STK++
The Statistical ToolKit

Serge Iovleff
Equipe Projet Modal
Lille Nord-Europe
SOMMAIRE

What is a framework?

Genesis of STK++

Core functionalities

Utilities

Other Projects

STK++ by Examples

rtkpp: Using STK++ inside R

STK++: The Future
Sommaire

What is a framework ?

Genesis of STK++

Core functionalities

Utilities

Other Projects

STK++ by Examples

rtkpp: Using STK++ inside R

STK++: The Future
What is a framework?

A framework is not absolutely necessary: it is "just" one of the tools that is available to help to develop *better* and *faster*!

- **Better**? Because instead of writing something from scratch, developers basically just extend a given, working application. They benefit from the, usually well tested, work of dozens if not hundreds of other developers (not *already* the STK++ case...).

- **Faster**? because it allows developers to save time by re-using generic modules in order to focus on other areas.

Thus a framework increases application productivity by bundling the common elements together, which then form the guidelines for the team members to follow.

As a side-effect, Frameworks require the designer to do certain things in certain ways.
A framework in Statistics?

STK++ is a C++ Framework divided in 12 projects (libraries) oriented to statistics under LGPL license.

- Few other frameworks using the same language (C++) and the same license exists
  - ROOT: is mainly used in data analysis and data acquisition in high energy physics experiments. Its architecture is a layered class hierarchy with, currently, around 1200 classes grouped in about 60 frameworks (libraries) divided in 19 main categories (modules).
  - mlpack: is a C++ machine learning library built on top of the Armadillo library with emphasis on scalability, speed, and ease-of-use.

- Other well referenced frameworks/libraries:
  - gsl is in C and GPL (goose is defunct)
  - Eigen or Armadillo are oriented to linear algebra computation.

- There exists many softwares dedicated to a specific statistical problem (mainly in the machine learning community).
Sommaire

What is a framework ?

Genesis of STK++

Core functionalities

Utilities

Other Projects

STK++ by Examples

rtkpp: Using STK++ inside R

STK++: The Future
2002: Creation and participation to a start-up (Salford Partners France)

- distribution of data-mining software and consulting,
- mount a software development project (ADNS),
- find partners and supports from ANVAR région Nord and iSoft (Alice,...),
- start a prototype,
- but close the start-up for lacking customers in 2003.
Genesis of STK++

2005/2006: Get a CRCT ("Congés pour recherches ou conversions thématiques") and extend the prototype.
End of 2006: STK++ had the following functionalities implemented

- Structures and methods allowing to handle heterogeneous and missing values
- Flexible containers (see Array2D presentation)
- Implementation of the usual statistics (mean, variance,... and order statistics)
- Built-in implementation of the Svd, Qr and eigenvalues (for symmetric matrices) decompositions
- Flexible Read/write access to csv files.
Genesis of STK++ (3)

From 2006 to 2011: few functionalities added to the project
- gamma, gamma ratio and beta ratio special functions
- A reduced set of probabilities distribution laws
- Robust read/write access to option text files (ini style)
In 2011, creation of the MODAL Team by Christophe Biernacki.

- Many projects involving Statistics and C++ on the rails, but few used STK++. The reasons are
  - difficulties of comprehension (missing documentation,...)
  - functionalities and speed given by other libraries (Eigen)
  - disadvantage STK++ on benchmarks
  - lack of confidence

- In 2012, I get a six months delegation at Inria
  - adding missing functionalities (Template Expression, fast matrix product)
  - developing intensively regression and dimension reduction tools for the Auto-Associative models (see later)
  - writing more documentation.
Genesis of STK++ (5)

STK++ actually:
- A small part of the STK++ library is used by the console coclust program (in order to read the options file)
- Is used by the "aam" console program file
- Is used by the HDPenReg R package (a MODAl project)
- Can be used from R using the rtkpp package.
Sommaire

What is a framework?

