rtkpp

Un package pour faire l'interface entre R et la bibliothèque STK++

Serge Iovleff
Equipe Projet Modal
Lille Nord-Europe

Sommaire

Sommaire

Core functionalities

The term "core" means anything not related directly to statistics but used everywhere. The main functionalities provided are :

1. A set of exception classes with associated macro for handling errors
2. Predefined types (Binary, Sign, Range), "fundamental" types (Char, String, Integer, Real) and for all these types
   - 2.1 Runtime type identification mechanism (not very used except in the DManager project)
   - 2.2 Missing Data handling
3. Two kinds of Containers with optimized visitors (using template metaprogramming techniques),
4. Template Expression implementation,
5. Optimized (parallelized) matrix product.

Handling Missing values

Missing values are common in statistics. A framework in statistics have to take care of them.

1. STK++ add to the C/C++ built-in types (double/float and int) a special missing value,
2. STK++ classes have predefined missing values if necessary.

Missing values in arrays are handled in a transparent way

1. Count missing values:

```
CArray2X a(2,4);
a << 0, 1, 2, 3
   , 4, 5, 6, 7;
a(1,1) = a(1,3) = Arithmetic<Real>::NA();
std::cout << "count(a.isNA())= " << count(a.isNA());
std::cout << "countByRow(a.isNA())= " << countByRow(a.isNA());
```

```
count(a.isNA())= 0 1 0 1
countByRow(a.isNA())= 0 2
```

2. Compute usual statistics:

```
std::cout << "meanSafe(a)= " << meanSafe(a);
std::cout << "meanSafeByRow(a)= " << meanSafeByRow(a);
```

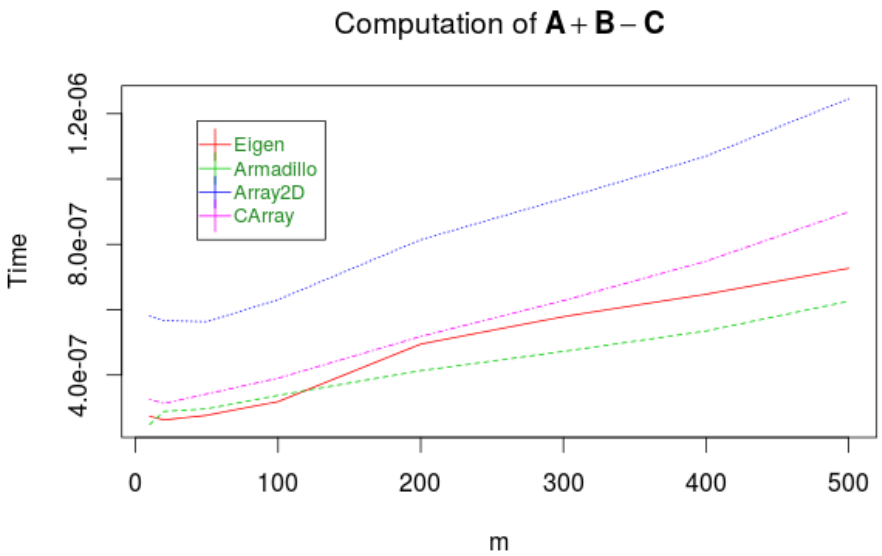
```
meanSafe(a)= 2 1 4 3
meanSafeByRow(a)= 1.5 5
```

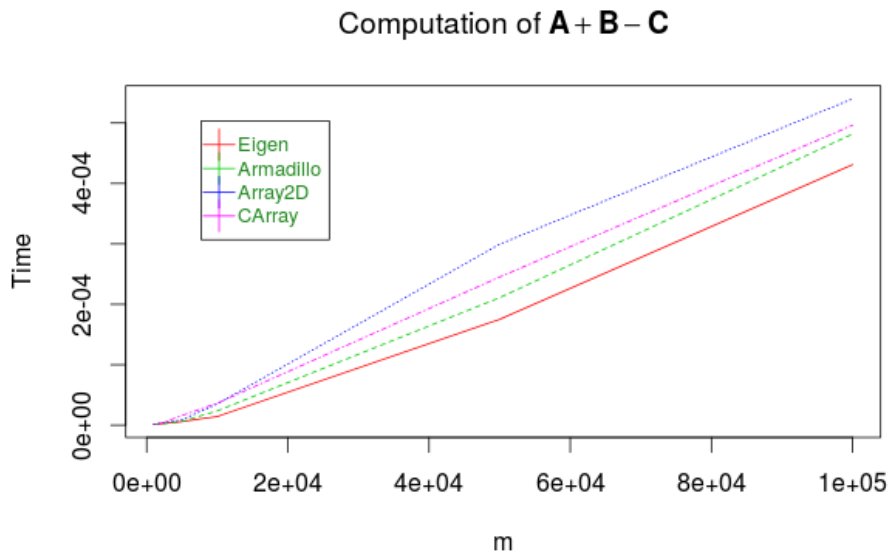
3. Read and write (csv) files with missing values...

