Statistical Scientific programming: challenges in converting R to C++

Olivia Quinet
olivia.quinet @cluepoints.com

Meeting C++
BeCPP July 2019
Introduction
CluePoints
Clinical Trials
Statistical tests
SMART package

The R language
A very short introduction to R
Some examples
R to C++?

Scientific programming challenges
Testing
Measure
Software architecture
Data structure
Smart pointers, Pimpl, Factories
Fail-fast/Fail-safe
Numerical errors
Accumulators
std::algorithms, boost, GSL, BLAS, LAPACK, ...

Conclusion
Questions, Remarks?

Credit xkcd
CluePoints is the premier provider of Risk-Based Monitoring and Data Quality Oversight Software. Our products utilize unique statistical algorithms to determine the quality, accuracy, and integrity of clinical trial data both during and after study conduct.
Clinical Trials

Why?

- Is it working?
- Is it safe?
- Scope, dosage, ...
- Approval from FDA, EMA, ...
Clinical Trials

How?

- Clinical protocol
- Study conducted at sites
- Patients are enrolled
- Data is collected: demographics, medical history, vital signs, adverse events, labs, patient journals, ...
- Data is verified and analyzed
Clinical Trials

 $$$?

- 1.5-2.5 billion on 10-plus years
- 30% for sending investigators on sites
Statistical tests

Medical Records → Statistical Test → Matrix of p-values → Atypical sites, patients
SMART package

- Initially developed in the R language by the R&D team
SMART package

- Initially developed in the R language by the R&D team
- Very good for research purposes
SMART package

- Initially developed in the R language by the R&D team
- Very good for research purposes
- Not so much for production
SMART package

- Initially developed in the R language by the R&D team
- Very good for research purposes
- Not so much for production
- Need for something reliable, robust and fast
SMART package

- Initially developed in the R language by the R&D team
- Very good for research purposes
- Not so much for production
- Need for something reliable, robust and fast
The R language
The R language

- R is a programming language for statisticians created by statisticians
- R is weakly/dynamically typed
- R operates on named data structures: vector, matrix, array, data frame, factors, lists, objects, functions
- It is very concise
- Lot of statistical libraries
Some examples (I)

1. \( w = \text{!is.na}(d[[\text{field}]]) \);
2. \( \text{ctr} = \text{factor}(d$center[w]) \);
3. \( \text{npat} = \text{unclass}(\text{table}(\text{ctr})) \);
4. \( v = d[w, \text{field}] \);
5. \( y = \text{rowsum}(v, \text{ctr}) \);
Some examples (I)

1. Select the rows where the values of the column "field" are not missing
2. Get the values as factor of the column "center" for the selected rows, i.e. the list of centers
3. Count the number of rows associated to the different centers, i.e. the number of patients per center
4. Get the values for the column "field" for the selected rows
5. Sum by center the values from (4)
Some examples (I)

```r
w = !is.na(d[[field]]);
ctr = factor(d$center[w]);
npat = unclass(table ctr));
v = d[w, field];
y = rowsum(v, ctr);
```

<table>
<thead>
<tr>
<th>center</th>
<th>xyz</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctr01</td>
<td></td>
</tr>
<tr>
<td>ctr02</td>
<td>1</td>
</tr>
<tr>
<td>ctr01</td>
<td>2</td>
</tr>
<tr>
<td>ctr03</td>
<td>3</td>
</tr>
<tr>
<td>ctr05</td>
<td>4</td>
</tr>
<tr>
<td>ctr02</td>
<td></td>
</tr>
<tr>
<td>ctr02</td>
<td>5</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ctr</th>
<th>npat</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctr01</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ctr02</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>ctr03</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Some examples (II)

1. $xc = x - \text{offset}$;
2. $v = \text{tapply}(xc, ctr, \text{mean}, \text{na.rm}=\text{T})$;
3. $Sn = \text{unclass}((\text{table})(ctr))$;
4. $Sn2 = \text{tapply}(sid, ctr, \text{function}(i) \text{sum}(\text{table}(i)^2))$;
5. $\text{sigma} = \sqrt{\text{Sn}*\text{vc}[3]^2 + \text{Sn2}*\text{vc}[2]^2 + \text{Sn}^2*\text{vc}[1]^2}/\text{Sn}$;
6. $p = \text{pnorm}(v, sd=\text{sigma})$
Some examples (II)

1. Apply an offset to x
2. Compute per center the mean of xc
3. Number of records per center
4. Compute per center the sum of the squares of the number of values per patient
5. Compute sigma
6. Compute the p-values for each center based on a normal distribution

```r
xc = x - offset;
v = tapply(xc, ctr, mean, na.rm=T);
Sn = unclass(table(ctr));
Sn2 = tapply(sid, ctr, function(i) sum(table(i)^2));
p = pnorm(v, sd=sigma)
```
Some examples (II)

```r
xc = x - offset;
v = tapply(xc, ctr, mean, na.rm=T);
Sn = unclass(table(ctr));
Sn2 = tapply(sid, ctr, function(i) sum(table(i)^2));
p = pnorm(v, sd=sigma)
```

