

NetSurvival.jl: A glimpse into relative survival analysis.

JuliaCon 2024 @ Eindhoven

Rim Alhajal and Oskar Laverny

July 10, 2024

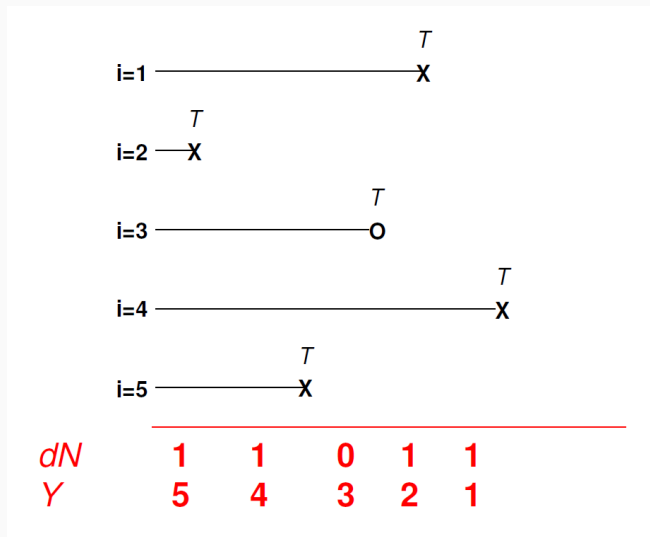
¹ Aix Marseille Univ, INSERM, IRD, SESSTIM, Sciences Economiques & Sociales de la Santé & Traitement de l'Information Médicale, ISSPAM, Marseille, France.

1. Introduction to relative survival analysis
2. Standard estimators in the field
3. Comparison with the R/C++ standard
4. Real data showcase
5. Conclusion

Introduction to relative survival analysis

Survival Analysis is a statistical theory that deals with censored positive random variables:

- (i) Events (usually death) can occur or not before censoring (exit of experiment)
- (ii) We want to model the time to event (survival)
- (iii) Relies on counting processes and local martingales.



Relative Survival Context: In population-based studies, the specific cause of death is often **unidentified, unreliable or even unavailable**. We then work with the following random variables:

Random Variable	Name	Observed ?
E	"Excess" lifetime	No
P	"Population" lifetime	No, but known distribution.
$O = E \wedge P$	"Overall" lifetime	No
C	"Censoring" time	No
D	Vector of covariates	Yes
$T = O \wedge C$	Event time	Yes
$\Delta = \mathbb{1}\{T \leq C\}$	Event status	Yes
$\mathbb{1}\{E \leq P\}$	Cause of death	No

Notations: We use both $S_X(t) = \mathbb{P}(X > t)$, $\Lambda_X = -\ln S_X$ or $\lambda_X = \partial\Lambda_X$ to characterize a distribution.

Goal: Estimate the distribution of E , say by it's hazard $\partial\Lambda_E(t) = -\partial\ln S_E(t)$.

Remark: Without the cause of death indicatrix, standard competing risks models cannot be used...

Remark: The joint distribution of (E, P, C, D) characterizes our observations.

Assumptions (Standard independence¹)

- (i) C, E and (P, D) are mutually independent
- (ii) The distribution of $P|D$ is known from standard life tables (at time 0) for each individual.
- (iii) Individuals are i.i.d.

Note: The population hazard for each individual λ_{P_i} is drawn from a reference RateTable from RateTables.jl and may depend on covariates D_1, \dots, D_p such as age, year, sex, country, race, etc...

¹Maja Pohar Perme, Janez Stare, and Jacques Estève. "On Estimation in Relative Survival". In: *Biometrics* 68.1 (June 2011), pp. 113–120. ISSN: 0006-341X. DOI: 10.1111/j.1541-0420.2011.01640.x.

Standard estimators in the field

The estimation of net survival is usually discussed in terms of the estimation of the cumulative excess hazard $\Lambda_E(t)$ and/or the instantaneous hazard $\lambda_E = \partial\Lambda_E$.

Definition (Pohar Perme 2012²)

The Pohar Perme estimator of the excess hazard is given, using $w_i(s) = S_{P_i}(s)^{-1}$, by:

$$\partial\hat{\Lambda}_E(s) = \frac{\sum_{i=1}^n w_i(s) \{\partial N_i(s) - Y_i(s)\partial\Lambda_{P_i}(s)\}}{\sum_{i=1}^n w_i(s) Y_i(s)}$$

and its variance can be estimated by:

$$\partial\hat{\sigma}_E^2(s) = \frac{\sum_{i=1}^n w_i(s)^2 \partial N_i(s)}{(\sum_{i=1}^n w_i(s)^2 Y_i(s))^2}$$

Good properties: This estimator is **biased** but **convergent** and asymptotically unbiased.

