C++ Implementation of Time-Varying Shared Frailty Cox Models

Author: Giulia Romani
Professor: Luca Formaggia
Teaching Assistants: Matteo Caldana, Paolo Joseph Baioni
Advisors: Chiara Masci, Alessandra Ragni

POLITECNICO
MILANO 1863

Introduction

Statistical Context, Problem and Solution

Reference paper: *"Centre-Effect on Survival After Bone Marrow Transplantation: Application of Time-Dependent Frailty Models"*, by C.M. Wintrebert, H. Putter, A.H. Zwinderman and J.C. van Houwelingen.

- ✍ *Time-Varying Shared Frailty Cox Models*
- ✗ No available numerical codes
- ✗ Slowness of our R implemented codes
- ➟ $\mathrm{C}++$ models implementation and speed-up through *OpenMP*
- ➟ Application to PoliMi dataset
- ➟ Comparison of performance.

PoliMi Dataset

Dataset composed of $(n_{individuals}, n_{covariates} + 2)$. For each individual:

- Covariates: *Gender*, *Origins*, *PoliMi Test Admission Score*, *CFUP*
- *Faculty* and *time-to-event*.

Survival analysis context:

- *Event*: student's dropout
- *Follow-up*: 3 academic years (6 semesters)
- *Time-to-event*: dropout time instant during follow-up, otherwise default value outside follow-up
- *Groups*: PoliMi engineering faculties ($n_{groups} = 16$).

Several dataset related to different academic years (e.g. 2010, 2018, $2017 - 2018$).

Survival Analysis Theory

Time-Invariant Cox Model

Let T be the random time-to-event variable and t any realization of T. Define:

- *Survival function* $S(t)$: probability of surviving longer than t.
- *Hazard function* $h(t)$: instantaneous risk of facing the event, considering it is not occurred yet.

Time-Invariant Cox Model: given a set of covariates $x_{i,r}$ and unknown regressor coefficients β_r ($r \in \{1, \ldots, R\}$), hazard for individual i:

$$h_i(t, \boldsymbol{x}_i) = h_0(t) exp \left\{ \sum_{r=1}^{R} \beta_r x_{i,r} \right\}$$

where:

$h_0(t)$ baseline hazard function, $\ln(L(\boldsymbol{\beta}))$ partial log-likelihood function and $\hat{\boldsymbol{\beta}} = \arg\max_{\boldsymbol{\beta} \in R^P} \ln(L(\boldsymbol{\beta}))$.

Time-Invariant Shared Gamma-Frailty Cox Model

Data heterogeneity:

- Covariates $\rightarrow$ observed heterogeneity
- Clustered students into faculties $\rightarrow$ unobserved heterogeneity $\rightarrow$ variance θ of the frailty $Z_j \sim gamma(1/\theta, 1/\theta)$, $\theta > 0$, $j \in \{1, \ldots, n_{groups}\}$.

Time-Invariant Shared Gamma-Frailty Cox Model has hazard for individual i in faculty j:

$$h_{ij}(t|Z_j) = Z_j h_0(t) \exp(\boldsymbol{\beta}^T \mathbf{x}_{ij})$$

Unknown $\boldsymbol{\beta}$ and θ get maximizing another partial penalized log-likelihood function.

Problem: Z_j constant over time $\rightarrow$ unchangeable characteristics of the groups.

Time-Varying Shared Frailty Cox Models

Partition time-domain T into intervals I_k, $k \in \{1, \dots, L\}$, and:

1. $Z_j \rightarrow Z_{jk} \rightarrow$ *Time-Varying Shared Frailty Cox Models*
2. baseline hazard $h_0(t) \rightarrow$ interval baseline log-hazard ϕ_k, $\forall k$
3. Other variables $\rightarrow$ time-varying variables.

Hazard function h_{ijk} for individual i, in the group j, in I_k, given by:

$$h_{ijk}(t_{ij}|Z_{jk}) = Z_{jk} \ \mathrm{e}^{(\boldsymbol{x}_{ij}\boldsymbol{\beta}+\phi_k)}$$

where $\boldsymbol{\beta}$ and $\boldsymbol{\phi}$ are unknown parameters.

