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

Olivia Quinet
olivia.quinet @cluepoints.com

Meeting C++
BeCPP July 2019

CluePoints
Intelligent Statistical Monitoring


MAN, I SUCK AT THIS GAME.
CAN YOU GIVE ME
A FEW POINTERS?
0x3A28213A
0x6339392C,
0x7363682E.
I HATE YOU.

Credit xkcd

CluePoints

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

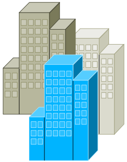
Why?

- Is it working?
- Is it safe?
- Scope, dosage, . . .
- Approuval from FDA, EMA, . . .

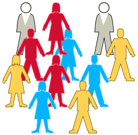
Clinical Trials

How?

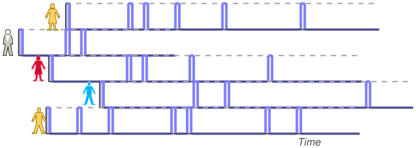
- Clinical protocol
- Study conducted at sites
- Patients are enrolled
- Data is collected: demographics, medical history, vital signs, adverse events, labs, patient journals, ...
- Data is verified and analyzed



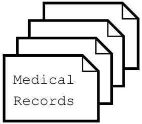
Sites



Patients



Visits



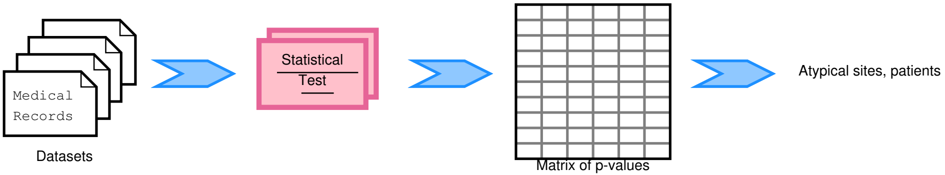
Datasets

Clinical Trials

$$$?

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

Statistical tests



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

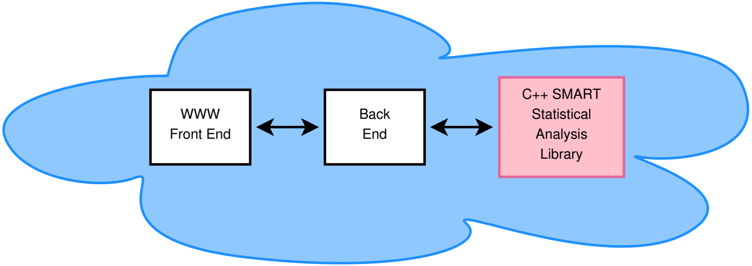
- Initially developed in the R language by the R&D team
- Very good for research purposes
- Not so much for production

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 programing 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)

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

Some examples (I)

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

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)

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

center	xyz
ctr01	
ctr02	1
ctr01	2
ctr03	3
ctr05	4
ctr02	
ctr02	5
ctr02	
...	

ctr	npat	y
ctr01	1	2
ctr02	2	7
ctr03	5	10
...		

Some examples (II)

```
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));
sigma = sqrt(Sn*vc[3]^2 + Sn2*vc[2]^2 + Sn^2*vc[1]^2)/Sn;
p = pnorm(v, sd=sigma)
```

Some examples (II)

```
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));
sigma = sqrt(Sn*vc[3]^2 + Sn2*vc[2]^2 + Sn^2*vc[1]^2)/Sn;
p = pnorm(v, sd=sigma)
```

1. Apply an offet 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

Some examples (II)

```
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));
sigma = sqrt(Sn*vc[3]^2 + Sn2*vc[2]^2 + Sn^2*vc[1]^2)/Sn;
p = pnorm(v, sd=sigma)
```

center	subjid	x
ctr01	s01001	
ctr02	s02001	1
ctr01	s01001	2
ctr03	s03001	3
ctr05	s05001	4
ctr02	s02002	
ctr02	s02001	5
ctr02	s02001	
...		

ctr	Sn	Sn2	sigma	p
ctr01	1	1	0.37	0.21
ctr02	3	5	0.25	0.55
ctr03	5	5	0.19	0.06
...				

