Mvlabund Package

|  |
| --- |
| // Interface between R and anova.cpp (Rcpp API >= 0.7.11) |
|  |

|  |
| --- |
| // |
|  |

|  |
| --- |
| // Author: Yi Wang (yi dot wang at unsw dot edu dot au) |
|  |

|  |
| --- |
| // Last modified: 20-April-2010 |
|  |

|  |
| --- |
| // |
|  |

|  |
| --- |
| // Rcpp/RcppGSL changes by Dirk Eddelbuettel, July - August 2015, Feb 2020 |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #include "mvabund\_types.h" |
|  |

|  |
| --- |
| #include "resampTest.h" |
|  |

|  |
| --- |
| extern "C" { |
|  |

|  |
| --- |
| #include "time.h" |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| using namespace Rcpp; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // declare a dependency on the headers in the RcppGSL package; |
|  |

|  |
| --- |
| // also activates plugin |
|  |

|  |
| --- |
| // [[Rcpp::depends(RcppGSL)]] |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // declare the function to be 'exported' to R |
|  |

|  |
| --- |
| // [[Rcpp::export]] |
|  |

|  |
| --- |
| List RtoAnovaCpp(const List &rparam, RcppGSL::Matrix &Y, RcppGSL::Matrix &X, |
|  |

|  |
| --- |
| RcppGSL::Matrix &isXvarIn, |
|  |

|  |
| --- |
| Rcpp::Nullable<RcppGSL::Matrix> &bID) { |
|  |

|  |
| --- |
| // pass parameters |
|  |

|  |
| --- |
| mv\_Method mm; |
|  |

|  |
| --- |
| // mm.tol = as<double>(rparam["tol"]); |
|  |

|  |
| --- |
| mm.nboot = as<unsigned int>(rparam["nboot"]); |
|  |

|  |
| --- |
| mm.corr = as<unsigned int>(rparam["cor\_type"]); |
|  |

|  |
| --- |
| mm.shrink\_param = as<double>(rparam["shrink\_param"]); |
|  |

|  |
| --- |
| mm.test = as<unsigned int>(rparam["test\_type"]); |
|  |

|  |
| --- |
| mm.resamp = as<unsigned int>(rparam["resamp"]); |
|  |

|  |
| --- |
| mm.reprand = as<unsigned int>(rparam["reprand"]); |
|  |

|  |
| --- |
| mm.student = as<unsigned int>(rparam["studentize"]); |
|  |

|  |
| --- |
| mm.punit = as<unsigned int>(rparam["punit"]); |
|  |

|  |
| --- |
| mm.rsquare = as<unsigned int>(rparam["rsquare"]); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| //unsigned int nRows = Y.nrow(); |
|  |

|  |
| --- |
| unsigned int nModels = isXvarIn.nrow(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // initialize anova class |
|  |

|  |
| --- |
| AnovaTest anova(&mm, Y, X, isXvarIn); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // Resampling indices |
|  |

|  |
| --- |
| if (bID.isNotNull()) { |
|  |

|  |
| --- |
| RcppGSL::Matrix m(bID); |
|  |

|  |
| --- |
| mm.nboot = m.nrow(); |
|  |

|  |
| --- |
| anova.bootID = m; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| // else Rprintf("bID is null"); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // resampling test |
|  |

|  |
| --- |
| anova.resampTest(); |
|  |

|  |
| --- |
| // anova.display(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // Wrap the gsl objects with Rcpp, then Rcpp -> R |
|  |

|  |
| --- |
| List rs = List::create( |
|  |

|  |
| --- |
| Named("multstat") = |
|  |

|  |
| --- |
| NumericVector(anova.multstat, anova.multstat + nModels - 1), |
|  |

|  |
| --- |
| Named("Pmultstat") = |
|  |

|  |
| --- |
| NumericVector(anova.Pmultstat, anova.Pmultstat + nModels - 1), |
|  |

|  |
| --- |
| Named("dfDiff") = NumericVector(anova.dfDiff, anova.dfDiff + nModels - 1), |
|  |

|  |
| --- |
| Named("statj") = RcppGSL::Matrix(anova.statj), |
|  |

|  |
| --- |
| Named("Pstatj") = RcppGSL::Matrix(anova.Pstatj), |
|  |

|  |
| --- |
| Named("nSamp") = anova.nSamp); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // clear objects |
|  |