Genesis of STK++

Core functionalities
- Missing values
- Handling Data with Array2D
- Using visitors (and Template MetaProgramming)
- Expression Templates
- Matrix Multiplication

Utilities

Other Projects

STK++ by Examples
Core functionalitites

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 (with template metaprogramming techniques),
4. Template Expression implementation
5. Optimized 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
3. Other types can also have missing values if they specialize the STK++ Arithmetic struct.

By default NA values are displayed with a "." but this can be dynamically modified using the stringNA variable.

```cpp
template<class Type>
void testNA()
{
  // Test Real
  Type x = Arithmetic<Type>::NA();
  std::cout << _T("x") << x << _T("\n");
  std::cout << _T("x") << Proxy<Type>(x) << _T("\n");
  Type y = STK::stringToType<Type>(stringNa); // by default stringNA is "."
  std::cout << _T("y") << y << _T("\n");
  std::cout << _T("y") << Proxy<Type>(y) << _T("\n");
  Type z = Arithmetic<Type>::infinity();
  std::cout << _T("z") << z << _T("\n");
  std::cout << _T("z") << Proxy<Type>(z) << _T("\n");
}
```
## Handling Missing values (2)

The piece of code

```cpp
std::cout << _T("test Real type\n");
testNA<Real>();
std::cout << _T("test int type\n");
testNA<int>();
```

produces the output

```
test Real type
x=nan
x=.
y=nan
y=.
z=inf
z=inf

test int type
x=-2147483648
x=.
y=-2147483648
y=.
z=0
z=0
```

And the piece of code

```cpp
std::cout << _T("test Binary type\n");
testNA<Binary>();
std::cout << _T("test Range type\n");
testNA<Range>();
```

produces the output

```
test Binary type
x=.
x=.
y=.
y=.
z=0
z=0

test Range type
x=.
x=.
y=.
y=.
z=0: -1
z=0: -1
```
Handling Missing values (2)

It becomes possible to handle the missing values in arrays in a transparent way. Some examples

1. Count missing values:

```cpp
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:

```cpp
std::cout << "meanSafe(a)= " << meanSafe(a);
std::cout << "meanSafeByRow(a)= " << meanSafeByRow(a);
std::cout << "varianceSafe(a)= " << Stat::varianceSafe(a);
std::cout << "varianceSafeByRow(a)= " << Stat::varianceSafeByRow(a);
```

```
meanSafe(a)= 2 1 4 3
meanSafeByRow(a)= 1.5 5
varianceSafe(a)= 4 0 4 0
varianceSafeByRow(a)= 1.25 1
```

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

4. ...
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

```cpp
#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 =
") << A;
    // Adding a column with 1
    A.pushFrontCols(ConstVectorX(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 =
") << A;
}
```

produces the output

```
A =
0.31492  -3.83202
1.70605   -2.77265
-0.132078  0.558416
-0.299732   -0.228056
A =
1  0.31492  0.0991748  0.0312322  -3.83202  14.6843  -56.2706
1  1.70605   2.91061   4.96566  -2.77265   7.68758  -21.315
1  1.70605   2.91061   4.96566  -2.77265   7.68758  -21.315
1  1.70605   2.91061   4.96566  -2.77265   7.68758  -21.315
1 -0.132078  0.0174446   -0.00230404   0.558416   0.311828   0.17413
1 -0.299732   0.0898394  -0.0269278  -0.228056   0.0520093  -0.011861
```
Handling data with Array2D

Adding columns is linear in time in sizeCols. When creating the array, STK++ add a small amount of available rows. Using the reserve method before modifying the container is also wise if you know how many rows you will add.
Using visitors (and Template MetaProgramming)

"Template metaprogramming allows the programmer to focus on architecture and delegate to the compiler the generation of any implementation required by client code. Thus, template metaprogramming can accomplish truly generic code, facilitating code minimization and better maintainability" - Wikipedia

This technique is used throughout the Arrays project. For example, the piece of code