Some examples (III)

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

Some examples (III)

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

1. Create a boolean vector indicating if a subjid is duplicated or not
2. Get the values of the column "field"
3. Do some wierd 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

Some examples (III)

```r
dd = duplicated(d$subjid);
v = d[[field]]
w = dd & c(FALSE, v[1:(length(v)-1)]==1);
x10 = rowsum(1-v[w], ctr[w]);
N10 = unclass(table(ctr[w]));
x10Max = rowsum(as.integer(!c(TRUE, w[1:(length(w)-1)])[w]), ctr[w]);
```

center	subjid	visit	x
ctr01	s01001	v01	1
ctr01	s01001	v02	0
ctr01	s01001	v03	1
ctr01	s01001	v04	0
ctr01	s01002	v01	1
ctr01	s01002	v02	1
ctr02	s02001	v03	0
ctr02	s02002	v01	1
ctr02	s02002	v02	1
ctr02	s02002	v03	0
ctr03	s03001	v01	1
ctr03	s03001	v02	0
ctr03	s03001	v03	1
...			

ctr	X10	N10	x10max
ctr01	2	3	3
ctr02	1	2	1
ctr03	1	1	1
...			

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 $\longrightarrow$ ± 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 $\longrightarrow$ ± 30 lines in C++
- Hardest: changing the data structure
  - PRO Less ressource/faster code
  - CON C++ and R codes are even more different

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 $\longrightarrow$ ± 30 lines in C++
- Hardest: changing the data structure
  - PRO Less ressource/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 $\longrightarrow$ $\pm$30 lines in C++
- Hardest: changing the data structure
  - PRO Less ressource/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 $\longrightarrow$ $\pm$30 lines in C++
- Hardest: changing the data structure
  - PRO Less ressource/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, . . .)

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

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: $f(x, y, w)$

X	Y	w
42.5	100	true
...	...	true
...	...	false
...	...	true
...	...	false
...	...	true
...	...	true
...	...	false
...	...	...

Measure!!! – an example

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

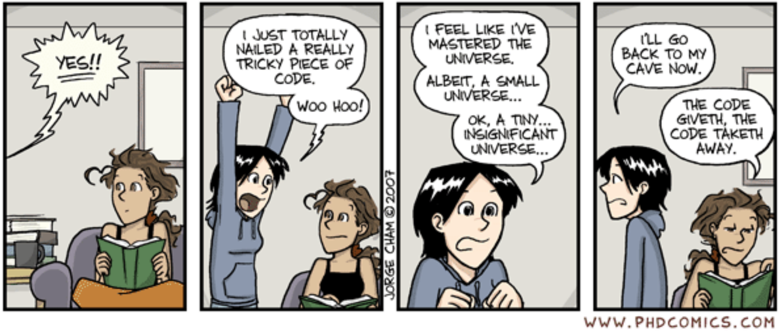
- Modify the algorithm to take into account only the filtered vectors'elements: *filter algo*
- Create pseudo vectors with the filtered elements: *filter vector*
- Create new vectors with the filtered elements: *copy vector*

Measure!!! – an example

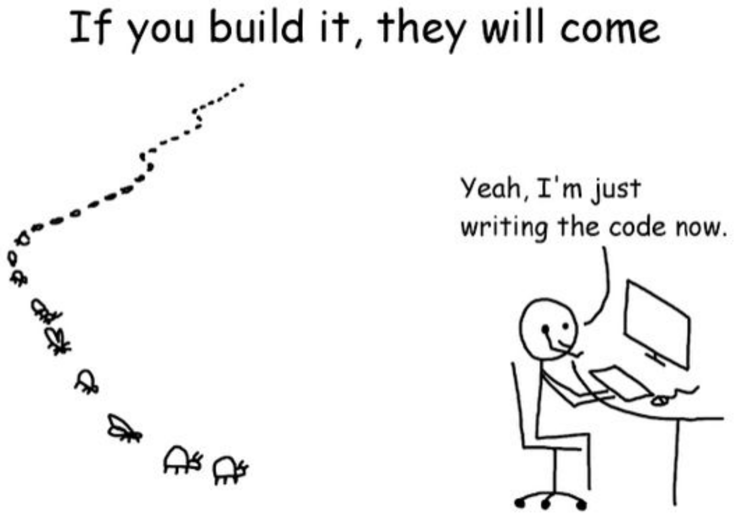
Context: Originally, an algorithm has to be applied on vectors: $f(x, y)$
Then only on some filtered elements: $f(x, y, w)$