Only three models in this family:

- *Adapted Paik et al.'s Model*
- *Centre-Specific Frailty Model with Power Parameter*
- *Stochastic Time-Dependent Centre-Specific Frailty Model*

C++ Implementation: Basic Structures

TimeVaryingSharedFrailtyCoxModels Folder and Input Files

In *TimeVaryingSharedFrailtyCoxModels* folder:

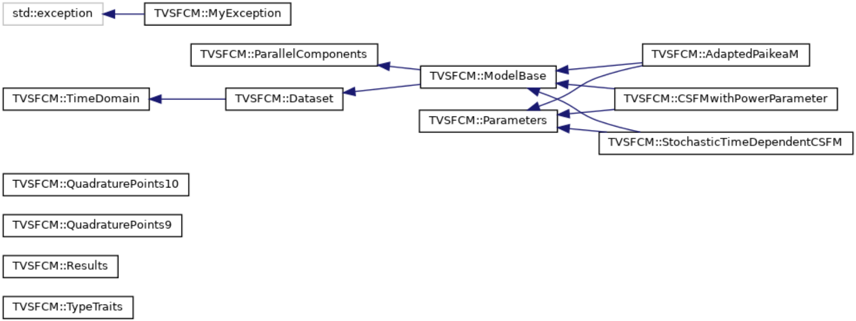
- *Src*: numerical codes
- *Data*: input .txt files in *DataTool* and *DataIndividuals*
- *BashScript*: bash scripts for running on terminal a model
- *Makefile*, *READMI.md*.

Two types of input files:

- *DataIndividualsFile.txt*: dataset
- *DataToolFile.txt*: variables for classes *Time-Domain*, *Parameters*, *Model*, *ParallelVersion*.

All read through *Getline* and *GetPot*.

Inheritance Scheme of the Classes



- *TypeTraits* struct: type aliases
- *QuadraturePoints* structs: nodes and weights for Gauss-Hermite quadrature formula
- *MyException* class: exception with input-passed message
- *Results* class: results of model application.

Time-Domain and Dataset Classes

Time-Domain class

- Time domain variables
- Base class.

Dataset class

- Publicly derived from *Time-Domain*
- Base class
- Individual dataset variables
- Map *map_groups*: (faculty, shared pointer $\rightarrow$ faculty members).
  E.g.:
  Faculty vector: $["EngA", "EngC", "EngB", "EngB", "EngA"]$.
  Map pairs: $\{("EngA", \rightarrow (0,4)), ("EngB", \rightarrow (2,3)), \dots\}$.

Certain matrices and vectors $\rightarrow$ dynamic *Eigen* matrices/vectors. E.g. dataset, parameter vector.

Parameters Class

Each time-varying model depends on n_p parameters:

- ? Same unknown β (R) and ϕ (L)
- ? Different unknown elements of Z_{jk}
- ! Some must be constrained.

collected in $\boldsymbol{p}$ and in *Parameters* class (base class).

Constrained Optimization Method in Multidimension: $\hat{\boldsymbol{p}} = \arg\max_{\boldsymbol{p} \in R^{n_p}} l(\boldsymbol{p})$

where:

- Overall log-likelihood $l(\boldsymbol{p}) = \sum_{j=1}^{N=n_{groups}} l_j(\boldsymbol{p})$
- Group log-likelihood $l_j(\boldsymbol{p})$.

Models Differences and Problems

Different Z_{jk} $\rightarrow$ different models $\rightarrow$ different:

- ➢ Resolution method of spatial integrals (gamma function vs Gauss-Hermite quadrature formula)
- ➢ $I(\boldsymbol{p})$, $I_j(\boldsymbol{p})$
- ! Execution time of I, I_j.

Problems:

- ✗ Working but inefficient optimization method
- ✗ Slowness of evaluation of I.

Therefore:

- ✄ Do not implement the optimization phase
- ☞ Use *OpenMP* to speed up the evaluation of I.

C++ Implementation: Time-Varying Models

Time-Varying Models Object Factory

Each model characterized by (Id, name), with Id $\in \{1, 2, 3\}$.