²Maja Pohar Perme, Janez Stare, and Jacques Estève. "On Estimation in Relative Survival". In: *Biometrics* 68.1 (June 2011), pp. 113–120. ISSN: 0006-341X. DOI: 10.1111/j.1541-0420.2011.01640.x.

One big advantage with Julia is the how easy to read and comprehensible the syntax is.

```
const PoharPerme = NPNEstimator{PoharPermeMethod}

function !!(::Type{PoharPermeMethod}, ∂Ne, Ye, ∂Np, Yp, ∂V, T, Δ, age, date, rate_preds, ratetable, grid, Δt)
    for i in eachindex(age)
        Ti = searchsortedlast(grid, T[i])
        Λp, wp, rti = 0.0, 1.0, ratetable[rates_preds[i,:]]...
        for j in 1:Ti
            λp = daily_hazard(rti, age[i] + grid[j], date[i] + grid[j])
            ∂Λp = λp * ∂t[j]
            Λp += ∂Λp
            wp = exp(Λp)
            ∂Np[j] += ∂Λp * wp
            Ye[j] += wp
        end
        ∂Ne[Ti] += wp * Δ[i]
        ∂V[Ti] += wp2 * Δ[i]
    end
    Yp .= Ye
end
```

Figure 1: The Pohar Perme function in the NetSurvival.jl package looks like the formula itself.

Note: The daily_hazard function is thoroughly optimized. 60% of runtime in exp !

The Graéo log-rank test³ is designed to compare net survival functions across multiple groups. There is a stratified version of this test as well.

Definition (Graéo log-rank test)

The null H_0 hypothesis tests the following assumption:

$$\forall t \in [0, T], \Lambda_{E, g_1}(t) = \Lambda_{E, g_2}(t) = \dots = \Lambda_{E, g_k}(t)$$

where $G = \{g_1, \dots, g_k\}$ is a partition of $1, \dots, n$ consisting of disjoint groups of individuals that we wish to compare to each other.

³Nathalie Graéo, Fabienne Castell, Aurélien Belot, and Roch Giorgi. "A Log-Rank-Type Test to Compare Net Survival Distributions". In: *Biometrics* 72.3 (Jan. 2016), pp. 760–769. ISSN: 0006-341X. DOI: 10.1111/biom.12477.

Definition (Graïó log-rank test)

For all groups $g \in G$, let's denote the numerator and denominator of the Pohar Perme (partial) excess hazard estimators, restricted to individuals in the group, by:

$$\partial N_{E,g}(s) = \sum_{i \in g} \frac{\partial N_i(s)}{S_{P_i}(s)} - \frac{Y_i(s)}{S_{P_i}(s)} \partial \Lambda_{P_i}(s)$$

$$Y_{E,g}(s) = \sum_{i \in g} \frac{Y_i(s)}{S_{P_i}(s)}$$

$$R_g(s) = \frac{Y_{E,g}(s)}{\sum_{g \in G} Y_{E,g}(s)}$$

Then, define the vector $Z = (Z_{g_r} : r \in 1, \dots, k-1)$ with entries:

$$Z_g(T) = N_{E,g}(s) - \int_0^T Y_{E,g}(s) \partial \hat{\Lambda}_E(s)$$

Definition (Gra éo log-rank test)

The test statistic is then given by:

$$U(T) = Z(T)' \hat{\Sigma}_Z^{-1} Z(T)$$

where the entries of the $\hat{\Sigma}_Z$ matrix are given by:

$$\hat{\sigma}_{g,h}(T) = \int_0^T \sum_{\ell \in G} (\delta_{g,\ell} - R_g(t)) (\delta_{h,\ell} - R_h(t)) \left(\sum_{i \in \ell} \frac{\partial N_i(s)}{S_{P_i}^2} \right)$$

Under H_0 , the statistic $U(T)$ is asymptotically $\chi^2(k-1)$ -distributed. We reject the H_0 hypothesis when the p-value obtained is under a certain value depending on the error rate chosen, thus admitting the notable difference between the groups.

Definition

The crude mortality rate is the global mortality rate for a population. It can be separated by cause of death, such as:

$$M_E(t) = \int_0^t S_O(u-) \partial \Lambda_E(u)$$

Note: We denote $M_P(t) = \int_0^t S_O(u-) \partial \Lambda_P(u) du$ and $F_O(t) = M_E(t) + M_P(t)$.