```cpp
Stat::Law::Gamma law(1.5, 6);
CArrayVector<Real, UnknownSize> a(5); a.rand(law);
CArrayVector<Real, 5> b; b.rand(law)
```

is equivalent to

```cpp
Stat::Law::Gamma law(1.5, 6);
CArrayVector<Real, UnknownSize> a(5);
for (int i = a.begin(); a.end(); i++) {a[i] = law.rand();}
CArrayVector<Real, 5> b;
b[baseIdx] = law.rand(); b[baseIdx+1] = law.rand(); b[baseIdx+2] = law.rand();
b[baseIdx+3] = law.rand(); b[baseIdx+4] = law.rand();
```
Template MetaProgramming decrypted

The `rand` method is implemented as

```cpp
/* set random values to this using a law given by the user */
template<typename Derived>
void ArrayBase<Derived>::rand(Law::IUnivLaw<Type> const& law)
{
    hidden::RandApplier<Type> visitor(law);
    apply(visitor);
}
```

and the `apply` method is a templated class implemented as

```cpp
/* Apply the Visitor @a visitor to the whole coefficients of the array.
 * The template parameter @a Visitor is the type of the visitor and provides
 * the following interface:
 * @code
 * struct MyVisitor {
 *     // called for all coefficients
 *     void operator()(Type& value);
 * };  
 * @endcode
 * @note visitors offer automatic unrolling for small fixed size matrix.
 * @sa setValue */
template<typename Derived>
template<typename Visitor>
void ArrayBase<Derived>::apply(Visitor& visitor)
{
    typedef typename hidden::VisitorSelector<Visitor, Derived>::Impl Impl;
    Impl::apply(this->asDerived(), visitor);
}
```
Template MetaProgramming decrypted (suite 1)

In this example Visitor is hidden::RandApplier<Real>, and Derived is either CArrayVector<Real, UnknownSize> or CArrayVector<Real, 5>.

```cpp
/** @ingroup hidden *
 * @brief visitor selector. If @c Derived is a full two-dimensional array and
 * the visitation can be unrolled, then we use directly the VisitorArrayImpl
 * class in order to compute the result of the visitation.
 *
 * Otherwise we delegate the search of the correct implementation to the
 * VisitorSelectorHelper class.
 */

template<typename Visitor, typename Derived>
struct VisitorSelector
{
    enum
    {
        structure_ = hidden::Traits<Derived>::structure_,
        orient_ = hidden::Traits<Derived>::orient_,
        sizeRows_ = hidden::Traits<Derived>::sizeRows_,
        sizeCols_ = hidden::Traits<Derived>::sizeCols_,
        storage_ = hidden::Traits<Derived>::storage_,
        unrollRows_ = (sizeRows_ < MaxUnrollSquareRoot),
        unrollCols_ = (sizeCols_ < MaxUnrollSquareRoot),
        is2D_ = (structure_ == (int)Arrays::array2D_ || structure_ == (int)Arrays::square_)
    };

    typedef typename VisitorSelectorHelper<Visitor, Derived, structure_>::Impl HelperImpl;
    typedef VisitorArrayUnrollImpl<Visitor, Derived, orient_, sizeRows_, sizeCols_> ArrayImpl;
    // the other cases will be take into account by the helper class
    typedef typename If<is2D_ & & unrollRows_ & & unrollCols_, ArrayImpl, HelperImpl >::Result Impl;
};
```
Template MetaProgramming decrypted (suite 2)

The structure of of a CArrayVector is allways a Arrays::vector_ the implementation to choose is now determined by the compiler, either as VisitorVectorImpl<Visitor, Derived, UnknownSize> for vector a, or VisitorVectorImpl<Visitor, Derived, sizeRows_> for vector b (MaxUnroll is 20).