To call it use *Polymorphism*:

- Abstract *ModelBase* class
- *ModelDerived* class for each model
- *MakeLikelihoodModel(...)* to return the pointer *ptrMethod*.

```
std::unique_ptr<ModelBase>
MakeLikelihoodModel(const T::IdType id, const T::FileNameType& filename1_,
                     const T::FileNameType& filename2_) {
    switch(id){
        case 1:  return
                 std::make_unique<AdaptedPaikeaM>(filename1_, filename2_);
        case 2:  return
                 std::make_unique<CSFMwithPowerParameter>(filename1_, filename2_);
        case 3:  return
                 std::make_unique<StochasticTimeDependentCSFM>(filename1_, filename2_);
        default:  throw MyException("Not existent or not provided id method!");
    };
};
// In main.cpp, to compute overall log-likelihood
ptrMethod -> evaluate_loglikelihood();
```

ModelBase and ModelDerived Classes

ModelBase class (for all models):

- Publicly derived by *Dataset* and *ParallelComponents*
- Virtual pure methods: *evaluate_loglikelihood()*, ...

ModelDerived class (for each model):

- Publicly derived by *ModelBase* and *Parameters*
- $l(\boldsymbol{p}), l_j(\boldsymbol{p})$ implemented as lambda functions.
  E.g.: *Adapted Paik eaM*:

```
ll_paik = [this] (T::VectorXdr& v_parameters_){
    T::VariableType log_likelihood_group, log_likelihood = 0;
    T::MapType::iterator it_map = Dataset::map_groups.begin();
    T::MapType::iterator it_map_end = Dataset::map_groups.end();
    for(; it_map != it_map_end; ++it_map){
        const auto& indexes_group = it_map->second; //! Extract the shared pointer
        log_likelihood_group = ll_group_paik(v_parameters_, indexes_group);
        log_likelihood += log_likelihood_group;
    }
    return log_likelihood;
};
```

AdaptedPaikeaM Class: Group Log-likelihood

How to compute $l_j(\boldsymbol{p})$:

```
ll_group_paik = [this] (T::VectorXdr& v_parameters_, T::SharedPtrType indexes_group_){
    // Extract variables and parameters from the vector
    auto [phi, betar, mu1, mu2, nu, gammak] = extract_parameters(v_parameters_);
    auto [A_ijk, A_ik, A_i] = extract_matrixA_variables(indexes_group_, phi, betar);
    auto [d_ijk, d_ik, d_i] = extract_dropout_variables(indexes_group_);
    // Compute the first term of the formula and then subtract the second term
    T::VariableType  dataset_betar, loglik1 = 0.;
    for(const auto &i: *indexes_group_){
        dataset_betar = Dataset::dataset.row(i) * betar;
        for(T::NumberType k = 0; k < Dataset::n_intervals; ++k){
            loglik1 += (dataset_betar + phi(k)) * Dataset::dropout_intervals(i,k);
        }
    }
    loglik1 -= (mu1/nu) * log(1 + nu * A_i);
    // Compute the rest of the formula
    [... loglik2, loglik3 ...]
    // Sum the terms
    result = loglik1 + loglik2 + loglik3;
    return (result);
};
```

C++ Implementation: OpenMP

ParallelComponents Class and OpenMP

Exploiting:

- Independence of groups j
- Structure of $I = \sum_{j=1}^{N=n_{groups}} I_j$
- Map *map_groups*.

For reducing execution time of $I \rightarrow$ *OpenMP*:

- Parallel for loop
- Each thread computes at least one I_j
- All threads have access to shared memory.

ParallelComponents class:

- Number of threads
- Chunk size
- For loop scheduling strategy (*schedule_type*).

Parallel For Loop

How to implement in parallel / for *Adapted Paik eaM*:

```
ll_paik_parallel = [this] (T::VectorXdr& v_parameters_){
    T::VariableType log_likelihood = 0;              //! Overall log-likelihood value

    //! Loop over the map through an iterator
    T::MapType::iterator it_map_begin = Dataset::map_groups.begin();
    T::MapType::iterator it_map = it_map_begin;

//! Parallel region
omp_set_schedule(omp_sched_t(ParallelComponents::schedule_type),
                 ParallelComponents::chunk_size);
#pragma omp parallel for num_threads(ParallelComponents::n_threads)
#                         firstprivate(it_map)
#                         schedule(runtime)
#                         reduction(+:log_likelihood)
    for(T::IndexType j = 0; j < n_groups; ++j){
        it_map = std::next(it_map_begin, j);
        const auto& indexes_group = it_map->second;
        log_likelihood += ll_group_paik(v_parameters_, indexes_group);
    }
    return log_likelihood;
};
```

Details of Parallel For Loop

Input numeric *schedule_type* implies:

- *schedule(runtime)*
- *omp_set_schedule(...)* before OpenMP parallel directive
- *omp_sched_t(schedule_type)* to select OpenMP strategy.