|  |
| --- |
| anova.releaseTest(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return rs; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // declare the function to be 'exported' to R |
|  |

|  |
| --- |
| // [[Rcpp::export]] |
|  |

|  |
| --- |
| List RtoGlmAnova(const List &sparam, const List &rparam, RcppGSL::Matrix &Y, |
|  |

|  |
| --- |
| RcppGSL::Matrix &X, RcppGSL::Matrix &O, |
|  |

|  |
| --- |
| RcppGSL::Matrix &isXvarIn, |
|  |

|  |
| --- |
| Rcpp::Nullable<RcppGSL::Matrix> &bID, |
|  |

|  |
| --- |
| RcppGSL::Vector &lambda) { |
|  |

|  |
| --- |
| // pass regression parameters |
|  |

|  |
| --- |
| reg\_Method mm; |
|  |

|  |
| --- |
| mm.tol = as<double>(sparam["tol"]); |
|  |

|  |
| --- |
| mm.model = as<unsigned int>(sparam["regression"]); |
|  |

|  |
| --- |
| mm.speclink = as<unsigned int>(sparam["link"]); |
|  |

|  |
| --- |
| mm.estiMethod = as<unsigned int>(sparam["estimation"]); |
|  |

|  |
| --- |
| mm.varStab = as<unsigned int>(sparam["stablizer"]); |
|  |

|  |
| --- |
| mm.n = as<unsigned int>(sparam["n"]); |
|  |

|  |
| --- |
| mm.maxiter = as<unsigned int>(sparam["maxiter"]); |
|  |

|  |
| --- |
| mm.maxiter2 = as<unsigned int>(sparam["maxiter2"]); |
|  |

|  |
| --- |
| mm.warning = as<unsigned int>(sparam["warning"]); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // pass test parameters |
|  |

|  |
| --- |
| mv\_Method tm; |
|  |

|  |
| --- |
| tm.nboot = as<unsigned int>(rparam["nboot"]); |
|  |

|  |
| --- |
| tm.corr = as<unsigned int>(rparam["cor\_type"]); |
|  |

|  |
| --- |
| tm.test = as<unsigned int>(rparam["test\_type"]); |
|  |

|  |
| --- |
| tm.resamp = as<unsigned int>(rparam["resamp"]); |
|  |

|  |
| --- |
| tm.reprand = as<unsigned int>(rparam["reprand"]); |
|  |

|  |
| --- |
| tm.punit = as<unsigned int>(rparam["punit"]); |
|  |

|  |
| --- |
| tm.showtime = as<unsigned int>(rparam["showtime"]); |
|  |

|  |
| --- |
| tm.warning = as<unsigned int>(rparam["warning"]); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| unsigned int nRows = Y.nrow(); |
|  |

|  |
| --- |
| unsigned int nVars = Y.ncol(); |
|  |

|  |
| --- |
| unsigned int nParam = X.ncol(); |
|  |

|  |
| --- |
| unsigned int nModels = isXvarIn.nrow(); |
|  |

|  |
| --- |
| unsigned int nLambda = lambda.size(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| tm.anova\_lambda = gsl\_vector\_alloc(nLambda); |
|  |

|  |
| --- |
| gsl\_vector\_memcpy(tm.anova\_lambda, lambda); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| tm.nRows = nRows; |
|  |

|  |
| --- |
| tm.nVars = nVars; |
|  |

|  |
| --- |
| tm.nParam = nParam; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // do stuff |
|  |

|  |
| --- |
| clock\_t clk\_start, clk\_end; |
|  |

|  |
| --- |
| clk\_start = clock(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // glmfit |
|  |

|  |
| --- |
| PoissonGlm pfit(&mm); |
|  |

|  |
| --- |
| NBinGlm nbfit(&mm); |
|  |

|  |
| --- |
| BinGlm binfit(&mm); |
|  |

|  |
| --- |
| GammaGlm gfit(&mm); |
|  |

|  |
| --- |
| glm \*glmPtr[4] = {&pfit, &nbfit, &binfit, &gfit}; |
|  |

|  |
| --- |
| unsigned int mtype = mm.model - 1; |
|  |

|  |
| --- |
| glmPtr[mtype]->regression(Y, X, O, NULL); |
|  |

|  |
| --- |
| // glmPtr[mtype]->display(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| GlmTest myTest(&tm); |
|  |