```cpp
/** *
 * @ingroup hidden *
 * @brief specialization for the vectors */

template< typename Visitor, typename Derived >
struct VisitorSelectorHelper< Visitor, Derived, Arrays::vector_ >
{
    enum
    {
        sizeRows_ = hidden::Traits< Derived >::sizeRows_,
        unrollRows_ = ( sizeRows_ < MaxUnroll )
    };
    typedef VisitorVectorImpl< Visitor, Derived, sizeRows_> Unroll;
    typedef VisitorVectorImpl< Visitor, Derived, UnknownSize > Loop;
    typedef typename If< unrollRows_, Unroll, Loop >::Result Impl;
};
```
Template MetaProgramming is achieved if the size is unknown. For the vector \( \text{a} \), the compiler will compile the code given below:

```cpp
#include <vector>

template<
typename Visitor,
typename Derived,
int SizeRows_
>
struct VisitorVectorImpl;

template<
typename Visitor,
typename Derived
>
struct VisitorVectorImpl<Visitor, Derived, UnknownSize>
{
    inline static void run(Derived const& tab, Visitor& visitor)
    {
        for(int i = tab.begin(); i < tab.end(); ++i)
            visitor(tab.elt(i), i, tab.colIdx());
    }

    inline static void apply(Derived& tab, Visitor& applier)
    {
        for(int i = tab.begin(); i < tab.end(); ++i)
            applier(tab.elt(i));
    }
};
```

Iovleff - The Statistical ToolKit
Using visitors (and Template MetaProgramming)

On this benchmark, the number of simulation has been repeated $100 \times 1000000$ for each kind of arrays.
The key idea behind expression templates is lazy evaluation of expressions.

"In programming language theory, lazy evaluation, or call-by-need is an evaluation strategy which delays the evaluation of an expression until its value is needed" - Wikipedia.

For example the STK++ code

```cpp
CVectorX AC(m), BC(m), CC(m);
CVectorX RC = AC + BC - CC;
```

The expression AC + BC - CC is encoded in a structure and evaluation is done only when the operator = is encountered by the compiler. If optimization is enabled, the previous code is equivalent to

```cpp
CVectorX AC(m), BC(m), CC(m);
CVectorX RC;
RC.resize(AC.range());
for (int i=AC.begin(); i<AC.end(); i++)
{RC[i] = AC[i] + BC[i] - CC[i];}
```
the compiler start parsing the right hand side expression. The instruction AC + BC is resolved as a

```
BinaryOperator< SumOp<TYPE, typename hidden::Traits<Rhs>::Type>, Derived, Rhs> const
```

```
BinaryOperator< SumOp<Real, Real>, VectorX, VectorX>
```

with Derived the type of AC and Rhs the type of BC. The functor SumOp is defined in the STKernel project as

```
/** @ingroup Functors *
 * @brief Template functor which compute the sum of two numbers *
 */
template<class Type1, class Type2 = Type1>
struct SumOp
{
  enum { NbParam_ = 2 };  
  typedef typename hidden::Promote<Type1, Type2>::result_type result_type; 
  typedef Type1 param1_type;  
  typedef Type2 param2_type;  
  
  inline result_type const operator()(param1_type a, param2_type b) const
  { return a + b; }
};
```

and just sum two numbers when called.
Expression Templates (suite 2)

The instruction \( AC+BC+CC \) is then resolved as a

\[
\text{BinaryOperator}< \text{SumOp}<\text{Real, Real}>, \text{BinaryOperator}< \text{SumOp}<\text{Real, Real}>, \text{VectorX}, \text{VectorX}>, \text{VectorX}>
\]

Finally the \text{BinaryOperator} class is specialized for vector by vector operations and an element of this structure is accessed with the instruction

```cpp
/** access to the element i */
inline typename BinaryOp::result_type const elt1Impl(int i) const
{ return this->asDerived().functor()( this->asDerived().lhs().elt(i), this->asDerived().rhs().elt(i));}
```

When compiler inspect, the piece of code \text{CVectorX RC = ...}, it will see that the operator \( = \) can be used with