Parallel for loop clauses:

- *firstprivate(it_map)*: each thread has its own copy of it_map
- *reduction(+: log_likelihood)*: each thread contributes to overall l.

Loop over the map:

- Parallel for loop $\rightarrow$ random access iterators (map iterator is not)
- Use *unsigned int* counter $j \in \{0, \ldots, n_{groups} - 1\}$
- Each thread it_map points to group j thanks to *std::next(it_map_begin, j)*.

Application

System Characteristics

Virtual Machine:

- Oracle VM VirtualBox 6.1
- Ubuntu 22.04.01 operative system
- $4GB$ RAM with 2 cores.

Installed on a *Dell Inspiron 15 5000 series*:

- Windows 10 operative system
- Intel Core i7
- $16GB$ RAM and 4 cores.

Comparison R and C++ Performance

./bash_test.sh: Test case: everything works.

./bash_app2010.sh: Same output R and C++, but faster time!

Model	Output in R	Output in C++
Adapted Paik eaM	Loglikelihood = -1498.5040 AIC = 3039.0080 $se[1:2] = [1.6523e-1, 9.1589e-2]$ $sd[1:2] = [3.6896e-1, 1.8987e-1]$ Elapsed time: $6.84s$	LogLikelihood = -1498.5044 AIC = 3039.0087 $se[1:2] = [1.6523e-1, 9.1602e-2]$ $sd[1:2] = [3.6892e-1, 1.8987e-1]$ Elapsed time: $0.0608s$
CSFM with Power Parameter	Loglikelihood = -1511.6165 AIC = 3061.2330 $se[1:2] = [1.2809e-1, 7.4126e-2]$ $sd[1:2] = [4.6784e-1, 2.1098e-7]$ Elapsed time: $14.23s$	LogLikelihood = -1511.6165 AIC = 3061.2330 $se[1:2] = [1.2810e-1, 7.4123e-2]$ $sd[1:2] = [4.6787e-1, 2.1096e-7]$ Elapsed time: $0.315s$
Stochastic Time-Dependent CSFM	Loglikelihood = -1500.4790 AIC = 3030.9580 $se[1:2] = [1.4064e-1, 8.9612e-2]$ $sd[1:2] = [3.0438e-1, 2.1617e-1]$ Elapsed time: $\approx 5min$	LogLikelihood = -1500. AIC = 3031. $se[1:2] = [1.406e-1, 8.960e-2]$ $sd[1:2] = [3.043e-1, 2.511e-1]$ Elapsed time: $35.3s$

Execution time increases with model complexity: *Adapted Paik eaM* → *CSFM with Power Parameter* → *Stochastic Time-Dependent CSFM*.

OpenMP for only Log-likelihood Evaluation

./bash_app2010.sh

Schedule	(n_threads, chunk_size)	Adapted Paik eaM	CSFM with Power Parameter	Stochastic Time-Dependent CSFM
static (id=1)	(4, 4)	0.00161s	0.0131s	0.405s
dynamic(id=2)	(4, default)	0.00152s	0.00648s	0.373s
guided(id=3)	(4, default)	0.0020s	0.0126s	0.458s
auto(id=4)	(4, default)	0.00146s	0.00713s	0.663s
serial	–	0.000998s	0.00668s	0.722s

./bash_app2018.sh

Schedule	(n_threads, chunk_size)	Adapted Paik eaM	CSFM with Power Parameter	Stochastic Time-Dependent CSFM
static (id=1)	(4, 4)	0.00443s	0.0121s	0.802s
dynamic(id=2)	(4, default)	0.0035s	0.0125s	0.691s
guided(id=3)	(4, default)	0.0041s	0.0135s	0.699s
auto(id=4)	(4, default)	0.00638s	0.0128s	0.744s
serial	–	0.00218s	0.0126s	1.37s

OpenMP for only Log-likelihood Evaluation

./bash_app201718.sh

Schedule	(n_threads, chunk_size)	Adapted Paik eaM	CSFM with Power Parameter	Stochastic Time-Dependent CSFM
static (id=1)	(4, 4)	0.00374s	0.0145s	1.34s
dynamic(id=2)	(4, default)	0.00277s	0.0116s	1.24s
guided(id=3)	(4, default)	0.00342s	0.0117s	1.28s
auto(id=4)	(4, default)	0.00355s	0.0120s	1.34s
serial	–	0.00305s	0.0201s	2.34s

Summarizing:

Adapted Paik eaM:

- ✗ : Parallel version: aligned or worse execution time (additional costs).
- ☞ : Fast and simple serial version.