<table>
<thead>
<tr>
<th>center</th>
<th>subj</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctr01</td>
<td>s01001</td>
<td>1</td>
</tr>
<tr>
<td>ctr02</td>
<td>s02001</td>
<td>2</td>
</tr>
<tr>
<td>ctr03</td>
<td>s03001</td>
<td>3</td>
</tr>
<tr>
<td>ctr05</td>
<td>s05001</td>
<td>4</td>
</tr>
<tr>
<td>ctr02</td>
<td>s02002</td>
<td></td>
</tr>
<tr>
<td>ctr02</td>
<td>s02001</td>
<td>5</td>
</tr>
<tr>
<td>ctr02</td>
<td>s02001</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ctr</th>
<th>Sn</th>
<th>Sn2</th>
<th>sigma</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctr01</td>
<td>1</td>
<td>1</td>
<td>0.37</td>
<td>0.21</td>
</tr>
<tr>
<td>ctr02</td>
<td>3</td>
<td>5</td>
<td>0.25</td>
<td>0.55</td>
</tr>
<tr>
<td>ctr03</td>
<td>5</td>
<td>5</td>
<td>0.19</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Some examples (III)

1  dd = duplicated(d$subjid);
2  v = d[[field]]
3  w = dd & c(FALSE, v[1:(length(v)-1)]==1);
4  x10 = rowsum(1-v[w], ctr[w]);
5  N10 = unclass(table(ctr[w]));
6  x10Max = rowsum(as.integer(!c(TRUE, w[1:(length(w)-1)])[w]), ctr[w]);
Some examples (III)

1. Create a boolean vector indicating if a subjid is duplicated or not
2. Get the values of the column "field"
3. Do some weird selection
4. Get the number of transitions 1→0 per center for each patient
5. Get the number of potential transitions 1→0 per center
6. Get the maximum number of valid transitions 1→0 per center

```
1  dd = duplicated(d$subjid);
2  v = d[[field]]
3  w = dd & c(FALSE, v[1:(length(v)-1)]==1);
4  x10 = rowsum(1-v[w], ctr[w]);
5  N10 = unclass(table(ctr[w]));
6  x10Max = rowsum(as.integer(!c(TRUE, w[1:(length(w)-1)])[w]), ctr[w]);
```
Some examples (III)

1. \(dd = \text{duplicated}(d$subjid)\);
2. \(v = d[[\text{field}]]\);
3. \(w = dd \& c(\text{FALSE}, v[1:(\text{length}(v)-1)]==1)\);
4. \(x10 = \text{rowsum}(1-v[w], ctr[w])\);
5. \(N10 = \text{unclass}(\text{table}(ctr[w]))\);
6. \(x10\text{Max} = \text{rowsum}(\text{as.integer}(!c(\text{TRUE}, w[1:(\text{length}(w)-1)])[w]), ctr[w])\);

<table>
<thead>
<tr>
<th>center</th>
<th>subjid</th>
<th>visit</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctr01</td>
<td>s01001</td>
<td>v01</td>
<td>1</td>
</tr>
<tr>
<td>ctr01</td>
<td>s01001</td>
<td>v02</td>
<td>0</td>
</tr>
<tr>
<td>ctr01</td>
<td>s01001</td>
<td>v03</td>
<td>1</td>
</tr>
<tr>
<td>ctr01</td>
<td>s01001</td>
<td>v04</td>
<td>0</td>
</tr>
<tr>
<td>ctr01</td>
<td>s01002</td>
<td>v01</td>
<td>1</td>
</tr>
<tr>
<td>ctr01</td>
<td>s01002</td>
<td>v02</td>
<td>1</td>
</tr>
<tr>
<td>ctr02</td>
<td>s02001</td>
<td>v03</td>
<td>0</td>
</tr>
<tr>
<td>ctr02</td>
<td>s02002</td>
<td>v01</td>
<td>1</td>
</tr>
<tr>
<td>ctr02</td>
<td>s02002</td>
<td>v02</td>
<td>1</td>
</tr>
<tr>
<td>ctr02</td>
<td>s02002</td>
<td>v03</td>
<td>0</td>
</tr>
<tr>
<td>ctr03</td>
<td>s03001</td>
<td>v01</td>
<td>1</td>
</tr>
<tr>
<td>ctr03</td>
<td>s03001</td>
<td>v02</td>
<td>0</td>
</tr>
<tr>
<td>ctr03</td>
<td>s03001</td>
<td>v03</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ctr</th>
<th>X10</th>
<th>N10</th>
<th>x10\text{Max}</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctr01</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>ctr02</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ctr03</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

...
How to translate R code to C++ code?

- Straightforward approach: Recode each R function in C++
  - PRO C++ and R codes are similar
  - CON Too many combinations of parameters/structures
How to translate R code to C++ code?

- Straightforward approach: Recode each R function in C++
  - **PRO** C++ and R codes are similar
  - **CON** Too many combinations of parameters/structures

- Hard: understanding what the researcher wanted to do
  - **PRO** Faster code
  - **CON** C++ and R codes can be very different
    - 1 line in R $\rightarrow$ ±30 lines in C++
How to translate R code to C++ code?

- Straightforward approach: Recode each R function in C++
  
  **PRO**  C++ and R codes are similar
  
  **CON**  Too many combinations of parameters/structures

- Hard: understanding what the researcher wanted to do
  
  **PRO**  Faster code
  
  **CON**  C++ and R codes can be very different
  
  1 line in R $\longrightarrow \pm 30$ lines in C++

- Hardest: changing the data structure
  
  **PRO**  Less resource/faster code
  
  **CON**  C++ and R codes are even more different

- Beware of Numerical (in)accuracy

- Testing and testing and testing (no data, invalid data, NaN, Inf, ...)