Lemma (The Cronin-Feuer estimator⁴)

To estimate the quantity above, we first introduce the Cronin-Feuer estimator given by:

$$\hat{M}_E(t) = \int_0^t \hat{S}_O(u-) \partial \hat{\Lambda}_E(u)$$

⁴Kathleen A Cronin and Eric J Feuer. "Cumulative cause-specific mortality for cancer patients in the presence of other causes: a crude analogue of relative survival". In: *Statistics in medicine* 19.13 (2000), pp. 1729–1740.

The `nessie` function allows us to calculate the estimated sample size by yearly intervals as well as the average lifespan left.

Definition (Estimated Sample Size or ESS)

The estimated sample size is given by: $ESS(t) = \sum_i^N S_{P_i}(t)$

Definition (expected lifespan)

Due to the constance of the hazard rates on each cell of the lifetable, the life expectation can be computed through the following formula⁵:

$$E(P) = \int_0^{\infty} S_P(t) \partial t = \sum_{j=0}^{\infty} \frac{S_P(t_j)}{\lambda_P(t_j)} \left(1 - e^{-\lambda_P(t_j)(t_{j+1}-t_j)}\right)^{-1}$$

Note: This feature is implemented in the `RateTables.jl` package and depends on the `Distributions.expectation` function.

⁵Per Kragh Andersen. "Life years lost among patients with a given disease". In: *Statistics in medicine* 36.22 (2017), pp. 3573–3582.

Comparison with the R/C++ standard

With Julia, the code is concise and easy to read, half of it is docs:

NetSurvival.jl files

- ▼ NETSURVIVAL
- > .github
- > .vscode
- > benchmark
- > data
- > docs
- > build
- ▼ src
- ▼ assets
- # citations.css
- ≡ references.bib
- ↓ benches.md
- ↓ example.md
- ↓ getting_started.md
- ↓ index.md
- ↓ references.md
- 🔗 make.jl
- ⚙ Manifest.toml
- ⚙ Project.toml
- ▼ src
- 🔗 CrudeMortality.jl
- 🔗 Ederer.jl
- 🔗 EdererII.jl
- 🔗 fetch_datasets.jl
- 🔗 GraffeoTest.jl
- 🔗 Hakulinen.jl
- 🔗 Levey.jl
- 🔗 NetSurvival.jl

```

src > 🔗 CrudeMortality.jl > ...
1  """
2  |   CrudeMortality
3  |   _____
4  |   This method calculates the crude mortality and presents both the excess morta
5  |
6  |   The default Cronin-Feuer estimator can be fitted to data with the following i
7  |
8  |   fit(CrudeMortality, args...)
9  |
10 | where the `args` are passed to `fit(EdererII,args...)` to compute the excess
11 |
12 | A more direct syntax can be used, specifying directly the estimator for the e
13 |
14 |   CrudeMortality(fit(EdererII,args...))
15 |
16 | To use another excess hazard estimator, simply replace `EdererII` with the me
17 |
18 | References:
19 | * [cronin2000cumulative](@cite) Cronin, Kathleen A and Feuer, Eric J (2000).
20 | """
21 | struct CrudeMortality
22 |     M₀::Vector{Float64}
23 |     M₁::Vector{Float64}
24 |     M₂::Vector{Float64}
25 |     function CrudeMortality(npe::NPNEstimator{Method}) where Method
26 |         S₀ = cumprod(1 .- npe.∂Λ₀)
27 |         M₁ = cumsum(npe.∂Λ₁ .* S₀)
28 |         M₂ = cumsum(npe.∂Λ₂ .* S₀)
29 |         return new(M₀ .+ M₁, M₁, M₂)
30 |     end
31 | end

```


With Julia, the code is concise and easy to read, half of it is docs:

NetSurvival.jl files

- NETSURVIVAL
- > .github
- > .vscode
- > benchmark
- > data
- > docs
- > build
- > src
 - assets
 - # citations.css
 - references.bib
 - benches.md
 - example.md
 - getting_started.md
 - index.md
 - references.md
 - make.jl
 - Manifest.toml
 - Project.toml
 - src
 - CrudeMortality.jl
 - Ederer.jl
 - Ederer.jl
 - fetch_datasets.jl
 - GraffeoTest.jl
 - Hakulinen.jl
 - Nessie.jl
 - NetSurvival.jl
 - NPNEstimator.jl
 - PoharPerme.jl
 - Surv_and_Strata.jl
 - test
 - runtests.jl
 - sampletest.jl
 - ignore
 - CITATION.bib

```

src > Nessie.jl > Nessie > Nessie
1  struct Nessie
2      expected_sample_size::Vector{Float64}
3      expected_life_time::Float64
4      grid::Vector{Float64}
5      function Nessie(T, Δ, age, year, rate_preds, ratetable)
6          annual_grid = 0:RateTables.RT_DAYS_IN_YEAR:maximum(T)
7          exp_spl_size = zeros(length(annual_grid))
8          exp_life_time = 0.0
9          for i in eachindex(age)
10             Pi = Life(ratetable[rate_preds[i,:].], age[i], year[i])
11             for j in eachindex(annual_grid)
12                 exp_spl_size[j] += cdf(Pi, annual_grid[j])
13             end
14             exp_life_time += expectation(Pi)
15         end
16         return new(exp_spl_size, exp_life_time / RateTables.RT_DAYS_IN_YEAR / length(age), annual_grid)
17     end
18 end
19
20 ---
21     nessie
22
23     To call this function, use the formula below:
24
25     nessie(@formula(Surv(time,status)~covariate), data, ratetable)
26 ---
27 function nessie(args...)
28     r = fit(Nessie,args...)
29     if typeof(r)<:Nessie
30         return r
31     end
32     transform!(r, :estimator => ByRow(x-> (x.grid, x.expected_life_time, x.expected_sample_size)) => [:gr:
33     select!(r, Not(:estimator))
34
35     lt = deepcopy(r)
36     select!(lt, Not(:expected_sample_size, :grid))
37
38     select!(r, Not(:expected_life_time))
39
40     end
    
```

On the other hand, most of the R and C++ files from R: : rel surv span over thousands of lines of code:

Rcode.r file

```

RELSURV_2.2-9
├─ relsurv
│ └─ data
│   ├── .Rhistory
│   ├── colrec.rda
│   ├── rdata.rda
│   ├── slopop.rda
│   └─ inst
│     ├── man
│     └─ R
│       ├── cmprel.r
│       ├── mystata.r
│       ├── plotssurv.r
│       └─ Rcode.r
│         ├── rformulate.r
│         ├── rsdiff.r
│         ├── rssurvrsadd.r
│         ├── survfitsadd.r
│         ├── years.R
│         └─ zzz.R
│           └─ src
│             ├── cmpfast.c
│             ├── dmatrix.c
│             ├── exps.c
│             ├── init.c
│             ├── netfast.c
│             ├── netfastpinter.c
│             ├── netfastpinter2.c
│             ├── netwei.c
│             ├── netweibm.c
│             └─ ruster.c
relsurv > R > Rcode.r
1 rsfitterem<-function(data,b,maxiter,ratetable,tol,bwin,p,cause,Nie){
2   # cause: = 2 (unknown), 0 in 1 known. Lahko preko argumenta cause v rsadd dolocis, c
3   # Nie: to je lambda_0 (t1), ki se oceni v M koraku v EM algoritmu
4
5   pr.time<-proc.time()[3]
6   if (maxiter<1) stop("There must be at least one iteration run")
7   n<-nrow(data)
8   m <- p
9   dtimes <- which(data$stat==1) #the positions of event times in data$Y
10  td <- data$Y[dtimes] #event times
11  ntd <- length(td) #number of event times
12  utimes <- which(c(1,diff(td))!=0) #the positions of unique event times among td
13  utd <- td[utimes] #unique event times
14  nutd <- length(utd) #number of unique event times
15  udtimes <- dtimes[utimes] #the positions of unique event times among data$Y
16  razteg <- function(x){
17    # x is a 0/1 vector, the output is a vector of length sum(x), with the correspondi
18    n <- length(x)
19    repu <- rep(1,n)
20    repu[x==1] <- 0
21    repu <- rev(cumsum(rev(repu)))
22    repu <- repu[x==1]
23    repu <- -diff(c(repu,0))+1
24    if(sum(repu)!=n)repu <- c(n-sum(repu),repu) #ce je prvi cas censoring, bo treba s
25    repu
26  }
27  rutd <- rep(0,ntd)
28  rutd[utimes] <- 1
29  nutd <- razteg(rutd) #from unique event times to event times
30  rntd <- razteg(data$stat) #from event times to data$Y
31
32  a <- data$a[data$stat==1]
33  #the R/C++ standard
34  if(bwin[1]!=0){

```

On the other hand, most of the R and C++ files from R: : rel surv span over thousands of lines of code:

Rcode.r file

```

reلسurv > R > Rcode.r
{
  348
  385
  386 if(missing(bwin))bwin <- -1
  387 if(bwin<0){
  388
  389   if(p>0)data1 <- data[,-c(varstart:varstop)] #NEW IN 2.05
  390   else data1 <- data
  391   nfk <- length(attributes(rform$ratetable)$dimid)
  392   names(data)[4:(3+nfk)] <- attributes(rform$ratetable)$dimid
  393   expe <- rs.surv(Surv(Y,stat)*1,data,ratetable=rform$ratetable,method="ederer2")
  394   esurv <- -log(expe$surv[expe$n.event!=0])
  395   if(esurv[length(esurv)]==Inf)esurv[length(esurv)] <- esurv[length(esurv)-1]
  396   x <- seq(.1,3,length=5)
  397   dif <- rep(NA,5)
  398   options(warn=-1)
  399   diter <- max(round(max(data$Y)/356.24),3)
  400   for(it in 1:5){
  401     fit <- rsfitterem(data1,NULL,diter,rform$ratetable,control$epsilon,x[it],0,rform$cause,Nie)
  402     dif[it] <- sum((esurv-fit$Lambda0)^2)
  403   }
  404   wh <- which.min(dif)
  405   if(wh==1)x <- seq(x[wh],x[wh+1]-.1,length=5)
  406   else if(wh==5)x <- c(x, max(data$Y)/ max(diff(data$Y)))
  407   if(wh!=1)
  408     x <- seq(x[wh-1]+.1,x[wh+1]-.1,length=5)
  409   dif <- rep(NA,5)
  410
  411   for(it in 1:5){
  412     fit <- rsfitterem(data1,NULL,diter,rform$ratetable,control$epsilon,x[it],0,rform$cause,Nie)
  413     dif[it] <- sum((esurv-fit$Lambda0)^2)
  414   }
  415   options(warn=0)
  416   Nie <- fit$Nie
  417   bwin <- x[which.min(dif)]
  418 }
  419
  420 fit <- rsfitterem(data, beta, control$maxit, rform$ratetable,
  421 control$epsilon, bwin, p, rform$cause, Nie)
  422
  423 Nie <- rep(0, nrow(data))

```


On the other hand, most of the R and C++ files from R: : rel surv span over thousands of lines of code:

Rcode.r file

```

reلسurv > R > Rcode.r
3626 expprep2 <- function (x, y, ratetable, status, times, fast=FALSE, ys, prec, cmp=F, netweidM=FALSE) { #func
3684   if(!missing(status)){ #the function was called from rs.surv
3689     if (any(times < 0))
3690       stop("Negative time point requested")
3691     ntime <- length(times)
3692     if(missing(ys)) ys <- rep(0,length(y))
3693     # times2 <- times
3694     # times2[1] <- prec1
3695     if(cmp) temp <- .Call("cmpfast", as.integer(rfac), #fast=pohar-perme or ederer2 - data from p
3696                          as.integer(attn$dim), as.double(unlist(cuts)), ratetable,
3697                          x, y, ys, as.integer(status), times, PACKAGE="reلسurv")
3698     else if(fast&!missing(prec)) temp <- .Call("netfastpinter2", as.integer(rfac), #fast=pohar-pe
3699                          as.integer(attn$dim), as.double(unlist(cuts)), ratetab
3700                          x, y, ys, as.integer(status), times, prec, PACKAGE="reلسurv")
3701     else if(fast&missing(prec)) temp <- .Call("netfastpinter", as.integer(rfac), #fast=pohar-perm
3702                          as.integer(attn$dim), as.double(unlist(cuts)), ratetabl
3703                          x, y, ys, as.integer(status), times, PACKAGE="reلسurv")
3704     else if(netweidM==TRUE) temp <- .Call("netweidM", as.integer(rfac),
3705                          as.integer(attn$dim), as.double(unlist(cuts)), ratetable,
3706                          x, y, ys, as.integer(status), times, PACKAGE="reلسurv")
3707     else temp <- .Call("netwei", as.integer(rfac),
3708                          as.integer(attn$dim), as.double(unlist(cuts)), ratetable,
3709                          x, y, as.integer(status), times, PACKAGE="reلسurv")
3710   }
3711   else{ #only expected survival at time y is needed for each individual
3712     if(length(y)!=1)y <- rep(y,nrow(x))
3713     if(length(y)!=nrow(x)) stop("Wrong length for status")
3714     temp <- .Call("expc", as.integer(rfac),
3715                  as.integer(attn$dim), as.double(unlist(cuts)), ratetable,
3716                  x, y, PACKAGE="reلسurv")
3717     temp <- temp$Surv
3718   }
3719   temp
3720 }
3721