```cpp
/** @return the matrix or vector obtained by evaluating this expression */
template<typename Rhs>
inline Derived& operator=( ExprBase<Rhs> const& rhs) { return assign(rhs.asDerived());}
```

as If optimization is enabled, the compiler will inline all operations and (eventually) vectorized the operations.
Expression Templates: Benchmark 1

```cpp
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 of size $m$

It seems computation is not vectorized for `CVector` (To fix!).
Expression Templates: Benchmark 1 bis

```cpp
Eigen::VectorXd Reig = 2*AEig.array() + 3*BEig.array().square() - CEig.array().abs().log();
arma::vec RA = 2*AA + 3*arma::square(BA) - arma::log(arma::abs(CA));
VectorX VectorX R2D = 2*A2D + 3*B2D.square() - C2D.abs().log();
CVectorX RC = 2*AC + 3*BC.square() - CC.abs().log();
```

Computational time of the expression $2 \cdot A + 3 \cdot B^2 - \log |C|$ with $A$, $B$, $C$ vectors of size $m$ and operations $^2$, $|$ and $\log$ applied on all coefficients of the vectors.
Expression Templates: Benchmark 2

Computational time of the expression $A + B + C$ with $A$, $B$, $C$ matrices of size $(m, 4)$
Expression Templates: Benchmark 3

\[
\text{Eigen::ArrayXXd } \text{Reig} = \text{AE.matrix()} - \text{Eigen::ArrayXd::Ones(m).matrix()} * \text{AE.colwise().mean().matrix();}
\]

\[
\text{arma::mat } \text{RA} = \text{AA} - \text{arma::ones(m)} * \text{arma::mean(AA)};
\]

ArrayXX R2D = AD - Const::Vector<STK::Real>(m) * Stat::mean(AD);
CArrayXX RC = AC - Const::Vector<STK::Real>(m) * Stat::mean(AC);

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

Matrice \( A(m, 10) \) computation of \( A - 1_m \bar{A}' \)
Matrix Multiplication

The amount of movements of data between memory layers is as much important as the amount of arithmetic operations performed on that data.
In current hardware environments different memory units can have slower access times thus minimizing data movements between such layers is extremely important problem.
Matrix Multiplication

On the first level of decomposition the operations are performed on sub-matrices, with one dimension dominating the other. Such matrices are called panels.

On the second level of decomposition the data is separated into smallest blocks of such size which guarantees efficient in-cache location of the operands.
Matrix Multiplication: Benchmark

```cpp
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 $A \times B \times C \times D$ with parallelization enabled.
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 $A \times B \times C \times D$ without parallelization
Sommaire

What is a framework ?

Genesis of STK++

Core functionalities

Utilities

DManager picked example: Read configuration/option files
Analysis picked example: finding zeros
STatistiK picked example: Generating random numbers

Other Projects

STK++ by Examples

rtkpp: Using STK++ inside R
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, compute 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 !)
DManager picked example: Read configuration files

Configuration files using a ini-like syntax are treated by dedicated classes in DManager project. Here is an example of a (part of a) configuration file used by the aam program

```
[INPUT]
FILE NAME = ./test/spect_data.csv
FILE FORMAT = csv
READ NAMES = true
DELIMITERS = ,
TYPE IMPORT = numeric

[MODEL]
# Does data should be standardized ? Default is false
standardize = false
#standardize = true

# type of dimension reduction. Optional: default is localVariance.
type reduction = totalVariance
# type of regression. Optional: default is additive.
# linear for a linear regression,
# additiveBSpline for an additive Bspline regression
type regression = additiveBSpline

dims = 1:17
# Criteria to use in order to select the model. Optional: default is BIC.
# criteria can be BIC or AIC.
criteria = BIC
[[LOCALVARIANCE]]
type graph = minimalDistance
neighborhood = 6
```
The corresponding code is the constructor of the `ModelPage` class reading a given configuration file