OpenMP for only Log-likelihood Evaluation

CSFM with Power Parameter

☞ : Serial version: best performance when moderate dataset dimensionality (2010, 2018).

☞ : Parallel version: *dynamic* scheduling strategy when increasing dimensionality (2017 – 2018).

✗ : No *static* strategy: students not uniformly distributed in faculties.

Stochastic Time-Dependent CSFM:

☞ : Parallel version: always best performance with *dynamic* strategy.

✗ : Serial version: too slow.

Conclusion and Future Developments

Conclusion and Future Developments

Conclusion:

- ✓ C++ codes produce same R output, but faster.
- ✓ According to:
  - Chosen model
  - Dataset dimensionality
  - Serial or parallel version of log-likelihood function

  log-likelihood execution time changes → parallel version always recommended for *Stochastic Time-Dependent CSFM.*

Possible future development:

- ☞ Change optimization method and include it.
- ☞ Binding *RCpp*.

Thank You for the Attention!

References

References

[1] M Abramowitz and I Stegun. *Handbook of Mathematical Functions*. New York: Dovers Publications, 1965.

[2] *Doxygen*. URL: https://www.doxygen.nl/.

[3] *Eigen Library*. URL: https://eigen.tuxfamily.org/dox/.

[4] David Kleinbaum and Mitchell Kelin. *Survival Analysis, A Self-Learning Text, Third Edition*. Statistics for Biology and Health. Springer Science+Business Media Inc, 2012.

[5] *The OpenMP API specification for parallel programming*. URL: https://www.openmp.org/.

[6] *The R Project for Statistical Computing*. URL: https://www.r-project.org/.

[7] Claire M A Wintrebert et al. "Centre-effect on Survival after Bone Marrow Transplantation: Application of Time-dependent Frailty Models". In: *Biometrical Journal* 46.5 (2004), pp. 512–525. URL: https://doi.org/10.1002/bimj.200310051.

Appendix: Log-likelihood functions

Adapted Paik eaM

Frailty: $Z_{jk}(t_{ij}) = (\alpha_j + \epsilon_{jk})$ for $t_{ij} \in I_k$
where α_j and ϵ_{jk} independent and distributed according to:

- $\alpha_j \sim Gamma(\mu_1/\nu, 1/\nu)\ \forall j$
- $\epsilon_{jk} \sim Gamma(\mu_2/\gamma_k, 1/\gamma_k)\ \forall j, k$

with $\mu_1, \mu_2, \nu, \gamma_k(\forall k) > 0$ and $E[Z_{jk}] = \mu_1 + \mu_2 = 1$.

Frailty Variance: $var(Z_{jk}) = \mu_1\nu + \mu_2\gamma_k$

Log-likelihood $l(\boldsymbol{p})$:

$$l = \sum_{j=1}^{N}\left[\sum_{i,k} d_{ijk}(\boldsymbol{x}_{ij}\boldsymbol{\beta} + \phi_k) - \frac{\mu_1}{\nu}\log(1 + \nu A_{j..}) + \sum_k \left[\frac{-\mu_2}{\gamma_k}\log(1 + \gamma_k A_{j.k})\right]\right] +$$
$$+ \sum_{j=1}^{N}\left[\sum_k \left[\log\left(\sum_{l=0}^{d_{j.k}} \binom{d_{j,k}}{l} \frac{\Gamma(\mu_2/\gamma_k + d_{j.k} - l)}{\Gamma(\mu_2/\gamma_k)} \frac{\Gamma(\mu_1/\nu + l)}{\Gamma(\mu_1/\nu)} \frac{(A_{j.k} + 1/\gamma_k)^{(l - d_{j.k})}}{(A_{j..} + 1/\nu)^l}\right)\right]\right]$$

with: $A_{ijk} = e_{ijk}\ \mathrm{e}^{(\boldsymbol{x}_{ij}\boldsymbol{\beta} + \phi_k)}$, $A_{j.k} = \sum_i A_{ijk}$, $A_{j..} = \sum_{i,k} A_{ijk}$, $d_{j.k} = \sum_i d_{ijk}$
and *Gamma function*: $\Gamma(\zeta) = \int_0^{\infty} x^{\zeta-1}\ \mathrm{e}^{-x} dx$

CSFM with Power Parameter

Frailty: $Z_{jk}(t_{ij}) = \alpha_j^{\gamma_k} = \mathrm{e}^{Y_j \gamma_k}$ for $t_{ij} \in I_k$,
with γ_k unknown $(\forall k)$ and $Y_j = \log(\alpha_j) \sim N(0, \sigma^2)$.