- Recoding model fitting algorithms is a huge (tremendous) task. It's easier
to call the R function from the C++ code
  
  **PRO**  Updates of the fitting model code
  
  **CON**  Added dependencies
How to translate R code to C++ code?

- **Straightforward approach**: Recode each R function in C++
  - **PRO** C++ and R codes are similar
  - **CON** Too many combinations of parameters/structures

- **Hard**: understanding what the researcher wanted to do
  - **PRO** Faster code
  - **CON** C++ and R codes can be very different
    - 1 line in R $\rightarrow$ ±30 lines in C++

- **Hardest**: changing the data structure
  - **PRO** Less resource/faster code
  - **CON** C++ and R codes are even more different

- **Recoding model fitting algorithms is a huge (tremendous) task. It’s easier to call the R function from the C++ code**
  - **PRO** Updates of the fitting model code
  - **CON** Added dependencies
How to translate R code to C++ code?

- **Straightforward approach**: Recode each R function in C++
  - **PRO** C++ and R codes are similar
  - **CON** Too many combinations of parameters/structures

- **Hard**: understanding what the researcher wanted to do
  - **PRO** Faster code
  - **CON** C++ and R codes can be very different
    - 1 line in R → ±30 lines in C++

- **Hardest**: changing the data structure
  - **PRO** Less resource/faster code
  - **CON** C++ and R codes are even more different

- Recoding model fitting algorithms is a huge (tremendous) task. It’s easier to call the R function from the C++ code
  - **PRO** Updates of the fitting model code
  - **CON** Added dependencies

- Beware of Numerical (in)accuracy
How to translate R code to C++ code?

- Straightforward approach: Recode each R function in C++
  - PRO C++ and R codes are similar
  - CON Too many combinations of parameters/structures

- Hard: understanding what the researcher wanted to do
  - PRO Faster code
  - CON C++ and R codes can be very different
    - 1 line in R $\rightarrow$ ±30 lines in C++

- Hardest: changing the data structure
  - PRO Less resource/faster code
  - CON C++ and R codes are even more different

- Recoding model fitting algorithms is a huge (tremendous) task. It’s easier to call the R function from the C++ code
  - PRO Updates of the fitting model code
  - CON Added dependencies

- Beware of Numerical (in)accuracy

- Testing and testing and testing (no data, invalid data, NaN, Inf, ...)

PRO C++ and R codes are similar
CON Too many combinations of parameters/structures

PRO Faster code
CON C++ and R codes can be very different

PRO Less resource/faster code
CON C++ and R codes are even more different

PRO Updates of the fitting model code
CON Added dependencies

PRO C++ and R codes are similar
CON Too many combinations of parameters/structures

PRO Faster code
CON C++ and R codes can be very different

PRO Less resource/faster code
CON C++ and R codes are even more different

PRO Updates of the fitting model code
CON Added dependencies

PRO C++ and R codes are similar
CON Too many combinations of parameters/structures

PRO Faster code
CON C++ and R codes can be very different

PRO Less resource/faster code
CON C++ and R codes are even more different

PRO Updates of the fitting model code
CON Added dependencies

PRO C++ and R codes are similar
CON Too many combinations of parameters/structures

PRO Faster code
CON C++ and R codes can be very different

PRO Less resource/faster code
CON C++ and R codes are even more different

PRO Updates of the fitting model code
CON Added dependencies

PRO C++ and R codes are similar
CON Too many combinations of parameters/structures

PRO Faster code
CON C++ and R codes can be very different

PRO Less resource/faster code
CON C++ and R codes are even more different

PRO Updates of the fitting model code
CON Added dependencies
Scientific programming challenges
Scientific programming challenges

- Requirements include low response time and memory usage, minimizing numerical errors and error propagation.
- Testing
- Software architecture
- Data structure
- Fail-fast/Fail-safe idioms
- Exceptions
- RAII
- Pimpl idiom and smart pointers
- Factory pattern
- Iterator pattern and accumulators
- std::algorithms, boost, GSL, BLAS, LAPACK, …
Testing

"Not so fast Michael! It says here that you should write the test case before you can execute it."

Credit xkcd
Testing

- Framework
- Unit testing, Integration testing, ...
- *Test Driven Development*
- *Behavior Driven Development* to replicate the documentation specification
- Continuous Integration
Testing

- Framework
- Unit testing, Integration testing, ...
- *Test Driven Development*
- *Behavior Driven Development* to replicate the documentation specification
- Continuous Integration
- Each bug must be tested
Code for Testing

- If you cannot test your code, rewrite it
- If you cannot test your code easily, rewrite it
- If you cannot test your code independently, rewrite it
- ... 

Tools like clang static analyzer and gcov/lcov code coverage are a great help
Measure!!!

Credit xkcd
Measure!!!

- Select between different data structures
- Select between different algorithms
- Use generated data
- Use real data
- Use data of different sizes
- …
Measure!!! – an example

**Context:** Originally, an algorithm has to be applied on vectors: \( f(x, y) \)
Measure!!! – an example

**Context:** Originally, an algorithm has to be applied on vectors: $f(x, y)$
Then only on some filtered elements: $f(x, y, w)$

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>42.5</td>
<td>100</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Measure!!! – an example

**Context:** Originally, an algorithm has to be applied on vectors: \( f(x, y) \)
Then only on some filtered elements: \( f(x, y, w) \)

- Modify the algorithm to take into account only the filtered vectors’ elements: *filter algo*
- Create pseudo vectors with the filtered elements: *filter vector*
- Create new vectors with the filtered elements: *copy vector*
Measure!!! – an example

**Context:** Originally, an algorithm has to be applied on vectors: $f(x, y)$
Then only on some filtered elements: $f(x, y, w)$

- Modify the algorithm to take into account only the filtered vectors’ elements: *filter algo*
- Create pseudo vectors with the filtered elements: *filter vector*
- Create new vectors with the filtered elements: *copy vector*

<table>
<thead>
<tr>
<th>Option</th>
<th>$N = 10^2$</th>
<th>$N = 10^4$</th>
<th>$N = 10^6$</th>
<th>$N = 10^8$</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>filter algo</em></td>
<td>0.0003</td>
<td>0.008</td>
<td>0.9</td>
<td>100</td>
</tr>
<tr>
<td><em>filter vector</em></td>
<td>0.0003</td>
<td>0.006</td>
<td>0.8</td>
<td>98</td>
</tr>
<tr>
<td><em>copy vector</em></td>
<td>0.0006</td>
<td>0.015</td>
<td>4.6</td>
<td>/</td>
</tr>
</tbody>
</table>
Introduction

CluePoints
Clinical Trials
Statistical tests
SMART package

The R language
A very short introduction to R
Some examples
R to C++?