```cpp
/* constructor. */
ModelPage::ModelPage(): IPage(_T("Model"), 1, true)
    , standardize_(false)
    , typeReduction_(Reduct::localVariance_)
    , typeRegression_ (Regress::additiveBSpline_)
    , dims_ (Range(2,1))
    , typeCriterion_ (Model::bic_)
{
    options_.reserve(5);
    options_.push_back(Option(_T("standardize"), Option::string_, true));
    options_.back().setValue(_T("false"));
    options_.push_back(Option(_T("type reduction"), Option::string_, true));
    options_.back().setValue(_T("localVariance"));
    options_.push_back(Option(_T("type regression"), Option::string_, true));
    options_.back().setValue(_T("additive"));
    options_.push_back(Option(_T("dims"), Option::range_, true));
    options_.back().setValue(_T("2:2"));
    options_.push_back(Option(_T("criteria"), Option::string_, true));
    options_.back().setValue(_T("BIC"));
}
```

All the parsing, error treatments, etc. is done by the interface base class `IPage`. 
Analysis picked example : finding zeros

In numerical analysis, Brent’s method is a complicated but popular root-finding algorithm combining the bisection method, the secant method and inverse quadratic interpolation. It is implemented with the following prototype:

```cpp
template <class Function
Real BrentMethod( IFunction<Function> const & f, Real const & x0, Real const & x1, Real tol);
```

The user have to implement the derived class of `IFunction` by implementing the following interface class:

```cpp
template <class Function
class IFunction : public IRecursiveTemplate<Function>
{
    protected:
        inline IFunction() {}
        inline ~IFunction() {}
    public:
        inline Real operator()(Real const & x) const { return this->asDerived().fImpl(x);}
        inline Real xmin() const { return this->asDerived().xminImpl();}
        inline Real xmax() const { return this->asDerived().xmaxImpl();}
        //** @return Default implementation of the minimal value */
        inline Real xminImpl() const { return -Arithmetic<Real>::max();}
        //** @return Default implementation of the maximal value */
        inline Real xmaxImpl() const { return Arithmetic<Real>::max();}
};
```
Consider the Mixture gamma mixture model $\text{gamma}_a\_j\_b\_jk$ with density function on $\mathbb{R}^d$

$$f(x_i|\theta) = \sum_{k=1}^{K} p_k \prod_{j=1}^{d} \prod_{j=1}^{d} \left( x_j \right)^{a_j-1} e^{-x_j/b_{jk}} \Gamma(a_j)(b_{jk})^{a_j}.$$ 

Given a sample, the parameters can be estimated using the iterative EM algorithm. At each iteration, the update formulas for the estimators are given by

$$\begin{cases} 
\Psi(\hat{a}_j) & = \frac{1}{n} \sum_{k=1}^{K} t.k \left( (\log(x))^j_k - \log(\hat{b}_{jk}) \right) \\
\hat{b}_{jk} & = \frac{x_k^j}{\hat{a}_j}, \end{cases}$$

The parameter $\hat{a}_j$ solve an equation of the form

$$y - \Psi(a) + \log(a) = 0$$
finding zeros example

In the **Clustering** project, we can find the following implementation

```c++
/** @ingroup hidden
 * Functor computing the derivative of the lnLikelihood of a gamma_ajk_bjk model */
class invPsiMLog : public IFUn
cion<invPsiMLog >
{
  public:
    inline invPsiMLog( Real const & y): y_(y) {}
    /* @return the value of the function at a
     * @param a a positive real value
     ***/
    inline Real fImpl(Real const & a) const
    { return (y_ + std::log(a) - Funct::psi_raw(a));}
    /* @return the minimal value of the function at x */
    inline Real xminImpl() const { return 0;}
  private:
    Real y_; 
};
```

which is used elsewhere as

```c++
hidden::invPsiMLog f(y);
Real a = Algo::findZero(f, x0, x1, 1e-08);
if (!Arithmetic<Real>::isFinite(a))
{
  a = x0; // use moment estimate
}
// set values
shape_[j] = a;
```
Concrete distribution laws implemented in the **STatistiK** project derive from the pure interface class