Expression Templates: Benchmark 1

```
Eigen::VectorXd Reig = AEig + BEig - CEig;
arma::vec RA = AA + BA - CA;
VectorX R2D = A2D + B2D - C2D;
CVectorX RC = AC + BC - CC;
```

Computational time of the expression $A + B - C$ with A, B, C **vectors**.

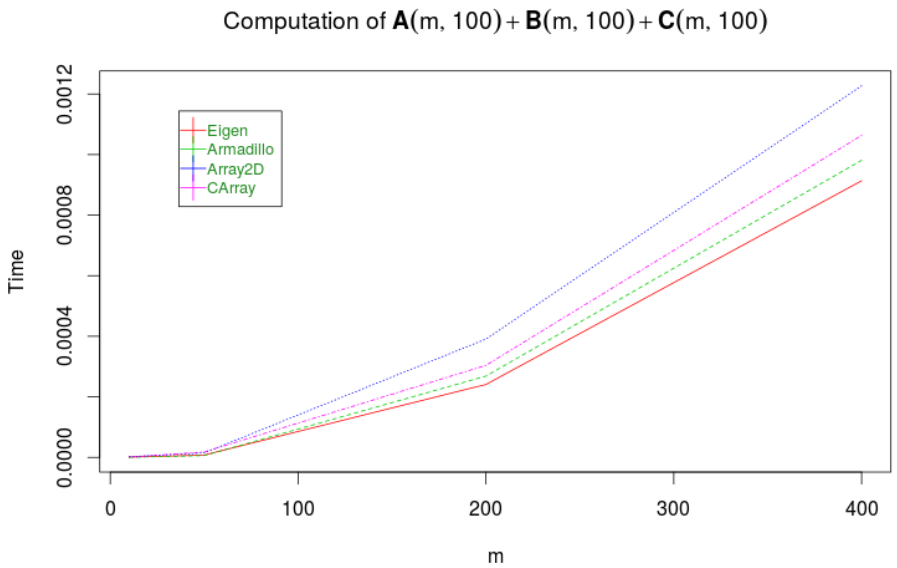


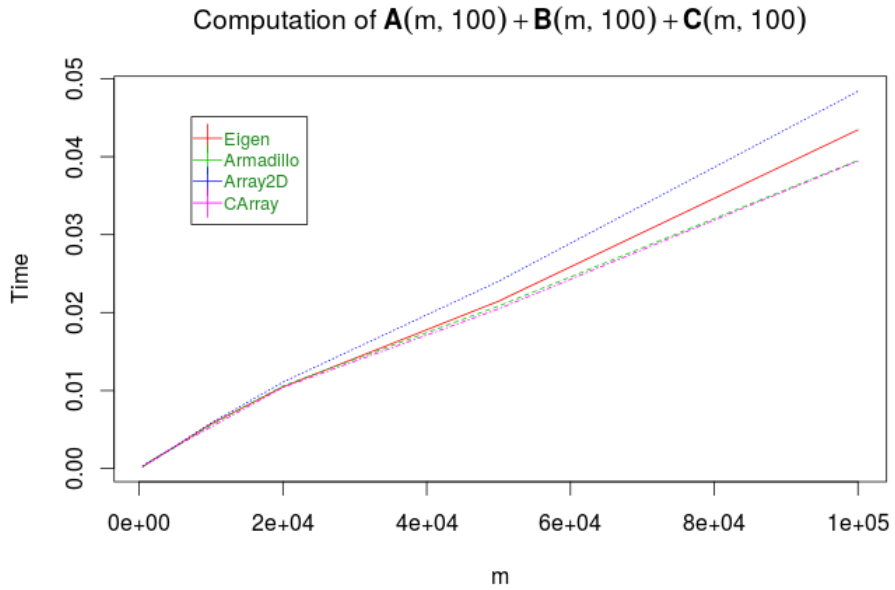


Expression Templates: Benchmark 2

```
Eigen::ArrayXXd Reig = Reig = AEig + BEig - CEig;
arma::mat RA = AA + BA - CA;
ArrayXX R2D = A2D + B2D - C2D);