|  |
| --- |
| // Resampling indices |
|  |

|  |
| --- |
| if (bID.isNotNull()) { |
|  |

|  |
| --- |
| RcppGSL::Matrix m(bID); |
|  |

|  |
| --- |
| tm.nboot = m.nrow(); |
|  |

|  |
| --- |
| myTest.bootID = m; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // resampling test |
|  |

|  |
| --- |
| myTest.anova(glmPtr[mtype], isXvarIn); |
|  |

|  |
| --- |
| // myTest.displayAnova(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // Timing |
|  |

|  |
| --- |
| clk\_end = clock(); |
|  |

|  |
| --- |
| unsigned long int dif = |
|  |

|  |
| --- |
| floor((double)(clk\_end - clk\_start) / (double)(CLOCKS\_PER\_SEC)); |
|  |

|  |
| --- |
| unsigned int hours = floor((double)(dif / (double)3600)); |
|  |

|  |
| --- |
| unsigned int min = floor((double)(dif % 3600) / (double)60); |
|  |

|  |
| --- |
| unsigned int sec = dif % 60; |
|  |

|  |
| --- |
| if (tm.showtime >= TRUE) |
|  |

|  |
| --- |
| Rprintf("Time elapsed: %d hr %d min %d sec\n", hours, min, sec); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // Wrap the gsl objects with Rcpp, then Rcpp -> R |
|  |

|  |
| --- |
| RcppGSL::VectorView mul = gsl\_matrix\_column(myTest.anovaStat, 0); |
|  |

|  |
| --- |
| RcppGSL::VectorView Pmul = gsl\_matrix\_column(myTest.Panova, 0); |
|  |

|  |
| --- |
| RcppGSL::MatrixView stat = |
|  |

|  |
| --- |
| gsl\_matrix\_submatrix(myTest.anovaStat, 0, 1, nModels - 1, nVars); |
|  |

|  |
| --- |
| RcppGSL::MatrixView Pstat = |
|  |

|  |
| --- |
| gsl\_matrix\_submatrix(myTest.Panova, 0, 1, nModels - 1, nVars); |
|  |

|  |
| --- |
| RcppGSL::MatrixView bootStore = |
|  |

|  |
| --- |
| gsl\_matrix\_submatrix(myTest.bootStore, 0, 0, tm.nboot, nVars + 1); |
|  |

|  |
| --- |
| List rs = List::create( |
|  |

|  |
| --- |
| Named("multstat") = mul, Named("Pmultstat") = Pmul, Named("statj") = stat, |
|  |

|  |
| --- |
| Named("Pstatj") = Pstat, |
|  |

|  |
| --- |
| Named("dfDiff") = |
|  |

|  |
| --- |
| NumericVector(myTest.dfDiff, myTest.dfDiff + nModels - 1), |
|  |

|  |
| --- |
| Named("nSamp") = myTest.nSamp, Named("bootStat") = bootStore); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // clear objects |
|  |

|  |
| --- |
| myTest.releaseTest(); |
|  |

|  |
| --- |
| glmPtr[mtype]->releaseGlm(); |
|  |

|  |
| --- |
| gsl\_vector\_free(tm.anova\_lambda); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return rs; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // declare the function to be 'exported' to R |
|  |

|  |
| --- |
| // [[Rcpp::export]] |
|  |

|  |
| --- |
| List RtoGlm(const List &rparam, RcppGSL::Matrix &Y, RcppGSL::Matrix &X, |
|  |

|  |
| --- |
| RcppGSL::Matrix &O) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // pass regression parameters |
|  |

|  |
| --- |
| reg\_Method mm; |
|  |

|  |
| --- |
| mm.tol = as<double>(rparam["tol"]); |
|  |

|  |
| --- |
| mm.model = as<unsigned int>(rparam["regression"]); |
|  |

|  |
| --- |
| mm.speclink = as<unsigned int>(rparam["link"]); |
|  |

|  |
| --- |
| mm.estiMethod = as<unsigned int>(rparam["estimation"]); |
|  |

|  |
| --- |
| mm.varStab = as<unsigned int>(rparam["stablizer"]); |
|  |

|  |
| --- |
| mm.n = as<unsigned int>(rparam["n"]); |
|  |

|  |
| --- |
| mm.maxiter = as<unsigned int>(rparam["maxiter"]); |
|  |

|  |
| --- |
| mm.maxiter2 = as<unsigned int>(rparam["maxiter2"]); |
|  |

|  |
| --- |
| mm.warning = as<unsigned int>(rparam["warning"]); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // do stuff |
|  |

