2: the vector of margins M_j, M_{j+1}
752 # Returns:
753 #   the negative two log nonparametric likelihood ratio (in Section 3.3
754 #   of the paper)
755 # - EL_CBcrit, for the j-th and (j+1)-th treatment groups,
756 # for a given number EL_CBcrit at a given time t
757 d1 = fit1$n.event[fit1$time <= t & fit1$n.event != 0]
758 r1 = fit1$n.risk[fit1$time <= t & fit1$n.event != 0]
759 d2 = fit2$n.event[fit2$time <= t & fit2$n.event != 0]
760 r2 = fit2$n.risk[fit2$time <= t & fit2$n.event != 0]
761 A1 = r1 - d1

```

```

747 A2 = r2 - d2
748 B1 = d1 / (r1 + M_vec2[1] * lambda0_hat) * as.numeric(r1 != d1) + d1 /
749     (d1 + M_vec2[1] * lambda0_hat) * as.numeric(r1 == d1)
750 B2 = d2 / (r2 - M_vec2[2] * lambda0_hat) * as.numeric(r2 != d2) + d2 /
751     (d2 - M_vec2[2] * lambda0_hat) * as.numeric(r2 == d2)
752 test1t = -2 * sum(d1 * log(B1)) - 2 * sum(product(A1, log(1 - B1))) -
753     2 * sum(d2 * log(B2)) - 2 * sum(product(A2, log(1 - B2))) + 2 * sum
754     (d1 * log(d1 / r1)) + 2 * sum(d2 * log(d2 / r2)) + 2 * sum(product(
755     A1, log(1 - d1 / r1))) + 2 * sum(product(A2, log(1 - d2 / r2)))
756 return(test1t - EL_CBcrit)
757 } # END neg2logR_adjpair
758 neg2logR_ret = function(lambda0_hat, t, fit1, fit2, fit3, EL_CBcrit, M_
759     ret) {
760 # Computes negative two log nonparametric likelihood ratio (in Section S
761 # .4 of
762 # the supplementary material) - EL_CBcrit, for a given number EL_CBcrit
763 # at a given time t
764 # Args:
765 #     lambda0_hat: the Lagrange multiplier in the first display in p. 17
766 #     of the supplementary material
767 #     t: the given t >= 0 in localized EL statistic
768 #     fit1: survfit applied to data from the 1st treatment group (see
769 #     power_calculations.R)
770 #     fit2: survfit applied to data from the 2nd treatment group
771 #     fit3: survfit applied to data from the 3rd treatment group
772 #     EL_CBcrit: the given number EL_CBcrit (default is 0)
773 #     M_ret: the margin M in (5.1) of the paper
774 # Returns:
775 #     the negative two log nonparametric likelihood ratio (in Section S.4
776 #     of
777 #     the supplementary material) - EL_CBcrit, for a given number EL_
778 #     CBcrit at a given time t
779 d1 = fit1$n.event[fit1$time <= t & fit1$n.event != 0]
780 r1 = fit1$n.risk[fit1$time <= t & fit1$n.event != 0]
781 d2 = fit2$n.event[fit2$time <= t & fit2$n.event != 0]
782 r2 = fit2$n.risk[fit2$time <= t & fit2$n.event != 0]
783 d3 = fit3$n.event[fit3$time <= t & fit3$n.event != 0]
784 r3 = fit3$n.risk[fit3$time <= t & fit3$n.event != 0]
785 A1 = r1 - d1
786 A2 = r2 - d2
787 A3 = r3 - d3
788 B1 = d1 / (r1 + (M_ret - 1) * lambda0_hat) * as.numeric(r1 != d1) + d1 /
789     (d1 + (M_ret - 1) * lambda0_hat) * as.numeric(r1 == d1)
790 B2 = d2 / (r2 + lambda0_hat) * as.numeric(r2 != d2) + d2 / (d2 +
791     lambda0_hat) * as.numeric(r2 == d2)
792 B3 = d3 / (r3 - M_ret * lambda0_hat) * as.numeric(r3 != d3) + d3 / (d3 -
793     M_ret * lambda0_hat) * as.numeric(r3 == d3)

```