CArrayXX RC = AC + BC - CC;
```

Computational time of the expression $A + B + C$ with A, B, C **matrices**.

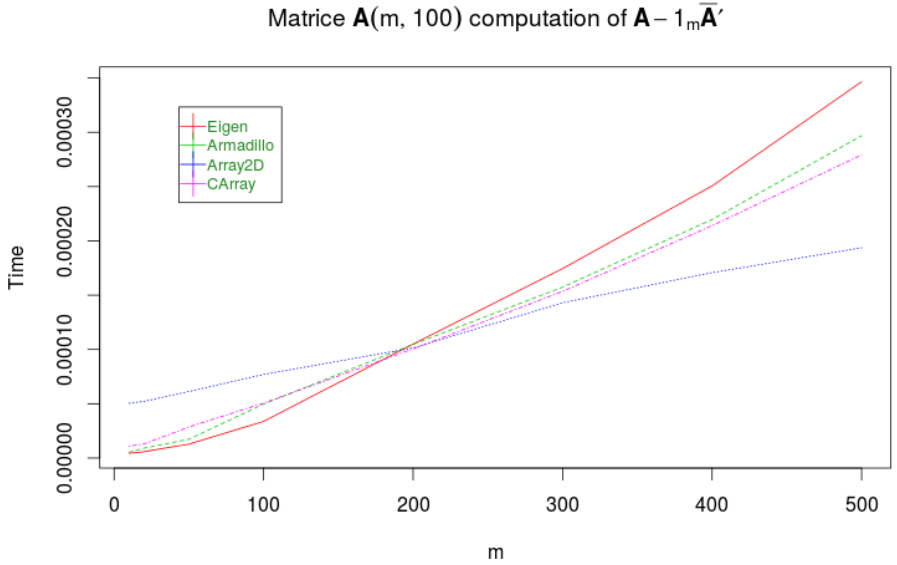


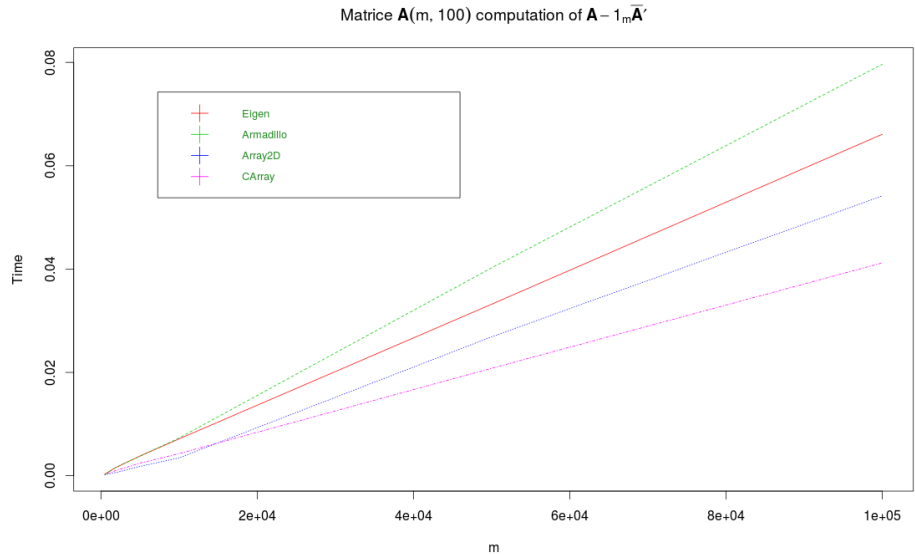


Expression Templates: Benchmark 3

```
Eigen::ArrayXXd Reig = AE.matrix() - Eigen::ArrayXd::Ones(m).matrix() * AE.colwise().mean().
      matrix();
arma::mat RA = AA - arma::ones(m) * arma::mean(AA);
ArrayXX R2D = AD - Const::VectorX(m) * Stat::mean(AD);
CArrayXX RC = AC - Const::VectorX(m) * Stat::mean(AC);
```