Scientific programming challenges
Testing
Measure
Software architecture
Data structure
Smart pointers, Pimpl, Factories
Fail-fast/Fail-safe
Numerical errors
Accumulators
std::algorithms, boost, GSL, BLAS, LAPACK, ...

Conclusion

Questions, Remarks?
If you build it, they will come

Yeah, I'm just writing the code now.

Software architecture & Data structure

▶ Input/output data structure?
▶ Computational units?
▶ Simple but not too simple!
▶ Which doors are you closing?
▶ Expressiveness

For this project

▶ Data is organized in datasets, i.e. tables in which each column represents a particular variable or key variable, and each row corresponds to a given record. There may also be missing values.

▶ Statistical tests are the computational units.
Software architecture & Data structure

**Important points to consider**

- Input/output data structure?
- *Computational units*?
- Simple but not too simple!
- Which doors are you closing?
- Expressiveness

For this project

▶ Data is organized in datasets, i.e. tables in which each column represents a particular variable or key variable, and each row corresponds to a given record. There may also be missing values.

▶ Statistical tests are the computational units.
Software architecture & Data structure

Important points to consider

- Input/output data structure?
- Computational units?
- Simple but not too simple!
- Which doors are you closing?
- Expressiveness

For this project

- Data is organized in datasets, i.e. tables in which each column represents a particular variable or key variable, and each row corresponds to a given record. There may also be missing values.
- Statistical tests are the computational units.
Levels of abstraction

- The **most** important good practice
- Divide and conquer
- *Top down* design
- *Bottom up* design
- *Separation of concerns*
- *Modularity: low coupling* $\leftrightarrow$ *high cohesion*
- *Design review*
Levels of abstraction – mathematical formula

\[ i\hbar \frac{\partial}{\partial t} \Psi(\vec{r}, t) = \left( -\frac{\hbar^2}{2m} \nabla^2 + V(\vec{r}, t) \right) \Psi(\vec{r}, t) \]

**WHAT PART OF**

**DON'T YOU UNDERSTAND?**
Levels of abstraction – mathematical formula

- Very tempting to code one mathematical formula into one function.
- Decompose the formula into meaningful steps, e.g. numerator, denominator, partial sums, ...
- Transform the function into a class
- Transform each step into a struct
Abstraction levels – Example

Sample variance – Standard formula

$$s_N^2 = \frac{1}{N - 1} \sum_{k=1}^{N} (x_k - \bar{x})^2$$
Abstraction levels – Example

Sample variance – Standard formula

$$s_N^2 = \frac{1}{N-1} \times \sum_{k=1}^{N} (x_k - \bar{x})^2$$

- Fraction
- Sum
- Square of difference
- Multiplication
- Mean

$$s_N^2 = \frac{1}{N-1} \times \sum_{k=1}^{N} (x_k - \bar{x})^2$$
Abstraction levels – Example

```cpp
namespace MATH_INTERNAL {
  template<typename T=double>
  struct sample_variance {
    T s2{std::numeric_limits<T>::quiet_NaN()};

    template<typename Container>
    sample_variance(const Container& X) { if(X.size()>1) s2 = frac(X)*sum(X, mean(X)); }

    operator T() const { return s2; }

    template<typename Container>
    static T frac(const Container& X) { return ONE/(X.size()-ONE); }

    template<typename Container>
    static T mean(const Container& X) { return std::accumulate(X.begin(), X.end(), ZERO)/X.size(); }

    struct square_of_difference {
      const T xbar;
      square_of_difference(const T mean) : xbar(mean) {} 
      T operator()(const T x) const { return (x-xbar)*(x-xbar); }
    };

    template<typename Container>
    static T sum(const Container& X, const T xbar) {
      const square_of_difference d(xbar);
      return std::accumulate(X.begin(), X.end(), ZERO, [d](const T s, const T x) { return s + d(x); });
    }
  }

  template<typename Container>
  inline typename Container::value_type sample_variance(const Container& X) {
    return MATH_INTERNAL::sample_variance<typename Container::value_type>(X);
  }
}
```
Data structure

- Performance requires well thought data structure
- Cache usage
- Prefetching
- Lazy evaluation
- Sparse representation
- ...
Data structure – Example