|  |
| --- |
| PoissonGlm pfit(&mm); |
|  |

|  |
| --- |
| BinGlm lfit(&mm); |
|  |

|  |
| --- |
| NBinGlm nbfit(&mm); |
|  |

|  |
| --- |
| GammaGlm gfit(&mm); |
|  |

|  |
| --- |
| glm \*glmPtr[4] = {&pfit, &nbfit, &lfit, &gfit}; |
|  |

|  |
| --- |
| unsigned int mtype = mm.model - 1; |
|  |

|  |
| --- |
| glmPtr[mtype]->regression(Y, X, O, NULL); |
|  |

|  |
| --- |
| // glmPtr[mtype]->display(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // Wrap the glm object with Rcpp, then Rcpp -> R |
|  |

|  |
| --- |
| List rs = List::create( |
|  |

|  |
| --- |
| Named("coefficients") = RcppGSL::Matrix(glmPtr[mtype]->Beta), |
|  |

|  |
| --- |
| Named("var.coefficients") = RcppGSL::Matrix(glmPtr[mtype]->varBeta), |
|  |

|  |
| --- |
| Named("fitted.values") = RcppGSL::Matrix(glmPtr[mtype]->Mu), |
|  |

|  |
| --- |
| Named("linear.predictor") = RcppGSL::Matrix(glmPtr[mtype]->Eta), |
|  |

|  |
| --- |
| Named("residuals") = RcppGSL::Matrix(glmPtr[mtype]->Res), |
|  |

|  |
| --- |
| Named("PIT.residuals") = RcppGSL::Matrix(glmPtr[mtype]->PitRes), |
|  |

|  |
| --- |
| Named("sqrt.1\_Hii") = RcppGSL::Matrix(glmPtr[mtype]->sqrt1\_Hii), |
|  |

|  |
| --- |
| Named("var.estimator") = RcppGSL::Matrix(glmPtr[mtype]->Var), |
|  |

|  |
| --- |
| Named("sqrt.weight") = RcppGSL::Matrix(glmPtr[mtype]->wHalf), |
|  |

|  |
| --- |
| Named("theta") = |
|  |

|  |
| --- |
| NumericVector(glmPtr[mtype]->theta, glmPtr[mtype]->theta + Y.ncol()), |
|  |

|  |
| --- |
| Named("two.loglike") = |
|  |

|  |
| --- |
| NumericVector(glmPtr[mtype]->ll, glmPtr[mtype]->ll + Y.ncol()), |
|  |

|  |
| --- |
| Named("aic") = |
|  |

|  |
| --- |
| NumericVector(glmPtr[mtype]->aic, glmPtr[mtype]->aic + Y.ncol()), |
|  |

|  |
| --- |
| Named("deviance") = |
|  |

|  |
| --- |
| NumericVector(glmPtr[mtype]->dev, glmPtr[mtype]->dev + Y.ncol()), |
|  |

|  |
| --- |
| Named("iter") = NumericVector(glmPtr[mtype]->iterconv, |
|  |

|  |
| --- |
| glmPtr[mtype]->iterconv + Y.ncol())); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // clear objects |
|  |

|  |
| --- |
| glmPtr[mtype]->releaseGlm(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return rs; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // declare the function to be 'exported' to R |
|  |