Centering an Array: computational time of the expression $A - \mathbf{1}_n \bar{A}'$ with A matrix of size $(m, 10)$

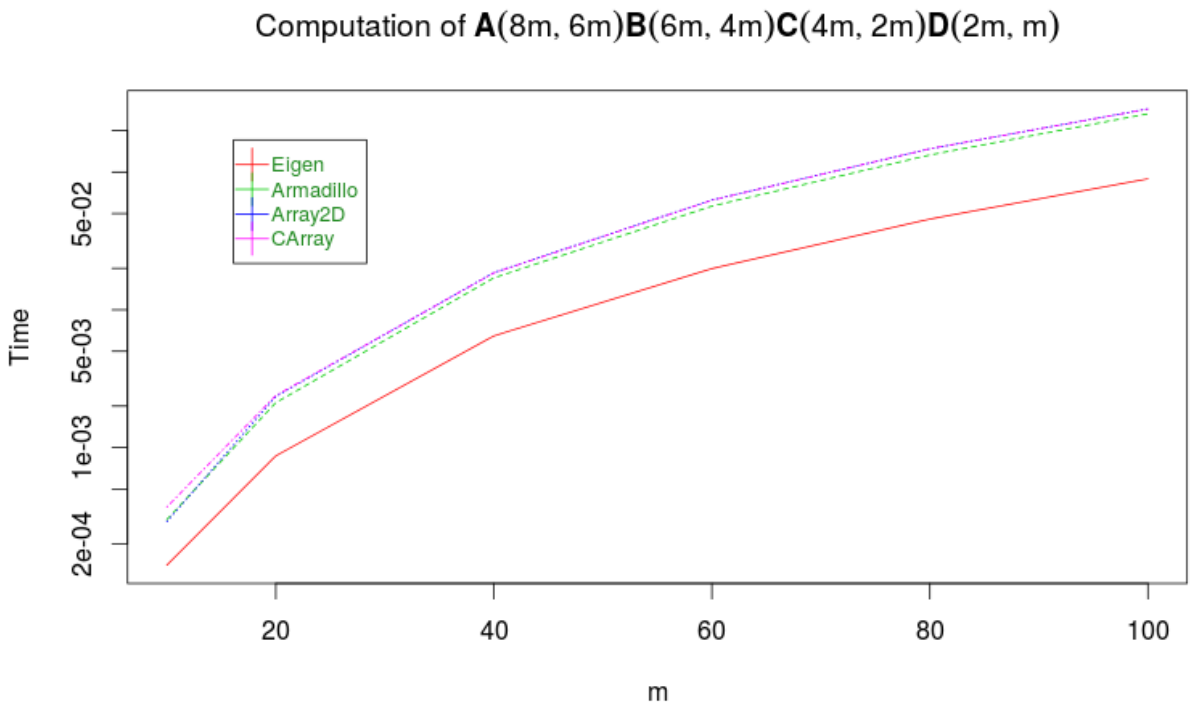




Matrix Multiplication: Benchmark

```
Eigen::ArrayXXd Reig = AE.matrix() * (BE.matrix() * (CE.matrix() * DE.matrix()));
arma::mat RA = AA * (BA * (CA * DA));
ArrayXX R2D = AD * (BD * (CD * DD));
CArrayXX RC = AC * (BC * (CC * DC));
```