Frailty Variance: $var(Z_{jk}) = e^{2\gamma_k^2 \sigma^2} - e^{\gamma_k^2 \sigma^2}$

Log-likelihood $l(\boldsymbol{p})$:

$$l = \sum_{j=1}^{N} \left[\sum_{i,k} d_{ijk} (\boldsymbol{X}_{ij} \boldsymbol{\beta} + \phi_k) \right] - \frac{N}{2} \log(\pi) + \\ + \sum_{j=1}^{N} \log \left[\sum_{q=1}^{9} w_q \mathrm{e}^{\left(\sqrt{2}\sigma \theta_q \sum_{i,k} d_{ijk} \gamma_k - \sum_{i,k} e_{ijk} \, \mathrm{e}^{\left(\sqrt{2}\sigma \gamma_k \theta_q + X_{ij} \beta + \phi_k \right)} \right)} \right]$$

Gauss-Hermite quadrature formula:
$\int_{-\infty}^{\infty} f(x) \, \mathrm{e}^{-x^2} dx = \sum_{q=1}^{Q} w_q f(\theta_q)$

Stochastic Time-Dependent CSFM

Frailty: $\log Z_j(t_{ij}) = (c_j + b_j t_{ij})$ for $t_{ij} \in [0, \infty)$,

$(c_j, b_j) \sim N_2(\mathbf{0}, \Sigma)$ and $\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_c^2 & \sigma_{cb} \\ \sigma_{cb} & \sigma_b^2 \end{bmatrix}$

Frailty Variance: $var(log(Z_{jk})) = \sigma_c^2 + \sigma_b^2 t_{ij}^2 + 2\sigma_{cb} t_{ij}$

Log-likelihood $l(\boldsymbol{p})$:

$$l = -N\log(\pi) + \sum_{j=1}^{N} \left[\sum_{i,k} d_{ijk}(\boldsymbol{X}_{ij}\boldsymbol{\beta} + \phi_k) + \log\left(\sum_{q=1}^{10} w_q \, \mathrm{e}^{\sqrt{2}\sigma_r \theta_q d_{j..}} G(\theta_q) \right) \right]$$

$$G(V) = \sum_{u=1}^{10} w_u \exp\left(\sqrt{2}\sigma_b \theta_u (\gamma d_{j..} + \sum_i d_{ij.} t_{ij}) - \frac{\mathrm{e}^{\left(\sqrt{2}\sigma_r V + \sqrt{2}\sigma_b \gamma \theta_u\right)}}{\sqrt{2}\sigma_b \theta_u} \sum_{i,k} \mathrm{e}^{\boldsymbol{x}_{ij}\boldsymbol{\beta}} f_{ijk}(\sqrt{2}\sigma_b \theta_u) \right)$$

$$f_{ijk}(x) = \begin{cases} 0 & \text{if } t_{ij} < a_{k-1} \\ \mathrm{e}^{\phi_k}(\mathrm{e}^{x t_{ijk}} - \mathrm{e}^{x a_{k-1}}) & \text{if } t_{ij} \in I_k \\ \mathrm{e}^{\phi_k}(\mathrm{e}^{x a_k} - \mathrm{e}^{x a_{k-1}}) & \text{if } t_{ij} \geq a_k \end{cases}$$

with: $d_{ij.} = \sum_k d_{ijk}, d_{j..} = \sum_i \sum_k d_{ijk}, \gamma = \frac{\sigma_{cb}}{\sigma_b^2}$ and $\sigma_r^2 = \sigma_c^2 - \gamma^2 \sigma_b^2$

Appendix: Vector of Parameters

Parameters Class

Characteristics of $\boldsymbol{p}$:

- n_p depends on the model, L and R
- Some parameters must be non-negative.