```cpp
template <class Type>
class IUnivLaw : public ILawBase
{
    protected:
        IUnivLaw(String const& name) : ILawBase(name) {}
        IUnivLaw(IUnivLaw const& law) : ILawBase(law.name_) {}

    public:
        inline virtual ~IUnivLaw() {}
        virtual Type rand() const =0;
        virtual Real pdf(Type const& x) const =0;
        virtual Real ldpdf(Type const& x) const = 0;
        virtual Real cdf(Real const& t) const =0;
        virtual Type icdf(Real const& p) const=0;
};
```

Currently the implemented laws (in some cases only partially) are the **Bernoulli, Beta, Cauchy, Exponential, Gamma, Normal, Poisson, Uniform** probabilities.
Utilities

STatistiK picked example : Generating random numbers

```cpp
std::random_device rd;
std::mt19937 gen(rd());
std::normal_distribution<> d(0,1);
Law::Normal law(0,1.);
A.rand(law);
for (int j=B.begin(); j< B.end(); ++j) B[j] = d(gen);
```

There exists many open-source or not-licensed code everywhere in internet. Most of the existing code is an adaptation of this already (open-source) available code, selected by speed and numerical stability criteria.

Vector $A(m)$ computation of $A.rand(N(0, 1))$
Sommaire

What is a framework ?

Genesis of STK++

Core functionalities

Utilities

Other Projects

STK++ by Examples

rtkpp: Using STK++ inside R

STK++: The Future
Other Projects

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 (see STK++ By example)

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
A word about Design Pattern

In software engineering, a design pattern is a general reusable solution to a commonly occurring problem within a given context in software design. [...] It is a description or template for how to solve a problem that can be used in many different situations. [source Wikipedia]

In all these projects the design pattern used is mostly abstract or concrete factory with (a naive) command pattern.

The essence of the Abstract Factory pattern is to "Provide an interface for creating families of related or dependent objects without specifying their concrete classes."

"Manipulating a family of objects without specifying their concrete class" means they all behave similarly and derive from a single interface.
What is an interface?

In the Reduct project the main interface class is (almost equivalent to)

```cpp
class IReduct
{
  protected:
    IReduct();
    IReduct( ArrayXX const data);
    IReduct( IReduct const reductor);

  public:
    virtual ~IReduct();
    virtual bool run() = 0;
    virtual bool run( Vector const & weights) = 0;

    inline int dim() const { return dim_;}
    inline ArrayXX* p_reduced() const { return p_reduced_; }
    inline void setDimension( int const & dim) { dim_ = dim;}

  protected:
    int dim_; 
    ArrayXX* p_reduced_; 
};
```

A concrete dimension reduction technique implement this Interface. A call to the pure virtual method `run()` will fill the array `p_reduced_` with the result.
Example of an Abstract Factory

In the Clustering project, the class IMixtureComposer is an abstract factory allowing to handle generative mixture models

class IMixtureComposer : public IStatModelBase
{
    protected:
        IMixtureComposer( int nbSample, int nbCluster);
        ...
    public:
        ...
        inline CPointX const & pk() const { return prop_;};
        inline CArrayXX const & tik() const { return tik_;};
        ...
        Real computeICL() const;
        ...
        virtual void mStep() = 0;
        ...
        void randomClassInit();
        int cStep();
        int sStep();
        Real eStep();

    protected:
        CPointX prop_; 
        CArrayXX tik_; 
        ...
};

Concrete composer will implement (complete) this interface in a convenient way. The type of data to cluster can lead to different implementations.
Sommaire

What is a framework ?

Genesis of STK++

Core functionalities

Utilities

Other Projects

STK++ by Examples
  The HDPenReg package
  The aam program
  The aam program

rtpkpp: Using STK++ inside R
The HDPenReg 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 algorithm Lars adds/removes columns to a matrix at each step and update a Qr decomposition. It use the following variables