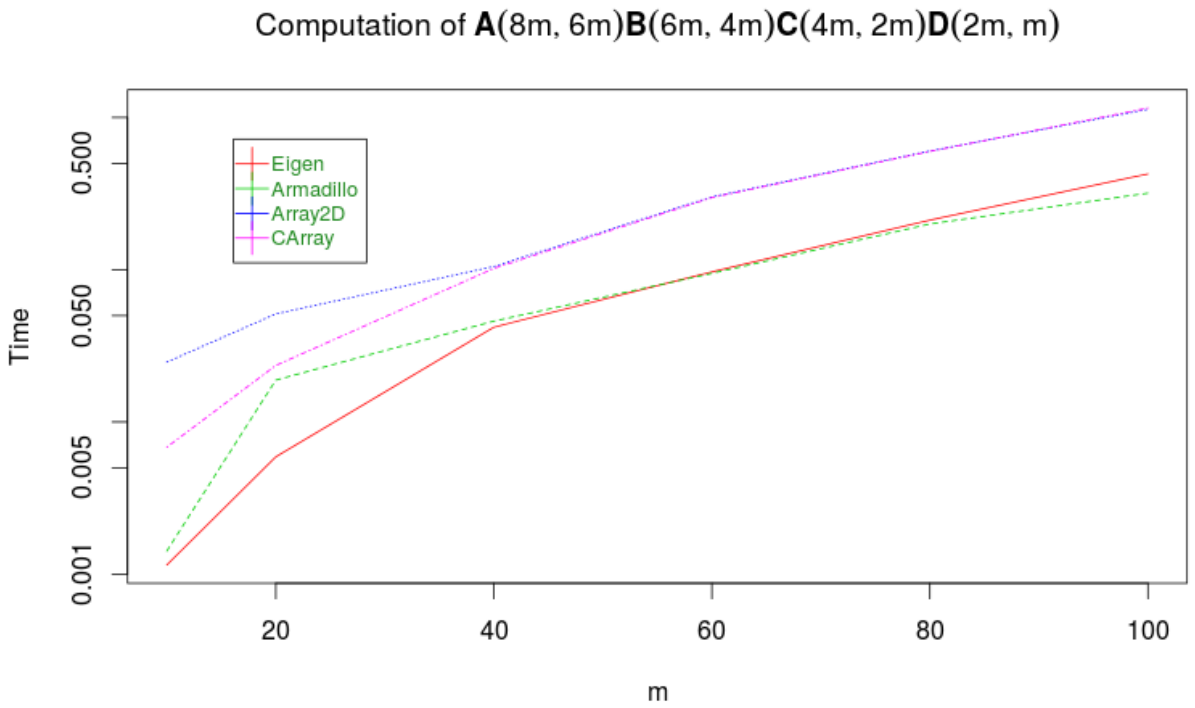
Computational time of the expression $A * B * C * D$ without parallelization



Matrix Multiplication: Benchmark

```
Eigen::ArrayXXd Reig = AE.matrix() * (BE.matrix() * (CE.matrix() * DE.matrix()));
arma::mat RA = AA * (BA * (CA * DA));
ArrayXX R2D = AD * (BD * (CD * DD));
CArrayXX RC = AC * (BC * (CC * DC));
```

Computational time of the expression $A * B * C * D$ with parallelization enabled



Handling data with Array2D

The Array2D templated class and its variants (vector, point, diagonal, etc.) were the original arrays used in STK++. Using them, it was possible to:

- Add, Remove, Insert row and columns
- concatenate the columns (without copy) of two arrays

For example, the code

```
#include "STKpp.h"
using namespace STK;
int main(int argc, char *argv[])
{
  ArrayXX A(4, Range(0,2));
  Law::Normal law(1.,2.);
  A.rand(law);
  stk_cout << _T("A =\n") << A;
  // Adding a column with 1
  A.pushFrontCols(Const::VectorX(4)
        );
  // Insert x^2 and x^3
  A.insertCols(2, 2);
  A.col(2) = A.col(1).square();
  A.col(3) = A.col(1).cube();
  A.pushBackCols(2);
  A.col(5) = A.col(4).square();
  A.col(6) = A.col(4).cube();
  stk_cout << _T("A =\n") << A;

  VectorX B(4); B << 1, 3, 5, 6;
  stk_cout << _T("B =\n") << B;
  B.insertElt(2); B[2] = 3;
  stk_cout << _T("B =\n") << B;
}
```

produces the output

```
A =
1.59264   -0.22753
3.47208 -0.0682241
2.32915    2.99338
 3.3007    2.73466
A =
1 1.59264  2.5365 4.03973   -0.22753  0.0517698   -0.0117792
1 3.47208 12.0553  41.857 -0.0682241 0.00465453 -0.000317551
1 2.32915 5.42495 12.6355    2.99338    8.96032      26.8216
1  3.3007 10.8946 35.9598    2.73466    7.47836      20.4508
B =
1 3 5 6
B =
1 3 3 5 6
```