|  |
| --- |
| // [[Rcpp::export]] |
|  |

|  |
| --- |
| List RtoGlmSmry(const List &sparam, // model params list |
|  |

|  |
| --- |
| const List &rparam, // test params list |
|  |

|  |
| --- |
| RcppGSL::Matrix &Y, // response matrix |
|  |

|  |
| --- |
| RcppGSL::Matrix &X, // Data matrix |
|  |

|  |
| --- |
| RcppGSL::Matrix &O, // offset |
|  |

|  |
| --- |
| Rcpp::Nullable<RcppGSL::Matrix> &bID, // boot id |
|  |

|  |
| --- |
| RcppGSL::Vector &lambda) { // shrinkage parameters |
|  |

|  |
| --- |
| // Pass regression parameters |
|  |

|  |
| --- |
| reg\_Method mm; |
|  |

|  |
| --- |
| mm.tol = as<double>(sparam["tol"]); |
|  |

|  |
| --- |
| mm.model = as<unsigned int>(sparam["regression"]); |
|  |

|  |
| --- |
| mm.speclink = as<unsigned int>(sparam["link"]); |
|  |

|  |
| --- |
| mm.estiMethod = as<unsigned int>(sparam["estimation"]); |
|  |

|  |
| --- |
| mm.varStab = as<unsigned int>(sparam["stablizer"]); |
|  |

|  |
| --- |
| mm.n = as<unsigned int>(sparam["n"]); |
|  |

|  |
| --- |
| mm.maxiter = as<unsigned int>(sparam["maxiter"]); |
|  |

|  |
| --- |
| mm.maxiter2 = as<unsigned int>(sparam["maxiter2"]); |
|  |

|  |
| --- |
| mm.warning = as<unsigned int>(sparam["warning"]); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // pass test parameters |
|  |

|  |
| --- |
| mv\_Method tm; |
|  |

|  |
| --- |
| tm.corr = as<unsigned int>(rparam["cor\_type"]); |
|  |

|  |
| --- |
| tm.test = as<unsigned int>(rparam["test\_type"]); |
|  |

|  |
| --- |
| tm.resamp = as<unsigned int>(rparam["resamp"]); |
|  |

|  |
| --- |
| tm.reprand = as<unsigned int>(rparam["reprand"]); |
|  |

|  |
| --- |
| tm.punit = as<unsigned int>(rparam["punit"]); |
|  |

|  |
| --- |
| tm.nboot = as<unsigned int>(rparam["nboot"]); |
|  |

|  |
| --- |
| tm.showtime = as<unsigned int>(rparam["showtime"]); |
|  |

|  |
| --- |
| tm.warning = as<unsigned int>(rparam["warning"]); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| unsigned int nRows = Y.nrow(); |
|  |

|  |
| --- |
| unsigned int nVars = Y.ncol(); |
|  |

|  |
| --- |
| unsigned int nParam = X.ncol(); |
|  |

|  |
| --- |
| unsigned int nLambda = lambda.size(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| tm.smry\_lambda = gsl\_vector\_alloc(nLambda); |
|  |

|  |
| --- |
| gsl\_vector\_memcpy(tm.smry\_lambda, lambda); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| tm.nRows = nRows; |
|  |

|  |
| --- |
| tm.nVars = nVars; |
|  |

|  |
| --- |
| tm.nParam = nParam; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // do stuff |
|  |

|  |
| --- |
| clock\_t clk\_start, clk\_end; |
|  |

|  |
| --- |
| clk\_start = clock(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // Glm fit |
|  |

|  |
| --- |
| PoissonGlm pfit(&mm); |
|  |

|  |
| --- |
| BinGlm lfit(&mm); |
|  |

|  |
| --- |
| NBinGlm nbfit(&mm); |
|  |

|  |
| --- |
| GammaGlm gfit(&mm); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| glm \*glmPtr[4] = {&pfit, &nbfit, &lfit, &gfit}; |
|  |

|  |
| --- |
| unsigned int mtype = mm.model - 1; |
|  |

|  |
| --- |
| // do the regression |
|  |

|  |
| --- |
| glmPtr[mtype]->regression(Y, X, O, NULL); |
|  |

|  |
| --- |
| if (mm.warning) { |
|  |

|  |
| --- |
| // glmPtr[mtype]->display(); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| GlmTest myTest(&tm); |
|  |

|  |
| --- |
| // Resampling indices |
|  |

|  |
| --- |
| if (bID.isNotNull()) { |
|  |

|  |
| --- |
| RcppGSL::Matrix m(bID); |
|  |

|  |
| --- |
| tm.nboot = m.nrow(); |
|  |

|  |
| --- |
| myTest.bootID = m; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| // resampling test |
|  |

|  |
| --- |
| myTest.summary(glmPtr[mtype]); |
|  |

|  |
| --- |
| // myTest.displaySmry(glmPtr[mtype]); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // Timing |
|  |

|  |
| --- |
| clk\_end = clock(); |
|  |

|  |
| --- |
| unsigned long int dif = |
|  |

|  |
| --- |
| floor((double)(clk\_end - clk\_start) / (double)(CLOCKS\_PER\_SEC)); |
|  |