```

On the other hand, most of the R and C++ files from R: : rel surv span over thousands of lines of code:

C code

```

relsurv > src > C cmpfast.c
39  SEXP cmpfast(  SEXP  efac2,  SEXP  edims2,
201  /*
210  etime = thiscell;
211  hazard =0;
212  hazspi=0;           //integration of haz/si
213  while (etime >0) {
214  et2 = pystep(edim, &indx, &indx2, &wt, data2, efac,
215  | edims, ecut, etime, 1);
216  hazspi+= et2* expect[indx]/(si[i]*exp(-hazard)); //add the integrated part
217  if (wt <1) hazard+= et2*(wt*expect[indx] +(1-wt)*expect[indx2]);
218  else hazard+= et2* expect[indx];
219  for (k=0; k<edim; k++)
220  | if (efac[k] !=1) data2[k] += et2;
221  etime -= et2;
222  }
223  sitt[i] = si[i];           // si at the beginning of the interval
224  si[i] = si[i]* exp(-hazard);
225
226
227  if(ys[i]<times[j]){ // if start of observation before this time
228  | yisi[j]+=1/si[i];
229  | yisitt[j]+=1/sitt[i];
230  | yidlisi[j]+=hazard/si[i];
231  | yidli[j]+=hazard;
232  | yi[j]+=1;
233  | if(y[i]==times[j]){
234  | | dnisij[j]+=status[i]/si[i];
235  | | dni[j]+=status[i];
236  | | dnisisq[j]+=status[i]/(si[i]*si[i]);
237  | } // if this person died at this time
238  | } // if start of observation before this time
239  | } // if still at risk
240  } // loop through individuals
241
242
243  dLambdap[j]=yidli[j]/yi[j];
244  dLambdao[j]=dni[j]/yi[j];
245  dLambdaj[j]=dnisij[j] - dLambdap[j];
246
247

```

On the other hand, most of the R and C++ files from R: : rel surv span over thousands of lines of code:

C code

```

relsurv > src > C netfastpinter2.c
171 //for (j=0; j<2 ; j++) {
225 */
229 ** The wt parameter only comes into play for older style US rate
230 ** tables, where pystep does interpolation.
231 ** Each call to pystep moves up to the next 'boundary' in the
232 ** expected table, data2 contains our current position therein
233 */
234
235 /* while (etime >0) {*/ //this loop is needed if changes can happen between the interval poi
236 et2 = pystep2(edim, &indx, &indx2, &wt, data2, efac, edims, ecut, fthiscell, 1);
237 lambdapi = expect[indx];
238 lambdapi2 = expect[indx2];
239 if(ys[i]<times[j]){ //he has entered before the crude interval - this guy is at risk for the
240     fydilsi+= lambdapi/si[i];
241     fydilsi2+= lambdapi/(si[i]*exp(-fthiscell* lambdapi));
242     fysi+=1/si[i];
243     fysi2+=1/(si[i]*exp(-fthiscell* lambdapi));
244     if (wt <1) hazard+= fthiscell*(wt*lambdapi +(1-wt)*lambdapi2);
245     else hazard+= fthiscell* lambdapi; //length of the time interval *
246 } // if start of observation before this time
247
248 /*for (k=0; k<edim; k++)
249     if (efac[k] !=1) data2[k] += et2;*/
250 /*etime -= et2;
251 */
252
253 si[i] = si[i]* exp(-fthiscell* lambdapi); //the value of SPI at the end of this fine interval
254
255 if(ys[i]<=times[j]){ //he has entered before the crude interval - this guy is at risk for
256     sisum+=1/si[i];
257     sisumtt+=1/sitt[i];
258 }
259 if(jfine==1){ //count the number at risk only on the first fine interval
260     yi[j]+=1;
261 }
262
263 if(y[i]==times[j]){
264     dnisi[j]+=status[i]/si[i];
265     dni[j]+=status[i];
266     dnisi2[j]+=status[i]/(si[i]*si[i]);
267 } // if this person died at this time
    
```

Count Lines of Code

The cl oc software yields the following numbers for each repository:

Language	files	blank	comment	code
R	40	883	1793	6096
C	11	284	526	1222
C/C++ Header	1	11	6	25
Total	52	1178	2325	7343

Table 1: Summary of the count of lines of code for different languages in rel surv

Language	files	blank	comment	code
Julia	16	151	143	547

Table 2: Summary of the count of lines of code in NetSurvival.jl

A glimpse in our benchmarks for a few standard algorithms:

	Unstratified	Stratified
Pohar Perme ($\partial\Lambda_E$)	20.8431	20.1461
Ederer I ($\partial\Lambda_E$)	7.216	4.1363
Ederer II ($\partial\Lambda_E$)	29.2397	29.0399
Gra éo (log-rank-type test)	13.1556	18.156

Table 3: Runtime multipliers comparing NetSurvival.jl to R: relsurv, computed on a i9-13900 processor. The data used is the colrec dataset and the slopop mortality table.