Utilities

There is four projects in STK++ providing utilities needed by scientists and statisticians:

1. The **DManager** project provides tools for input and output handling,
2. The **Analysis** project allows to get usual mathematical constants, to compute usual analytical functions, zeros of functions, ...
3. The **Algebra** project furnish the usual matrix decomposition methods and an interface with LAPACK,
4. The **STatistiK** project implements,
   - 4.1 some of the usual probability laws (computation of the pdf, log-pdf, cdf, quantiles, and random number generators),
   - 4.2 a set of `functors` computing usual statistics and acting on arrays and expressions
   - 4.3 a set of class allowing to compute the usual descriptive statistics on vectors (oldest code from ADNS !)

Statistics

There is five projects directly dedicated to specific statistical problems

1. the **Regress** project dedicated to (least square) regression problem. It proposes
   1.1 Multidimensional regression
   1.2 BSpline regression and Additive BSpline regression
2. the **Reduct** project dedicated to dimension reduction technique. It proposes PCA and Contiguity analysis
3. The **AAModels** project I develop for the Auto-Associative models
4. The **Clustering** project implements mixture model melange for Categorical, Gaussian (with diagonal covariance) and Gamma distributions with various constraints on the parameters and flexible estimation strategies
5. The **StatModels** project allows to estimate the parameters of usual parameterized statistical models

Sommaire

rtkpp: Using STK++ inside R

R and the Cran is becoming the main way to diffuse new methods. In my research team (MODAL) almost all publications is accompanied with a R package. But

- `R`, as a language, can be not well suited for complex methods and algorithms. In this case, one used rather `C` or `C++` implementations and bind them using the `.Call` R method allowing dynamic link with external libraries.
- The package `Rcpp` is the easiest way to glue R data storage (in `C`) with `C++` storages: it provides wrappers allowing to manipulate the `R` data at a high level without taking care of the underlying `C` storage.
- the challenge was thus to facilitate the data movement between `R` and `STK++` in a transparent way.

rtkpp: extending Rcpp

Rcpp facilitates data interchange between R and C++ through the templated functions `Rcpp::as` (for conversion of objects from R to C++) and `Rcpp::wrap` (for conversion from C++ to R). In other words, we convert between the so-called S-expression pointers (in type SEXP) to a templated C++ type, and vice versa. The corresponding function declarations are as follows:

```
//bconversion from R to C ++
template<typename T> T as (SEXP x) ;
// conversion from C++ to R
template<typename T>
SEXP wrap(const T& object) ;))
```

[Source http://dirk.eddelbuettel.com/code/rcpp/Rcpp-extending.pdf]

rtkpp: extending Rcpp

rtkpp extends these two functions so that they can be used directly by the STK++ user. But SEXP structures, Rcpp matrices and vectors can also be directly wrapped by rtkpp.
Here is an example taken from the ClusterLauncher class in package MixAll

```
NumericMatrix m_data = s4_component.slot("data"); // get a Rcpp matrrix
RcppMatrix<double> data(m_data); // wrap it in a STK++ Array/Expression
```

and an example of the wrap usage

```
s4_model_.slot("tik") = Rcpp::wrap(p_composer_->tik()); // copy the CArrayXX in a SEXP structure
```

rtkpp: extending Rcpp

In some cases one want to wrap complex expression rather than arrays. `rtkpp` extends the `Rcpp` `wrap` method by implementing its own `wrap`.