|  |
| --- |
| unsigned int hours = floor((double)dif / (double)3600); |
|  |

|  |
| --- |
| unsigned int min = floor((double)(dif - hours \* 3600) / (double)60); |
|  |

|  |
| --- |
| unsigned int sec = dif - hours \* 3600 - min \* 60; |
|  |

|  |
| --- |
| if (tm.showtime >= TRUE) |
|  |

|  |
| --- |
| Rprintf("Time elapsed: %d hr %d min %d sec\n", hours, min, sec); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // Wrap gsl vectors with Rcpp, then Rcpp -> R |
|  |

|  |
| --- |
| gsl\_vector\_view mult = gsl\_matrix\_row(myTest.smryStat, 0); |
|  |

|  |
| --- |
| gsl\_vector\_view Pmul = gsl\_matrix\_row(myTest.Psmry, 0); |
|  |

|  |
| --- |
| RcppGSL::VectorView umult = gsl\_vector\_subvector(&mult.vector, 1, nVars); |
|  |

|  |
| --- |
| RcppGSL::VectorView uPmul = gsl\_vector\_subvector(&Pmul.vector, 1, nVars); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| gsl\_vector\_view stat = gsl\_matrix\_column(myTest.smryStat, 0); |
|  |

|  |
| --- |
| gsl\_vector\_view Pstat = gsl\_matrix\_column(myTest.Psmry, 0); |
|  |

|  |
| --- |
| RcppGSL::VectorView signi = gsl\_vector\_subvector(&stat.vector, 1, nParam); |
|  |

|  |
| --- |
| RcppGSL::VectorView Psign = gsl\_vector\_subvector(&Pstat.vector, 1, nParam); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| RcppGSL::MatrixView usig = |
|  |

|  |
| --- |
| gsl\_matrix\_submatrix(myTest.smryStat, 1, 1, nParam, nVars); |
|  |

|  |
| --- |
| RcppGSL::MatrixView uPsig = |
|  |

|  |
| --- |
| gsl\_matrix\_submatrix(myTest.Psmry, 1, 1, nParam, nVars); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| List rs = List::create(Named("multstat") = gsl\_vector\_get(&mult.vector, 0), |
|  |

|  |
| --- |
| Named("Pmultstat") = gsl\_vector\_get(&Pmul.vector, 0), |
|  |

|  |
| --- |
| Named("unitmult") = umult, Named("Punitmult") = uPmul, |
|  |

|  |
| --- |
| Named("signific") = signi, Named("Psignific") = Psign, |
|  |

|  |
| --- |
| Named("unitsign") = usig, Named("Punitsign") = uPsig, |
|  |

|  |
| --- |
| Named("nSamp") = myTest.nSamp, |
|  |

|  |
| --- |
| Named("aic") = |
|  |

|  |
| --- |
| NumericVector(myTest.aic, myTest.aic + nVars)); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // clear objects |
|  |

|  |
| --- |
| glmPtr[mtype]->releaseGlm(); |
|  |

|  |
| --- |
| myTest.releaseTest(); |
|  |

|  |
| --- |
| gsl\_vector\_free(tm.smry\_lambda); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return rs; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // [[Rcpp::export]] |
|  |

|  |
| --- |
| List RtoSmryCpp(const List &rparam, RcppGSL::Matrix &Y, RcppGSL::Matrix &X, |
|  |

|  |
| --- |
| Rcpp::Nullable<RcppGSL::Matrix> &bID) { |
|  |

|  |
| --- |
| // pass regression parameters |
|  |

|  |
| --- |
| mv\_Method mm; |
|  |

|  |
| --- |
| mm.nboot = as<unsigned int>(rparam["nboot"]); |
|  |

|  |
| --- |
| mm.corr = as<unsigned int>(rparam["cor\_type"]); |
|  |

|  |
| --- |
| mm.shrink\_param = as<double>(rparam["shrink\_param"]); |
|  |

|  |
| --- |
| mm.test = as<unsigned int>(rparam["test\_type"]); |
|  |

|  |
| --- |
| mm.resamp = as<unsigned int>(rparam["resamp"]); |
|  |

|  |
| --- |
| mm.reprand = as<unsigned int>(rparam["reprand"]); |
|  |

|  |
| --- |
| mm.student = as<unsigned int>(rparam["studentize"]); |
|  |

|  |
| --- |
| mm.punit = as<unsigned int>(rparam["punit"]); |
|  |

|  |
| --- |
| mm.rsquare = as<unsigned int>(rparam["rsquare"]); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| //unsigned int nRows = Y.nrow(); |
|  |