- 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*

Option	Timing (s)			
	$N = 10^2$	$N = 10^4$	$N = 10^6$	$N = 10^8$
filter algo	0.0003	0.008	0.9	100
filter vector	0.0003	0.006	0.8	98
copy vector	0.0006	0.015	4.6	/


YES!!
I JUST TOTALLY NAILED A REALLY TRICKY PIECE OF CODE.
WOO HOO!
JORGE CHAM © 2007
I FEEL LIKE I'VE MASTERED THE UNIVERSE.
ALBEIT, A SMALL UNIVERSE...
OK, A TINY... INSIGNIFICANT UNIVERSE...
I'LL GO BACK TO MY CAVE NOW.
THE CODE GIVETH, THE CODE TAKETH AWAY.
WWW.PHDCOMICS.COM

Software architecture & Data structure



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

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

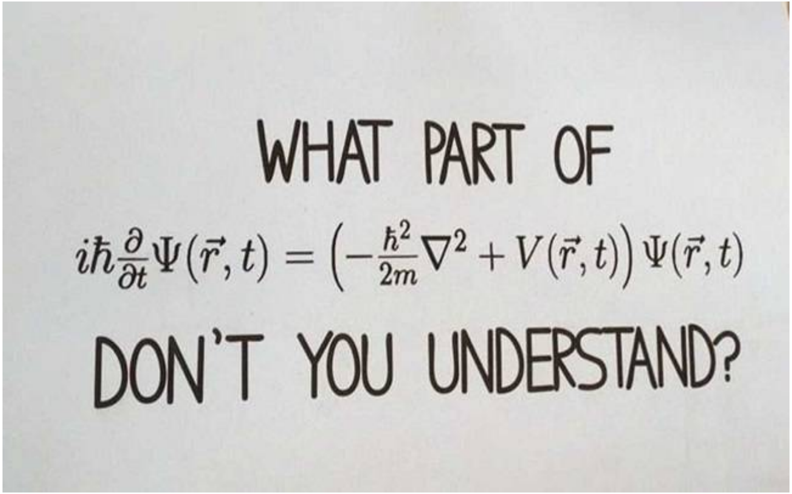
For this project

- 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* $\longleftrightarrow$ *high cohesion*
- *Design review*

Levels of abstraction – mathematical formula



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

$$s_N^2 = \underbrace{\frac{1}{N-1}}_{\text{Fraction}} \times \underbrace{\underbrace{\sum_{k=1}^{N} \underbrace{(x_k - \underbrace{\bar{x}}_{\text{Mean}})^2}_{\text{Square of difference}}}_{\text{Sum}}}_{\text{Multiplication}}$$