Example: The Pohar Perme function on Julia takes ≈ 0.11 seconds to run whereas R takes ≈ 2.27 .

Remark: One **key** advantage the NetSurvival.jl package has on R: relsurv is the function that fetches the **daily hazard rates from life tables** and matches them to the individuals from the dataset, hosted in the JuliaSurv/RateTables.jl. *Our implementation is blazing fast w.r.t. the original C++ one.*

Real data showcase

Cohort: col rec

- (i) 5971 patients
- (ii) Colon or rectal cancer diagnosis between 1994 and 2000
- (iii) 7 variables: age, year, sex, status, follow-up time, cancer stage, and cancer site.
- (iv) Sourced from the Slovenian cancer registry

Rate table: sl opop

- (i) Slovenian mortality table
- (ii) Includes information on age, year, and sex.
- (iii) Extracted from official census mortality rates.

Let's take a closer look at those variables:

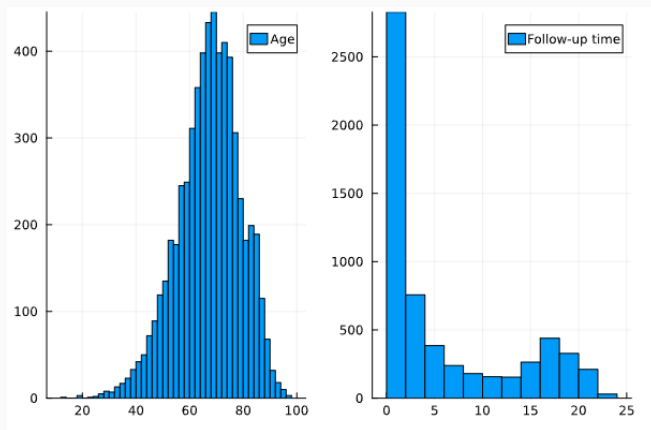


Figure 4: Histograms showing the distributions of age and the follow-up time (in years).

Given the cohort `col_rec` and the related mortality table `sl_opop`, we will apply the Pohar Perme estimator using:

Julia code

```
using NetSurvival, DataFrames, RateTables
pp_estimator = fit(PoharPerme, @formula(Surv(time, status)~1), col_rec, sl_opop)
```

By compiling the code above, we get a table with the net survival probability in daily intervals. Let's plot the results for better visualization.

Output of the Pohar Perme estimator

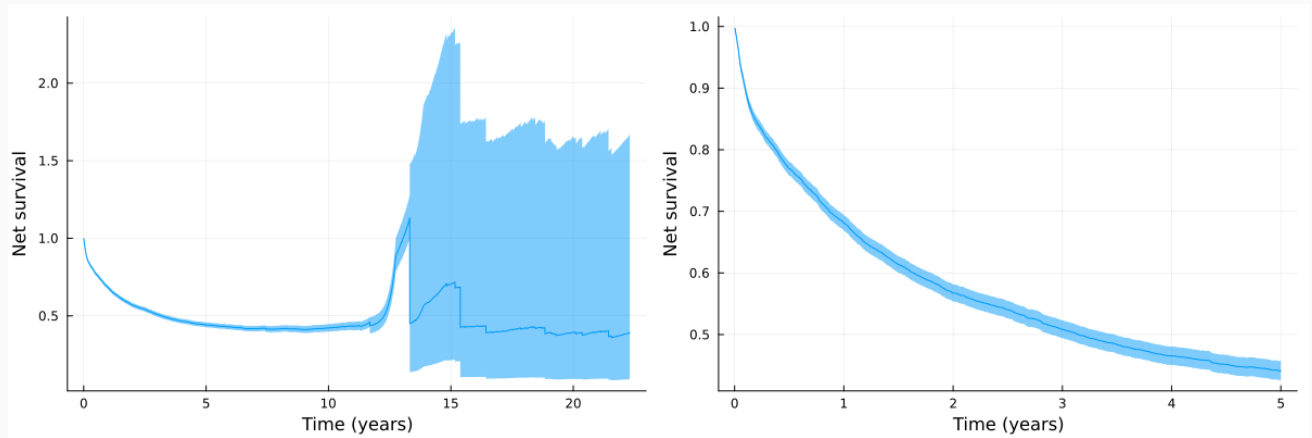


Figure 5: Pohar Perme net survival estimator. Right: only the first 5 years.