**Duplicate patients:** comparing patient’s fingerprints
Data structure – Example

- $N$ numerical vectors of the same length $M$
  
  Typical cases: $N = 1000 - 40000$ and $M = 20 - 20000$
Data structure – Example

- \( N \) numerical vectors of the same length \( M \)
  Typical cases: \( N = 1000 – 40000 \) and \( M = 20 – 20000 \)
- \( N \times (N - 1)/2 \) scalar products

\[
X \cdot Y = \sum_{i}^{M} X_i \times Y_j
\]
Data structure – Example

- $N$ numerical vectors of the same length $M$
  
  Typical cases: $N = 1000 - 40000$ and $M = 20 - 20000$

- $N \times (N - 1)/2$ scalar products

$$X \cdot Y = \sum_{i}^{M} X_i \times Y_j$$

- Sparse vectors!!!
Data structure – Example

- $N$ numerical vectors of the same length $M$
  Typical cases: $N = 1000 - 40000$ and $M = 20 - 20000$
- $N \times (N - 1)/2$ scalar products

$$X \cdot Y = \sum_{i}^{M} X_i \times Y_j$$

- Sparse vectors!!!
- Performance results

<table>
<thead>
<tr>
<th></th>
<th>$N = 841$</th>
<th>$N = 35613$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M$</td>
<td>1060</td>
<td>14304</td>
</tr>
<tr>
<td>Memory</td>
<td>57MB</td>
<td>9.8GB</td>
</tr>
<tr>
<td>Timing</td>
<td>0.68s</td>
<td>29m</td>
</tr>
<tr>
<td>Memory</td>
<td>34MB</td>
<td>6.6GB</td>
</tr>
<tr>
<td>Timing</td>
<td>0.45s</td>
<td>38s</td>
</tr>
</tbody>
</table>

R: ±1m / C++ (normal vectors): 57MB 0.68s 9.8GB 29m
C++ (sparse vectors): 34MB 0.45s 6.6GB 38s
Smart pointers, Pimpl, Factory pattern

file.h

Public Interface
+class impl;
+std::shared_ptr<impl> m_impl;
+operation1()

Derived Public Interface

file-imp.h/file.cpp

Private implementation
+private data
+operation1()

implA
+operation1()

implB
+operation1()
Smart pointers, Pimpl, Factory pattern

*Pointer to implementation or Private implementation*
Smart pointers, Pimpl, Factory pattern

*Pointer to implementation or Private implementation*

**PROS**

- Separate interface from implementation
- Decrease recompilation cycles
- Binary compatibility of shared libraries
Smart pointers, Pimpl, Factory pattern

*Pointer to implementation or Private implementation*

**PROS**
- Separate interface from implementation
- Decrease recompilation cycles
- Binary compatibility of shared libraries

**CONS**
- Increase in memory usage
- Increase in maintenance effort
- Performance loss
- Doesn’t work well with templates
Smart pointers, Pimpl, Factory pattern

- std::unique_ptr or std::shared_ptr?
- *Mutable* or *non mutable* objects?
- Access to the objects, how often?
- Multiple inheritance, virtual inheritance (diamond problem)?
- Template member functions, template classes?
- Objects in a coherent state!
Smart pointers, Pimpl, Factory pattern: Inheritance

API

Pimpl
+class impl
+std::shared_ptr<impl> m_ptr
&_get_ptr<I>()
&_valid_ptr()

Test
+class impl
+error()
+results()

CountField
+class impl

BetaBinomial
+class impl
+model()

Implementation

Pimpl::impl
#make_pimpl(Arg&& ...arg)

Test::impl
+m_error
+error()
#add_error()
+results(): =0

BetaBinomial::impl
+m_model
+build(...) 
+results()
+model()
#compute()

CountField::impl

impl_no_visits
#impl_no_visits

impl_with_visits
#impl_with_visits

COUNTFIELD_INTERNAL
impl_no_visits
impl_with_visits
Smart pointers, Pimpl, Factory pattern: Diamond problem

API

- Pimpl
  +class impl
- Test
  +class impl
- CNR_Base_ByCenter
  +class impl
- CNR_Base_BySubjid
  +class impl
- CNR_Mean_ByCenter
  +class impl
- CNR_Mean_BySubjid
  +class impl
- CNR_Sd_ByCenter
  +class impl
- CNR_ByCenter
  +class impl
- CNR_BySubjid
  +class impl

Implementation

- Pimpl::impl
- Test::impl
- CNR_Base_ByCenter::impl
- CNR_Base_BySubjid::impl
- CNR_Mean_ByCenter::impl
- CNR_Mean_BySubjid::impl
- CNR_Sd_ByCenter::impl
- CNR_ByCenter::impl
- CNR_BySubjid::impl
Smart pointers, Pimpl, Factory pattern: Template members

api.hpp

class MyPublic : public Pimpl {  
public:
    class impl;
    MyPublic(...);
    template<typename Tp> Tp as() const;
};

api.cpp

MyPublic::MyPublic(...) : Pimpl(impl::build(...)) {}  
template<>  
double MyPublic::as() const { return _valid_ptr() ? _get_ptr<impl>()->asNumber() : NaN(); }  
template<>  
std::string MyPublic::as() const { return _valid_ptr() ? _get_ptr<impl>()->asString() : std::string(); }  

api-impl.hpp

class MyPublic::impl : public Pimpl::impl {  
public:
    static PTR build(...) { return make_pimpl<impl>(...); }  
    double asNumber() const { return ...; }  
    std::string asString() const { return ...; }  
protected:
    ~impl();  
};
typedef std::vector<Test> TESTS;

// Create the tests
TESTS tests;
tests.push_back(BetaBinomial(...));
tests.push_back(CountField(...));
tests.push_back(CNR_ByCenter(...));
...

// Export the results
json_ostream os(...);
print_results(os, tests);

void print_results(json_ostream& os, const TESTS& tests)
{
    for(const auto& test: tests) {
        ...
    }
}
Pimpl: use
Fail-fast/Fail-safe idioms

It doesn't work...... why?

It works....... why?