```

779 test1t = -2 * sum(d1 * log(B1)) - 2 * sum(product(A1, log(1 - B1))) -
    2 * sum(d2 * log(B2)) - 2 * sum(product(A2, log(1 - B2))) - 2 * sum(
    (d3 * log(B3)) - 2 * sum(product(A3, log(1 - B3))) + 2 * sum(d1 *
    log(d1 / r1)) + 2 * sum(d2 * log(d2 / r2)) + 2 * sum(d3 * log(d3 /
    r3)) + 2 * sum(product(A1, log(1 - d1 / r1))) + 2 * sum(product(A2,
    log(1 - d2 / r2))) + 2 * sum(product(A3, log(1 - d3 / r3)))
780 return(test1t - EL_CBcrit)
781 } # END neg2logR_ret
782 ### computing quantities related to the NPLR statistics:
783 teststat_pre = 1:length(Td_sort_boot) * 0
784 error_vec = 1:length(Td_sort_boot) * 0
785 teststat_pre_12 = 1:length(Td_sort_boot) * 0
786 teststat_pre_23 = 1:length(Td_sort_boot) * 0
787 teststat_pre_13 = 1:length(Td_sort_boot) * 0
788 teststat_pre_ret = 1:length(Td_sort_boot) * 0
789 for (j in 1:(upperbindx_boot - lowerbindx_boot + 1)) {
790   t = Td_sort_boot[j] # the given t >= 0 in localized EL statistic
791   neg2logRt = neg2logR(t, fit1, fit2, fit3, M_vec, EL_CBcrit = 0)
792   error_vec[j] = neg2logRt$still_error
793   if (error_vec[j] == 1) next
794   teststat_pre[j] = neg2logRt$out # the -2 log NPLR statistic in
      Section 2.2 of the paper at the given t >=0 specified above
795   lambda0_hat_12 = lambda0(t, fit1, fit2, tilde_theta = 1, M_vec[1:2])
796   lambda0_hat_23 = lambda0(t, fit2, fit3, tilde_theta = 1, M_vec[2:3])
797   lambda0_hat_13 = lambda0(t, fit1, fit3, tilde_theta = 1, M_vec[c(1, 3)
      ])
798   lambda0_hat_ret = lambda0_ret(t, fit1, fit2, fit3, tilde_theta = 1, M_
      ret)
799   if (lambda0_hat_ret < 0) { # else teststat_pre_ret[j] is 0
800 # the -2 log NPLR statistic in Section S.4 of the supplementary material
      at the given t >=0 specified above
801     teststat_pre_ret[j] = neg2logR_ret(lambda0_hat_ret, t, fit1, fit2,
      fit3, EL_CBcrit = 0, M_ret)
802   } # END if
803   if (lambda0_hat_12 < 0) { # else teststat_pre_12[j] is 0
804 # the -2 log NPLR statistic for j=1 vs 2 in Section 3.3 of the paper at
      the given t >=0 specified above
805     teststat_pre_12[j] = neg2logR_adjpair(lambda0_hat_12, t, fit1, fit2,
      EL_CBcrit = 0, M_vec[1:2])
806   } # END if
807   if (lambda0_hat_23 < 0) { # else teststat_pre_23[j] is 0
808 # the -2 log NPLR statistic for j=2 vs 3 in Section 3.3 of the paper at
      the given t >=0 specified above
809     teststat_pre_23[j] = neg2logR_adjpair(lambda0_hat_23, t, fit2, fit3,
      EL_CBcrit = 0, M_vec[2:3])
810   } # END if
811   if (lambda0_hat_13 < 0) { # else teststat_pre_13[j] is 0

```

```

812 # the -2 log NPLR statistic for j=3 vs 1 in Section 3.3 of the paper at
813 # the given t >=0 specified above
814 teststat_pre_13[j]=neg2logR_adjpair(lambda0_hat_13,t,fit1,fit3,EL_
815 CBcrit=0,M_vec[c(1,3)])
816 } # END if
817 } # END for
818 F_123 = 1 - (vjs[1, 1, ] * S1_hat_M1 + vjs[2, 1, ] * S2_hat_M2 + vjs[3,
819 1, ] * S3_hat_M3)
820 inttest_pre_dF = teststat_pre * diff(c(0, F_123))[lowerbindx_boot:
821 upperbindx_boot] # -2logR(t) d\hat{F}_0(t) in (3.1) of the paper
822 inttest_pre_dF_12 = teststat_pre_12 * diff(c(0, F_123))[lowerbindx_boot:
823 upperbindx_boot]
824 inttest_pre_dF_23 = teststat_pre_23 * diff(c(0, F_123))[lowerbindx_boot:
825 upperbindx_boot]
826 inttest_dF_12 = sum(inttest_pre_dF_12) #I_{1n} in Section 3.3 of the
827 paper
828 inttest_dF_23 = sum(inttest_pre_dF_23) #I_{2n} in Section 3.3 of the
829 paper
830 inttest_dF_infadj = min(sum(inttest_pre_dF_12), sum(inttest_pre_dF_23))
831 F_123_ret = 1 - (vjs_ret[1, 1, ] * S1_hat_M1_ret * S2_hat_M2_ret + vjs_
832 ret[2, 1, ] * S3_hat_M3_ret)
833 inttest_pre_dF_ret = teststat_pre_ret * diff(c(0, F_123_ret))[lowerbindx_
834 _boot:upperbindx_boot]
835 inttest_dF_ret = sum(inttest_pre_dF_ret) #I_n, retention of effect
836 version, as in Section S.4 of the supplementary material
837 inttest_dF_ret_infadj = min(sum(inttest_pre_dF_12), sum(inttest_pre_dF_
838 ret))