The graph above shows that, with time, the estimator loses a lot of its accuracy. Let's find out why.

Looking at the previous histograms, we can see that the age variable is important. The nessie function gives an estimated of expected sample size w.r.t. cancer only:

Julia code

```
el t, ess = nessie(@formula(Surv(time, status)~age65), col rec, sl opop)
```

Output:

Year	Young	Old
1	2352.0	3619.0
2	2323.53	3389.0
...
23	1419.49	316.588

Table 4: Estimated sample size by year for patients above and under 65 years old.

Again, for a better understanding of these values, we present them in the graph below:

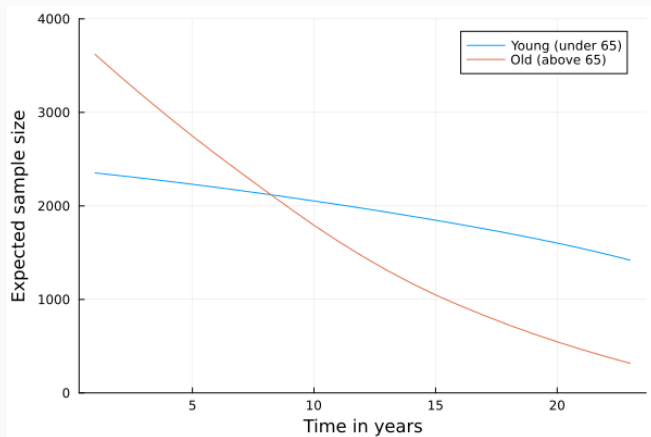


Figure 6: Graph representation of the estimated sample size for younger and older patients in yearly intervals.

Conclusion: We choose to censor the data after 5 years.

As for the expected years left for these two groups, we get the output below:

Age category	Expected life time
young	24.7882
old	10.2949

Table 5: Expected life time for patients grouped by age.

Around 5 years into the study, the crude mortality is calculated:

```
Julia code
```

```
CrudeMortality(pp_estimator5)
```

Output at time = 5 years:

$\hat{F}_O(5)$	$\hat{M}_E(5)$	$\hat{M}_P(5)$
0.645	0.531	0.114

Table 6: Crude mortality at year 5.

This shows that of the 64% patients that have died, 53% is due to colorectal cancer while 11% is due to other causes.

When applying the Graéo test, we find that only two variables carry real significance in the study.

Julia code

```
test_age = fit(GraffeoTest, @formula(Surv(time5, status5)~age65), colrec, slopop)  
test_stage = fit(GraffeoTest, @formula(Surv(time5, status5)~stage), colrec, slopop)
```

They are unsurprisingly age65 and stage, with p-values of $1.85 * 10^{-18}$ and $7.18 * 10^{-237}$ respectively for the first 5 years of the study.

Conclusion: In the col rec dataset, and with reference to the sl opop mortality table, age and cancer stages are, unsurprisingly, the more important variables in a net survival context. The older and the more advanced the cancer is, the lower the risk of survival.

Conclusion

The [JuliaSurv](#) organisation on GitHub was created to house all the packages related to survival analysis in one place in an attempt to grow the Julia general registry within that context, and to keep track of older packages.

So far, it comprises:

- (i) `NetSurvival.jl`
- (ii) `RateTables.jl`
- (iii) `SurvivalBase.jl`
- (iv) `SurvivalDistributions.jl`.

More to come in the near future!

To summarize:

- (i) Survival analysis deals with (*censored*) data where the variable of interest is the time until a certain event occurs (e.g., death).
- (ii) In some specific cases, in particular for cancer registries, the exact cause of death of each individual is unavailable and/or unreliable.
- (iii) Relative Survival is a theory built to handle this issue.
- (iv) Our `JuliaSurv` implementation is **easier to read, more concise, and faster** than R's, thus, making it future-proof.
- (v) The `JuliaSurv` organisation has a bright future ahead!

Contributions to `NetSurvival.jl` as well as to `JuliaSurv` are more than welcome!

`JuliaSurv/NetSurvival.jl`



Star it on Github :)