$ 9GAG is your best source of fun.
Fail-fast/Fail-safe idioms

► Check constraints on input/output

```cpp
double foo(const std::vector<size_t>& l, const std::vector<double>& x, const std::vector<bool>& w)
{
    CP_ASSERT(l.size() == x.size());
    CP_ASSERT(l.size() == w.size());
    // Rest of the code
}
```

Fitting of statistical models might fail:

```cpp
try {
    fit = vglm("cbind(a,b)~1",
                Named("family", family),
                Named("data", dateframe),
                Named("control", control(Named("criterion", "coef"),
                                           Named("stepsize", 0.5))));
} catch(std::exception& e) {
    // Retry with other parameters
}
```

- Check constraints on input/output
- Fitting of statistical models might fail
- Handle errors gracefully
Fail-fast/Fail-safe idioms

- Check constraints on input/output

```cpp
double foo(const std::vector<size_t>& l, const std::vector<double>& x, const std::vector<bool>& w)
{
    CP_ASSERT(l.size() == x.size());
    CP_ASSERT(l.size() == w.size());
    // Rest of the code
}
```

- Fitting of statistical models might fail

```cpp
try {
    fit = vglm("cbind(a,b)-1",
               Named("family", family),
               Named("data", dataframe),
               Named("control", control(Named("criterion", "coef"),
                                           Named("stepsize", 0.5))));
} catch(std::exception& e) {
    // Retry with other parameters
}
```

- Propagate the error message
- Rethrow the exception
- Store the exception as an error message inside the object
  ```cpp```
```
Fail-fast/Fail-safe idioms

- Check constraints on input/output

```cpp
double foo(const std::vector<size_t>& l, const std::vector<double>& x, const std::vector<bool>& w) {
  CP_ASSERT(l.size() == x.size());
  CP_ASSERT(l.size() == w.size());
  // Rest of the code
}
```

- Fitting of statistical models might fails

```cpp
try {
  fit = vglm("cbind(a,b)-1",
             Named("family", family),
             Named("data", dataframe),
             Named("control", control(Named("criterion", "coef"),
                                       Named("stepsize", 0.5))));
} catch(std::exception& e) {
  // Retry with other parameters
}
```

- Propagate the error message
  - Rethrow the exception
  - Store the exception as an error message inside the object
  - ...
Numerical instabilities

Credit xkcd
Numerical instabilities

Test on standard deviations

- P values computed from the integration of two functions:

\[ f_1(x) = \text{pchisq}(s/x^2; N, \text{left.tail}) \times \text{dgamma}(x; \text{scale, shape}) \]

\[ f_2(x) = \text{pchisq}(s/x^2; N, \text{right.tail}) \times \text{dgamma}(x; \text{scale, shape}) \]
Numerical instabilities

calcPsd: test on standard deviations

- $f_1(x)$ is unstable in case $\text{shape} < 1$
Numerical instabilities

calcPsd: test on standard deviations

- $f_1(x)$ is unstable in case $\text{shape} < 1$
- $f_1(x)$ can be rewritten by using the integration by parts theorem

\[ \int_0^a uv = [uv]_0^a - \int_0^a vdu \]

\[ f_1'(x) = \frac{2s}{x^3} \times dchisq(s/x^2; N) \times pgamma(x; \text{scale}, \text{shape}, \text{left.tail}) \]

\[ f_1(x); f_1'(x) \]
Minimizing numerical errors

- Sample variance – Standard formula

\[ s^2_N = \frac{1}{N - 1} \sum_{k=1}^{N} (x_k - \bar{x})^2 \]
Minimizing numerical errors

- Sample variance – Standard formula

\[ s^2_N = \frac{1}{N - 1} \sum_{k=1}^{N} (x_k - \bar{x})^2 \]

- Can be implemented as a 2 pass algorithm, first the mean \( \bar{x} \), and the variance \( s^2 \) afterwards.

\[ \text{MINIMIZING NUMERICAL ERRORS} \]

\[ s^2_N = \frac{1}{N - 1} \sum_{k=1}^{N} (x_k - \bar{x})^2 \]

- Can be implemented as a 2 pass algorithm, first the mean \( \bar{x} \), and the variance \( s^2 \) afterwards.

\[ \text{MINIMIZING NUMERICAL ERRORS} \]

\[ s^2_N = \frac{1}{N - 1} \sum_{k=1}^{N} (x_k - \bar{x})^2 \]

- Can be implemented as a 2 pass algorithm, first the mean \( \bar{x} \), and the variance \( s^2 \) afterwards.
Minimizing numerical errors

- Sample variance – Standard formula

\[ s^2_N = \frac{1}{N-1} \sum_{k=1}^{N} (x_k - \bar{x})^2 \]

- Can be implemented as a 2 pass algorithm, first the mean \( \bar{x} \), and the variance \( s^2 \) afterwards.
  
  **BUT** the number of items \( N \) can be huge
Minimizing numerical errors

- Sample variance – Standard formula

\[ s^2_N = \frac{1}{N - 1} \sum_{k=1}^{N} (x_k - \bar{x})^2 \]

- Can be implemented as a 2 pass algorithm, first the mean \( \bar{x} \), and the variance \( s^2 \) afterwards.
  - **BUT** the number of items \( N \) can be huge
- Have a one pass algorithm
Minimizing numerical errors

- Sample variance – Standard formula

\[ s^2_N = \frac{1}{N - 1} \sum_{k=1}^{N} (x_k - \bar{x})^2 \]

- Can be implemented as a 2 pass algorithm, first the mean \( \bar{x} \), and the variance \( s^2 \) afterwards.
  - BUT the number of items \( N \) can be huge
- Have a one pass algorithm
- Compute the variance for increasing \( N \) to observe convergence.
Minimizing numerical errors

- Sample variance – One pass algorithm: *Sum of squares method*

\[ s_N^2 = \frac{1}{N(N - 1)} \left( N \sum_{k=1}^{N} x_k^2 - \left( \sum_{k=1}^{N} x_k \right)^2 \right) \]

One pass algorithm but the formula is unstable:

- *float precision:* for \{10000f, 10001f, 10002f\}, the result is \(-1.0666667e+01\) instead of 1.