```cpp
// qr decomposition of Xi
STK::lapack::Qr qrX_;  
// X reduced to covariates of active Set
STK::Array2D<STK::Real> Xi_; 
```

which are used somewhere in the code

```cpp
Xi_.pushBackCols(1);  
Xi_.col(Xi_.lastIdxCols()) = X_.col(idxVar);  
qrX_.pushBackCol(Xi_.col(Xi_.lastIdxCols()));

if( std::abs(qrX_.R()( min(n_,nbActiveVariable_+1), nbActiveVariable_+1) ) < eps_ )
{
  qrX_.popBackCols();
  toIgnore_[idxVar-1]=true; // the variable is add to the ignore set
  nbIgnoreVariable_++;
  Xi_.popBackCols(1);
} 
else ... 
```
The aam program

Definition
A function $g$ is an auto-associative function of dimension $d$ if it is a map from $\mathbb{R}^p$ to $\mathbb{R}^p$ that can be written as $g = R \circ P$ where $P$ (the “Projection”) is a map from $\mathbb{R}^p$ to $\mathbb{R}^d$ (generally $d < p$) and $R$ (the “Restoration” or the ”Regression”) is a map from $\mathbb{R}^d$ to $\mathbb{R}^p$.

An auto-associative model (AAM) of dimension $d$ is a manifold $\mathcal{M}_g$ of the form

$$\mathcal{M}_g = \{ y \in \mathbb{R}^p, \ y - g(y) = 0 \}$$

where $g$ is an auto-associative function of dimension $d$.

- The main idea under this definition is that data in the big-space can be explained using information contained by some smaller linear-space (linear projection), or manifold (non linear projection).
- There is as many Auto-Associative models as there exits projection and regression methods.
The aam program

The aam is a console based program based on the library STK++ but not directly integrated to the library.

- In order to write it I had to add two projects to the library: Regress and AAModel and to enhance the Reduct project.
- The program itself makes also use of the DManager and StatModels projects.
- The AAModel project is essentially composed of two factory. A first abstract factory called IAAModel take care of the geometry of the model. The second factory GaussianAAModel derived from the previous assumes Gaussianity of the residuals.
A factory for the auto-associative models

The IAAModel factor look

class IAAModel
{
    protected:
    IAAModel( ArrayXX & workData);
    ~IAAModel();

    public:
    ...
    void setDimension( int const & dim);
    void setWorkData( ArrayXX & workData);
    void setReductor( IReduct * p_reductor);
    void setRegressor( Regressor* p_regressor);

    protected:
    Regressor* p_regressor_;
    IReduct* p_reductor_;    
    ArrayXX* p_workData_;    
    ArrayXX* p_reduced_;      
    ArrayXX* p_predicted_;    
    ArrayXX* p_residuals_;   
};

The classes Regressor and IReduct
Sommaire

What is a framework ?

Genesis of STK++

Core functionalities

Utilities

Other Projects

STK++ by Examples

rtpkp: Using STK++ inside R

STK++: The Future
R and the Cran is becoming the main way to diffuse new methods. In my research team (MODAL) almost all publications in the clustering area 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 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.
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:

```cpp
// 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 Rcpp-extending]
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` rtkpp class

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

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

What is a framework ?

Genesis of STK++

Core functionalities

Utilities

Other Projects

STK++ by Examples

rtkpp: Using STK++ inside R

STK++: The Future
STK++: 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)
- The Regress and Reduct projects can only be used with Array2D structures and have to be adapted
- Some important functionalities (less often needed in statistics) like sparse matrices and complex data should be implemented.

The real challenge for STK++ 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++ in more Inria research projects
- find private partners needing packaged solutions for statistical or learning problems
- find partnerships with the same interests for both C++ and Statistics
- advertising ?
MERCI

Inria
Lille Nord Europe
www.stkpp.org