This example is taken from the `countMissing` package.

```
RcppExport SEXP countNA( SEXP r_matrix)
{
  BEGIN_RCPP
  // wrap r matrix in a STK matrix
  STK::RMatrix<double> m_data(r_matrix);
  // use STK::wrap for a direct evaluation in cols and rows of count and countByRow
  return Rcpp::List::create( Rcpp::Named("rows") = STK::wrap(STK::countByRow(m_data.isNA()))
                           , Rcpp::Named("cols") = STK::wrap(STK::count(m_data.isNA()))
                           );
  END_RCPP
}
```

Sommaire

Using STK++/rtkpp

In order to use `rtkpp` one must

1. add to the DESCRIPTION file

```
LinkingTo: Rcpp, rtkpp
```

2. defines the variables `PKG_CXXFLAGS`, `PKG_CPPFLAGS` and `PKG_LIBS` in the makevars file

```
# makevars sample
#
...
PKG_CXXFLAGS = `${R_HOME}/bin/Rscript -e "rtkpp:::CxxFlags()"`
PKG_CPPFLAGS = `${R_HOME}/bin/Rscript -e "rtkpp:::CppFlags()"`

## use $(LAPACK_LIBS) $(BLAS_LIBS) $(FLIBS) if you want lapack
PKG_LIBS = `$(R_HOME)/bin/Rscript -e "rtkpp:::LdFlags()"` \
           $(SHLIB_OPENMP_CFLAGS) $(LAPACK_LIBS) $(BLAS_LIBS) $(FLIBS) $(PKGLIB)

all: $(SHLIB)
$(SHLIB): pkglib

pklib:...
```

The HDPenReg package

The `HDPenReg` package implements the Lars algorithm taking advantage of the `Array2D` capabilities to add/remove columns quickly. it uses also the updating formula of the Qr class implemented in the Algebra project.

"*The LARS (Least Angle Regression) algorithm is a stepwise procedure for solving the lasso problem. The principle of the algorithm is to add or drop covariate one-at-a-time in the active set (non zero covariates). At each step, coefficients will be updated in making a step in the equiangular direction of the most correlated covariates until a new covariate is added or dropped*"
[Source HDPenReg Vignette].

The MixAll package

The Clustering project in STK++ implements a set of mixture model allowing to perform clustering on various data set using generative models. There is four kind of generative models implemented:

1. the diagonal Gaussian mixture models (8 models),
2. the diagonal gamma mixture models (24 models),
3. the diagonal categorical mixture models (8 models),
4. the diagonal Poisson mixture models (6 models).

These models and the estimation algorithms can take into account missing values. It is thus possible to use these models in order to cluster, but also to complete data set with missing values.

```
> library(MixAll)
> data(HeartDisease.cat)
> data(HeartDisease.cont)
> ldata = list(HeartDisease.cat,HeartDisease.cont);
> lnames = c("categorical_pk_pjk","gaussian_pk_sjk")
> model <- clusterHeterogeneous(ldata, lnames, nbCluster=2:7, strategy = clusterSemiSEMStrategy
      ())
> summary(model)
************************************************************
* nbSample        =  303
* nbCluster       =  6
* lnLikelihood    =  -4746.767
* nbFreeParameter=  155
* criterion       =  10487.07
************************************************************
> missingValues(model)
[[1]]
     row col value
[1,] 167   7     1
[2,] 193   7     1
[3,] 288   7     1
[4,] 303   7     1
[5,]  88   8     1
[6,] 267   8     1

[[2]]
     row col value
```

Sommaire

STK++ and rtkpp: The Future

The core code in STK++ is mature but still the status of STK++ is "beta"

- improvements can be achieved in order to compete better with the best arrays libraries (Armadillo and Eigen)
- Some important functionalities (less often needed in statistics) like sparse matrices and complex type will be implemented (to come in 0.9.0 version).

The real challenge for STK++/rtkpp is to cause the "snowball effect" in order to get a community of users and developers. This can be achieved by

- enhancing existing documentations, tutorials and examples, helping and reassuring potential users
- involving STK++/rtkpp in more research projects
- find private partners needing packaged solutions for statistical or learning problems
- find partnerships with the same interests for both C++ and Statistics
- ...

MERCI

www.stkpp.org

[2] Serge Iovleff (2015). rtkore: STK++ Core Library Integration to R using Rcpp. *http://cran.r-project.org/web/packages/rtkore/*

[3] Serge Iovleff (2015). MixAll: Clustering using Mixture Models. *http://cran.r-project.org/web/packages/MixAll/*

Inria

Lille Nord Europe