- *double precision:* for \{100000000, 100000001, 100000002\}, the result is 0 instead of 1.
Minimizing numerical errors

- Sample variance – Iterative algorithm: *Welford’s recursion method*

\[
M_k = M_{k-1} + \frac{x_k - M_{k-1}}{k}
\]
\[
S_k = S_{k-1} + (x_k - M_{k-1})(x_k - M_k),
\]

with \( M_0 = 0 \) and \( S_0 = 0 \), and then

\[
s_N^2 = \frac{S_N}{N - 1},
\]

This stable algorithm with can be easily turned into an accumulator.
Accumulators

Start

Loop

Accumulator

End
Accumulators

- Separation of the operations on the elements from the iteration leads to smaller testable code.

- Statistical tests involve operations (aggregation, sum, average, variance, ...) on one or more variables based on one or more several key variables. E.g.: Preprocess involves taking the mean by visits or the sum by patients, the count of non missing values per center, ...
Accumulators as a OO pattern

Mean of the elements of a vector

▶ without accumulator

```cpp
1   double sum{0};
2   for(const auto x: myvector) sum += x;
3   const double mean = sum / myvector.size();
```
Accumulators as a OO pattern

Mean of the elements of a vector

► without accumulator

```cpp
1 double sum{0};
2 for(const auto x: myvector) sum += x;
3 const double mean = sum / myvector.size();
```

► with accumulator

```cpp
1 using namespace boost::accumulators;
2 accumulator_set<
3 double, stats<tag::mean> > acc;
4 for(const auto x: myvector) acc(x);
5 mean(acc);
```
Accumulator implementation

Sample variance – Welford’s recursion method

\[
M_k = M_{k-1} + \frac{x_k - M_{k-1}}{k}
\]

\[
S_k = S_{k-1} + (x_k - M_{k-1})(x_k - M_k)
\]

\[
s_k^2 = \frac{S_k}{k - 1},
\]

```cpp
1 template<typename T> class Variance {
2     size_t k; /**< Number of elements */
3     T m; /**< 0th order moment, i.e. average */
4     T s; /**< 1st order moment */
5 public:
6     Variance() : k(0), m(0), s(0) {}
7     void operator()(const T x) {
8         if(std::isnan(x)) return;
9         ++k;
10         const T pm(m);
11         m += (x-pm) * (ONE/k);
12         s += (x-pm) * (x-m);
13     }
14     T average() const noexcept { return m; }
15     T s2() const noexcept { return k > 1 ? s / (k-1) : ZERO; }
16     }
```
std::algorithms, boost, GSL, BLAS, LAPACK, ...  

- `<algorithm>`
- `boost`
  - Statistics, ...
  - Logging facilities
  - System (command line arguments, ...)
  - Thread, MPI, Serialization
  - ...
- `GNU Scientific Library`
  - Optimization, minimization, ...
- `BLAS` and `LAPACK`
  - Operations on matrices
- `Numerical Recipes`
  - Lots of algorithms
Implementing Beta-Binomial distribution with boost

In probability theory and statistics, the beta-binomial distribution is a family of discrete probability distributions on a finite support of non-negative integers arising when the probability of success in each of a fixed or known number of Bernoulli trials is either unknown or random.
Implementing Beta-Binomial distribution with boost

\[ f(k; n, \alpha, \beta) = \binom{n}{k} \frac{B(k + \alpha, n - k + \beta)}{B(\alpha, \beta)} \]

- \( k \) and \( n \) are positive integers with \( k \leq n \)
- \( \alpha \) and \( \beta \) are strictly positive numbers
- Binomial coefficient
  \[ \binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{\Gamma(n+1)}{\Gamma(k+1)\Gamma(n-k+1)} \]
- Beta function
  \[ B(x, y) = \frac{\Gamma(x) + \Gamma(y)}{\Gamma(x+y)} \]
Implementing Beta-Binomial distribution with boost

\[ f(k; n, \alpha, \beta) = \binom{n}{k} \frac{B(k + \alpha, n - k + \beta)}{B(\alpha, \beta)} \]

- Numerically fine as long as \( \alpha \) and \( \beta \) are small
- When \( \alpha \) and \( \beta \) are not small, \( B(\alpha, \beta) \) tends toward zero.
Implementing Beta-Binomial distribution with boost

\[ f(k; n, \alpha, \beta) = \binom{n}{k} \frac{B(k + \alpha, n - k + \beta)}{B(\alpha, \beta)} \]

- Numerically fine as long as \( \alpha \) and \( \beta \) are small
- When \( \alpha \) and \( \beta \) are not small, \( B(\alpha, \beta) \) tends toward zero.