Therefore, for each model:

- Group similar parameters into the same "category" ($\{\beta_1, \beta_2, \ldots, \beta_R\} \rightarrow \boldsymbol{\beta}$)
- Provide $range = [range_{min}, range_{max}]$ to each category
- Provide number of category and vector with their cardinality.

Extraction of Parameters from p

For the *Adapted Paik eaM*:

$$\boldsymbol{p} = [\phi_1, \ldots, \phi_L, \beta_1, \ldots, \beta_R, \mu_1, \nu, \gamma_1, \ldots, \gamma_L] \rightarrow \boldsymbol{p} = [\boldsymbol{\phi}, \boldsymbol{\beta}, \mu_1, \nu, \boldsymbol{\gamma}],$$

$$n_p = 2L + R + 2,\ n_{category} = 5,\ v_{category} = [L, R, 1, 1, L]$$

How to extract the parameters:

```
T::TuplePaikType
AdaptedPaikeaM::extract_parameters(T::VectorXdr& v_parameters_) noexcept{
    T::VectorXdr phi = v_parameters_.head(Dataset::n_intervals);
    T::VectorXdr betar = v_parameters_.block(Dataset::n_intervals, 0,
                       Dataset::n_regressors, 1);
    T::VariableType mu1 = v_parameters_(Dataset::n_intervals + Dataset::n_regressors);
    T::VariableType mu2 = 1 - mu1;
    T::VariableType nu = v_parameters_(Dataset::n_intervals+Dataset::n_regressors+1);
    T::VectorXdr gammak = v_parameters_.tail(Dataset::n_intervals);
    return std::make_tuple(phi, betar, mu1, mu2, nu, gammak);
};
```

- Each model has its own *extract_parameters(. . .)* method
- Returned type is a model-related tuple
- Tuple contained extracted parameters and related variables (e.g. μ_2).

How Vector of Categories is used

Vector *all_n_parameters* used for checking parameters well-poseness condition:

```
void Parameters::check_condition(const T::VectorXdr& v_parameters_) const{
    T::NumberType n = 0;             //! Numerosity of a category
    T::IndexType actual_j = 0;       //! Index for the parameter vector
    T::VariableType a,b = 0.;        //! Min and max range
    //! Loop over the categories in all_n_parameters
    for(T::IndexType i = 0; i < n_ranges; ++i){
        n = all_n_parameters[i];
        a = range_min_parameters[i];
        b = range_max_parameters[i];
        //! Loop over the parameters in the category
        for(T::IndexType j = 0; j < n; ++j){
            if(std::isnan(v_parameters(actual_j))){
                throw MyException("At least one parameter is not provided ");
                    }
              else if((v_parameters(actual_j) < a) || (v_parameters(actual_j) > b)){
                throw MyException(...Value of parameter not in the range...);
                }
            actual_j += 1;
            }
    }
};
```

Appendix: OpenMP

Scheduling Strategy

omp_sched_t collects scheduling strategies:

```
typedef enum omp_sched_t {
  omp_sched_static = 0x1,
  omp_sched_dynamic = 0x2,
  omp_sched_guided = 0x3,
  omp_sched_auto = 0x4,
} omp_sched_t;
```

Select the chosen scheduling:
omp_sched_t(ParallelComponents::schedule_type),
being *schedule_type* the numeric scheduling id.

Appendix: Makefile and Bash Script

Example of Bash Script

Execute *"bash_app2010.sh"* in *BashScript* terminal.

```
#!/bin/bash

# Change directory to go into the main one, where Makefile is contained
cd ..
# Clean the sub-directories
make distclean
# Create doxygen documentation. It directly opens the index.html link
make docs
# Compile
make

# Change directory and go in Src, where the executable ./main is contained
cd Src
# Clear the terminal
clear

# Execute
./main ../Data/DataTool/DataToolFile2010.txt
       ../Data/DataIndividuals/DataIndividualsFile2010.txt

# Remove all doxygen documentation (Doc/doxygen folder)
make docsclean
```

Makefile

Makefile in *TimeVaryingSharedFrailtyCoxModels* folder:

- *make*: call make in indicated sub-folders and compile codes.
- *make docs*: generate doxygen documentation in sub-folders.
- *make distclean, make docsclean*: remove object files, executables, doxygen documentation.

Makefile in *Src* folder:

- Compile with c++=17, *-fopnemp* (OpenMP), *-O3 -DNDEBUG* (full Eigen speed).
- Add *-I$mkEigenInc* for Eigen library.

Really the End!