|  |
| --- |
| unsigned int nVars = Y.ncol(); |
|  |

|  |
| --- |
| unsigned int nParam = X.ncol(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // do stuff |
|  |

|  |
| --- |
| // clock\_t clk\_start, clk\_end; |
|  |

|  |
| --- |
| // clk\_start = clock(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // initialize summary class |
|  |

|  |
| --- |
| Summary smry(&mm, Y, X); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // Resampling indices |
|  |

|  |
| --- |
| if (bID.isNotNull()) { |
|  |

|  |
| --- |
| RcppGSL::Matrix m(bID); |
|  |

|  |
| --- |
| mm.nboot = m.nrow(); |
|  |

|  |
| --- |
| smry.bootID = m; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // resampling test |
|  |

|  |
| --- |
| smry.resampTest(); |
|  |

|  |
| --- |
| // smry.display(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // Wrap gsl vectors with Rcpp, then Rcpp -> R |
|  |

|  |
| --- |
| RcppGSL::VectorView umulti = gsl\_matrix\_row(smry.unitstat, 0); |
|  |

|  |
| --- |
| RcppGSL::VectorView uPmult = gsl\_matrix\_row(smry.Punitstat, 0); |
|  |

|  |
| --- |
| RcppGSL::MatrixView usigni = |
|  |

|  |
| --- |
| gsl\_matrix\_submatrix(smry.unitstat, 1, 0, nParam, nVars); |
|  |

|  |
| --- |
| RcppGSL::MatrixView uPsign = |
|  |

|  |
| --- |
| gsl\_matrix\_submatrix(smry.Punitstat, 1, 0, nParam, nVars); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| List rs = List::create( |
|  |

|  |
| --- |
| Named("multstat") = smry.multstat[0], |
|  |

|  |
| --- |
| Named("Pmultstat") = smry.Pmultstat[0], |
|  |

|  |
| --- |
| Named("signific") = |
|  |

|  |
| --- |
| NumericVector(smry.multstat + 1, smry.multstat + nParam + 1), |
|  |

|  |
| --- |
| Named("Psignific") = |
|  |

|  |
| --- |
| NumericVector(smry.Pmultstat + 1, smry.Pmultstat + nParam + 1), |
|  |

|  |
| --- |
| Named("unitmult") = umulti, Named("Punitmult") = uPmult, |
|  |

|  |
| --- |
| Named("unitsign") = usigni, Named("Punitsign") = uPsign, |
|  |

|  |
| --- |
| Named("nSamp") = smry.nSamp, Named("R2") = smry.R2); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // clear objects |
|  |

|  |
| --- |
| smry.releaseSummary(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return rs; |
|  |

}

# The Actual Code Of the Nullable-Optional-Arguments-in-Rcpp-functions

### Nullability of Vector, Matrix or Logical Vector

// Checking setting Vector, Matrix and LogicalVector to NULL by default and

// using the input if not set to NULL

#include

using namespace Rcpp;

// [[Rcpp::export]]

void nullable1(Nullable<NumericVector> NV\_ = R\_NilValue,

Nullable<NumericMatrix> NM\_ = R\_NilValue,

Nullable<LogicalVector> LG\_ = R\_NilValue){

if (NV\_.isNotNull()) {

NumericVector NV(NV\_); // casting to underlying type NumericVector

Rcout << "Numeric Vector is set to not NULL." << std::endl;

Rcout << NV << std::endl;

} else if (NM\_.isNotNull()){

NumericMatrix NM(NM\_); // casting to underlying type NumericMatrix

Rcout << "Numeric Matrix is set to not NULL." << std::endl;

Rcout << NM << std::endl;

} else if (LG\_.isNotNull()){

LogicalVector LG(LG\_); // casting to underlying type Boolean

Rcout << "Logical Vector is set to not NULL." << std::endl;

Rcout << LG << std::endl;

} else {

warning("All arguments are set to NULL.\n");

}

}

Running a few examples with setting NULL for a vector, matrix or a boolean value gives us the  
expected results.

nullable1(c(1,2), NULL, NULL)

Numeric Vector is set to not NULL.