**Trick:** Do the calculation in the log scale:

\[ f(k; n, \alpha, \beta) = \exp \left( \log \binom{n}{k} + \log B(k + \alpha, n - k + \beta) - \log B(\alpha, \beta) \right) \]

\[ \log \binom{n}{k} = \text{l_binomial_coefficient}(n, k) \]

\[ = \text{lgamma}(n + 1) - \text{lgamma}(k + 1) - \text{lgamma}(-k + n + 1) \]

\[ \log B(x, y) = \text{lbeta}(x, y) \]

\[ = \text{lgamma}(x) + \text{lgamma}(y) - \text{lgamma}(x + y) \]
Minimize with GSL

- GSL is a C library
- Use wrapper classes
- Pointer to the minimizer created/owned by GSL
- Pointer to the function definition struct
Minimizers – Example

```
enum class M_TYPE {
    NO_GRADIENT,
    ...
};

template<M_TYPE> struct M_API; /**< Template for the C API */
template<M_TYPE> class M_FCT; /**< Template for defining the function to minimize */
template<M_TYPE> class IMinimizer; /**< Template for the minimizer */
```
Minimizers – Example

```cpp
/// Specialized template defining the function to minimize (no gradient)
template<>
class M_FCT<M_TYPE::NO GRADIENT> {
public:
    friend class IMinimizer<M_TYPE::NO GRADIENT>;

    /// Virtual base class for the implementation of the function to minimize
class Base : public boost::noncopyable {
public:
    virtual ~Base() {}  
    virtual double evaluate(const double _x) = 0; 
    
    typedef std::unique_ptr<Base> PTR; //** Type of the pointer to the instance of the function to minimize */
    typedef gsl_function DEF; //** Type for the definition */

    M_FCT(PTR _fct, const NUMBER _minimum, const NUMBER _lower, const NUMBER _upper) : ...

    double evaluate(const double _x) { return m_fct->evaluate(_x); }  
    double get_lowest_bound() const { return m_f_lower < m_f_upper ? m_f_lower : m_f_upper; }  
    double get_lowest_f_bound() const { return std::min(m_f_lower, m_f_upper); }

private:
    ...
};
```
Minimizers – Example

```cpp
template<M_TYPE _Type> class IMinimizer : public boost::noncopyable {
public:
  typedef M_FCT<_Type> FCT;

  explicit IMinimizer(const std::string& _type, FCT& _fct, const double _epsabs, const double _epsrel) { ... }
  ~IMinimizer() { ... }

  bool iterate(const size_t _maxiter) {
    if(m_can_minimize) {
      for(size_t iter = 0; iter<_maxiter && next(); ++iter) {
        if(converged()) return true;
      }
    }
    return false;
  }

  std::string name() const {
    return m_ptr ? M_API<_Type>::name(m_ptr) : std::string(); }
  bool next() {
    return m_ptr && m_can_minimize ? M_API<_Type>::iterate(m_ptr) == 0 : false; }
  bool converged() const {
    return m_ptr && m_can_minimize ? M_API<_Type>::converged(m_ptr, m_epsabs, m_epsrel) : false; }
  double x() const {
    return m_ptr ? (m_can_minimize ? M_API<_Type>::x_minimum(m_ptr) : m_fct.get_lowest_bound()) : 0; }
  double y() const {
    return m_ptr ? (m_can_minimize ? M_API<_Type>::f_minimum(m_ptr) : m_fct.get_lowest_f_bound()) : 0; }
private:
  ... 
}

typedef IMinimizer<M_TYPE::NO_GRADIENT> MinimizerNoGradient;
```
Minimizers – Example

```cpp
template<M_TYPE> struct M_API; // Generic template holding the API

template<>
struct M_API<M_TYPE::NO_GRADIENT> {

typedef gsl_min_fminimizer* PTR; /**< Type of the pointer to the minimizer */
typedef gsl_function* DEF; /**< Type of the pointer to the definition */

static PTR alloc(const STRING& _type) { return gsl_min_fminimizer_alloc(type(_type)); }
static void free(PTR _p) { gsl_min_fminimizer_free(_p); }

static const gsl_min_fminimizer_type* type(const std::string& _type);
static std::string name(PTR _p) { return gsl_min_fminimizer_name(_p); }

static bool set(PTR _p, DEF _fct, const double _minimum, const double _lower, const double _upper)
{ return gsl_min_fminimizer_set(_p, _fct, _minimum, _lower, _upper) != GSL_EINVAL; }
static bool set(PTR _p, DEF _fct, const double _minimum, const double _fminimum, const double _lower, const double _flower, const double _upper, const double _fupper)
{ return gsl_min_fminimizer_set_with_values(_p, _fct, _minimum, _fminimum, _lower, _flower, _upper, _fupper) != GSL_EINVAL; }

static int iterate(PTR _p) { return gsl_min_fminimizer_iterate(_p); }
static bool converged(PTR _p, const double _epsabs, const double _epsrel)
{ return gsl_min_test_interval(x_lower(_p), x_upper(_p), _epsabs, _epsrel) == GSL_SUCCESS; }

static double x_minimum(PTR _p) { return gsl_min_fminimizer_x_minimum(_p); }
static double x_upper(PTR _p) { return gsl_min_fminimizer_x_upper(_p); }
static double x_lower(PTR _p) { return gsl_min_fminimizer_x_lower(_p); }
static double f_minimum(PTR _p) { return gsl_min_fminimizer_f_minimum(_p); }
static double f_upper(PTR _p) { return gsl_min_fminimizer_f_upper(_p); }
static double f_lower(PTR _p) { return gsl_min_fminimizer_f_lower(_p); }
};
```
class MyFctNoGradient : public gsl::MinimizerNoGradient::FCT::Base {
public:
    MyFctNoGradient(...) { ... }

double evaluate(const double _x) override { ... }
};

double minimize_my_fct(...) {
    gsl::MinimizerNoGradient::FCT f(new MyFctNoGradient(...), .5*(low+hi), low, hi);
    gsl::MinimizerNoGradient minimizer("Brent", f, 0.1, 0.1);
    minimizer.iterate(10);
    return minimizer.x();
}

Conclusion

- Fast production code
- No task is impossible
- Seek expertise
- Testing: Don’t trust your code
- Have fun and keep learning
Questions, Remarks?

Thank you for your attention!