#### computing quantities related to the first step of our composite
procedure:
828 test_nocross = 0
829 HW_CBdistr = pmax(apply(abs(M_vec[2] * sum_DWknGImuw_2_big - M_vec[1] *
830 sum_DWknGImuw_1_big), 1, max), apply(abs(M_vec[3] * sum_DWknGImuw_3 -
831 big - M_vec[2] * sum_DWknGImuw_2_big), 1, max))
832 HW_CBCrit = as.vector(quantile(HW_CBdistr, 0.95))
833 HW_CB_ubs_12 = diff_12 + HW_CBCrit / sqrt(sum(nn))
834 HW_CB_ubs_23 = diff_23 + HW_CBCrit / sqrt(sum(nn))
835 HW_CB_lbs_12 = diff_12 - HW_CBCrit / sqrt(sum(nn))
836 HW_CB_lbs_23 = diff_23 - HW_CBCrit / sqrt(sum(nn))
837 S1gS2eqS3 = (sum(HW_CB_lbs_12 <= 0) != (upperbindx_boot - lowerbindx_
838 boot + 1) & sum(HW_CB_lbs_23 <= 0) == (upperbindx_boot - lowerbindx_
839 boot + 1) & sum(HW_CB_ubs_23 >= 0) == (upperbindx_boot - lowerbindx_
840 boot + 1))
841 S1eqS2gS3 = (sum(HW_CB_lbs_23 <= 0) != (upperbindx_boot - lowerbindx_
842 boot + 1) & sum(HW_CB_lbs_12 <= 0) == (upperbindx_boot - lowerbindx_
843 boot + 1) & sum(HW_CB_ubs_12 >= 0) == (upperbindx_boot - lowerbindx_
844 boot + 1))
845 S1geS2geS3 = (sum(HW_CB_ubs_12 >= 0) == (upperbindx_boot - lowerbindx_
846 boot + 1) & sum(HW_CB_ubs_23 >= 0) == (upperbindx_boot - lowerbindx_
847 boot + 1))

```

```

boot + 1))
838 S1gS2gS3 = (sum(HW_CB_ubs_12 >= 0) == (upperbindx_boot - lowerbindx_boot
+ 1) & sum(HW_CB_ubs_23 >= 0) == (upperbindx_boot - lowerbindx_boot
+ 1) & sum(HW_CB_lbs_12 <= 0) != (upperbindx_boot - lowerbindx_boot
+ 1) & sum(HW_CB_lbs_23 <= 0) != (upperbindx_boot - lowerbindx_boot +
1))
839 S1eqS2eqS3 = (sum(HW_CB_lbs_12 <= 0) == (upperbindx_boot - lowerbindx_
boot + 1) & sum(HW_CB_ubs_12 >= 0) == (upperbindx_boot - lowerbindx_
boot + 1) & sum(HW_CB_lbs_23 <= 0) == (upperbindx_boot - lowerbindx_
boot + 1) & sum(HW_CB_ubs_23 >= 0) == (upperbindx_boot - lowerbindx_
boot + 1))
840 if (S1gS2gS3 | S1eqS2eqS3) {
841   test_nocross = 1
842 }
843 return(list(
844   test_nocross = test_nocross,
845   test = max(teststat_pre), # K_n in (3.1) of the paper
846   EL_SOcrit = EL_SOcrit,
847   out_sup_pval = mean(sup_boot_H1 >= max(teststat_pre)), # p-value for
     the NPLR test based on K_n in (3.1) of the paper
848   inttest_dF = sum(inttest_pre_dF), # I_{n} in (3.1) of the paper
849   int_dFEL_SOcrit = int_dFEL_SOcrit,
850   out_dF_pval = mean(int_dF_boot_H1 >= sum(inttest_pre_dF)), # p-value
     for the NPLR test based on I_n in (3.1) of the paper
851   inttest_dF_12 = inttest_dF_12,
852   inttest_dF_23 = inttest_dF_23,
853   int_dFEL_SOcrit_12 = int_dFEL_SOcrit_12,
854   int_dFEL_SOcrit_23 = int_dFEL_SOcrit_23,
855   out_dF_12_pval = mean(int_dF_boot_12 >= inttest_dF_12), # p-value for
     the pairwise NPLR test comparing group 1 and 2
856   out_dF_23_pval = mean(int_dF_boot_23 >= inttest_dF_23), # p-value for
     the pairwise NPLR test comparing group 2 and 3
857   inttest_dF_ret = inttest_dF_ret,
858   int_dFEL_SOcrit_ret = int_dFEL_SOcrit_ret,
859   out_dF_ret_pval = mean(int_dF_boot_ret >= inttest_dF_ret) # p-value
     for the pairwise NPLR test comparing group 2 and 3 using retention-
     of-effect formulation
860 )) # END return
861 } # END teststat

```

functions_to_be_sourced.R

S.7 References

- Chang, H.-w. and McKeague, I. W. (2016). Empirical likelihood based tests for stochastic ordering under right censorship. *Electronic Journal of Statistics*, 10(2):2511–2536.
- Landau, S. and Stahl, D. (2013). Sample size and power calculations for medical studies by simulation when closed form expressions are not available. *Statistical Methods in Medical Research*, 22(3):324–345.
- Li, G. (1995). On nonparametric likelihood ratio estimation of survival probabilities for censored data. *Statistics & Probability Letters*, 25:95–104.
- Usmani, R. A. (1994). Inversion of a tridiagonal Jacobi matrix. *Linear Algebra and its Applications*, 212:413–414.