1 2

m <- matrix(-0.5, 3, 3)

nullable1(NULL, m, NULL)

Numeric Matrix is set to not NULL.

-0.500000 -0.500000 -0.500000

-0.500000 -0.500000 -0.500000

-0.500000 -0.500000 -0.500000

nullable1(NULL, NULL, FALSE)

Logical Vector is set to not NULL.

0

nullable1(NULL, NULL, NULL)

Warning in nullable1(NULL, NULL, NULL): All arguments are set to NULL.

nullable1(c(), NULL, NULL)

Warning in nullable1(c(), NULL, NULL): All arguments are set to NULL.

We get the same result when the input to the NumericVector argument is not NULL but an empty  
vector, i.e., c(), which is also expected since is.null(c()) is TRUE in R.

A stricter test whether the input is usable can be (aptly named) isUsable().

// Testing another check, isUsable for a Nullable Vector

#include

using namespace Rcpp;

// [[Rcpp::export]]

void nullable2(Nullable<NumericVector> NV\_ = R\_NilValue) {

if (NV\_.isUsable()) {

NumericVector NV(NV\_); // casting to underlying type NumericVector

Rcout << "Input is usable." << std::endl;

Rcout << NV << std::endl;

} else {

Rcout << "Input is either NULL or not usable." << std::endl;

}

}

### Nullability of DataFrame and List

Rcpp::Nullable<> works for SEXP based Rcpp types, so Rcpp::DataFrame and Rcpp::List can  
also be set to Nullable and instantiated if not NULL.

// Checking setting List and DataFrame to NULL by default and

// using the input if not set to NULL

#include

using namespace Rcpp;

// [[Rcpp::export]]

void nullable3(Nullable<List> ls\_ = R\_NilValue, Nullable<DataFrame> df\_ = R\_NilValue){

if (ls\_.isNotNull()){

Rcpp::List ls(ls\_); // casting to underlying type List

Rcout << "List is not NULL." << std::endl;

Rcout << "List length of " << ls.length() << " elements." << std::endl;

} else if(df\_.isNotNull()) {

Rcpp::DataFrame df(df\_); // casting to underlying type DataFrame

Rcout << "DataFrame is not NULL." << std::endl;

Rcout << "DataFrame of " << df.nrows() << " rows and " << df.length() << " columns." << std::endl;

} else {

warning("Both inputs are NULL.\n");

}

}

Testing with Rcpp::List and Rcpp::DataFrame gives expected results, i.e.,

mylist <- list(A = 1:10, B = letters[1:10])

nullable3(mylist, NULL)

List is not NULL.

List length of 2 elements.

df <- data.frame(A = 1:20, B = letters[1:20])

nullable3(NULL, df)

DataFrame is not NULL.

DataFrame of 20 rows and 2 columns.

### Nullability of RcppGSL::Matrix

In addition to Rcpp types, RcppGSL::Matrix can also be set with Nullable e.g.,

// Checking setting RcppGSL Matrix to NULL by default and

// using the input if not set to NULL

// [[Rcpp::depends(RcppGSL)]]

#include

using namespace Rcpp;

// [[Rcpp::export]]

void nullable4(Rcpp::Nullable<RcppGSL::Matrix> M\_ = R\_NilValue) {

if (M\_.isNotNull()){

RcppGSL::Matrix M(M\_); // casting to underlying type RcppGSL::Matrix

Rcout << "Input is not NULL." << std::endl;

Rcout << "Input GSL matrix has " << M.nrow() << " and " << M.ncol() << " columns.\n";

} else {

warning("Input GSL Matrix is NULL.\n");

}

}

Finally, testing with RcppGSL::Matrix which can also be set to Nullable<>, i.e.,

nullable4(NULL) # testing with NULL

Warning in nullable4(NULL): Input GSL Matrix is NULL.

m <- matrix(-0.5, 3, 3) # testing with a non-NULL matrix

nullable4(m)

Input is not NULL.

Input GSL matrix has 3 and 3 columns.

### Summary

Rcpp provides a convenient construct to set datatypes to NULL using R\_NilValue and application  
of the datatype if not set to NULL using the .isNotNull() check. This construct to applied to  
set datatypes to NULL as default values and possible simple simplification.