Abstraction levels – Example

```
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

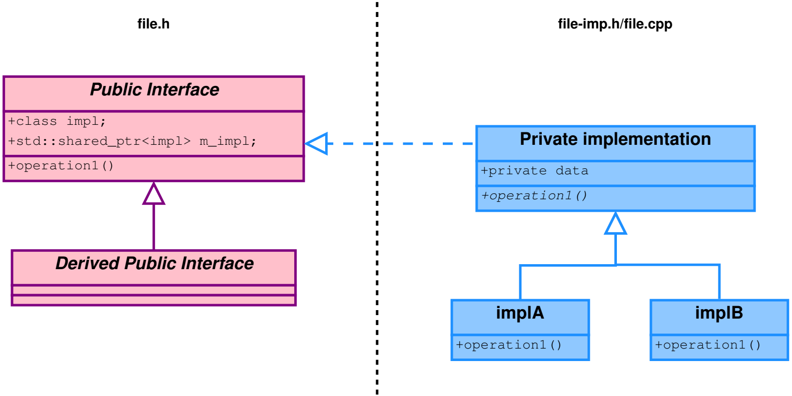
- N numerical vectors of the same length M
  Typical cases: $N = 1000 - 40000$ and $M = 20 - 20000$
- $N \times (N-1)/2$ scalar products

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

- Sparse vectors!!!
- Performance results

	$N = 841$ $M = 1060$ Memory	Timing	$N = 35613$ $M = 14304$ Memory	Timing
R		$\pm$1m		/
C++ (normal vectors)	57MB	0.68s	9.8GB	29m
C++ (sparse vectors)	34MB	0.45s	6.6GB	38s

Smart pointers, Pimpl, Factory pattern



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

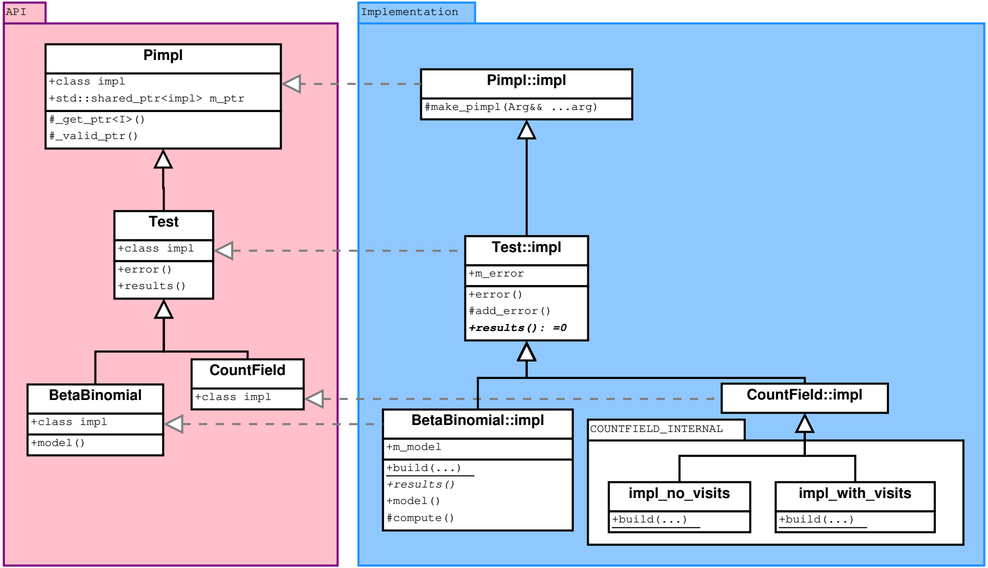
CONS

- Increase in memory usage
- Increase in maintenance effort
- Performance loss
- Doesn't work well with templates

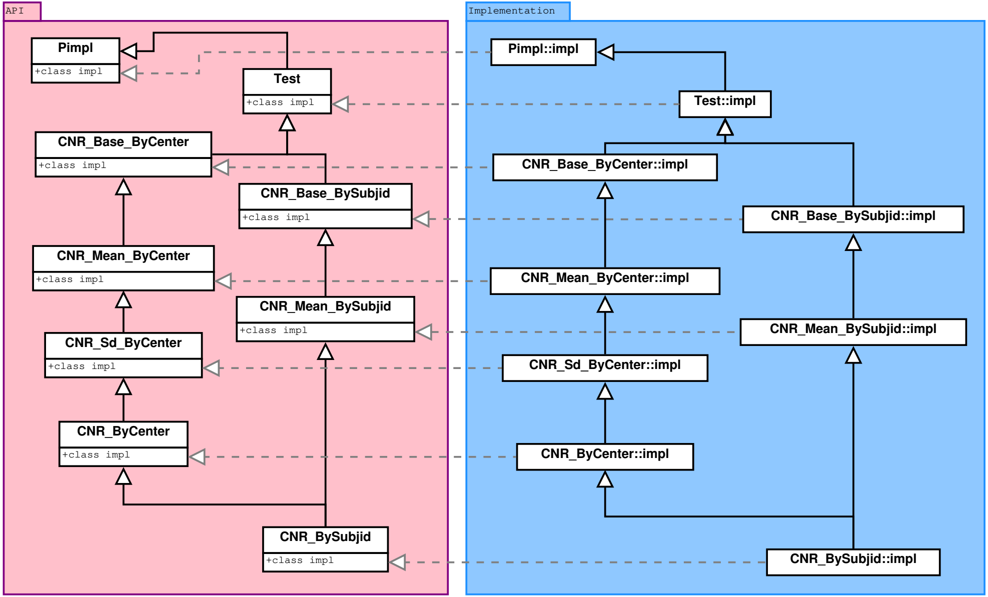
Smart pointers, Pimpl, Factory pattern

- `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



Smart pointers, Pimpl, Factory pattern: Diamond problem



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(); }
```

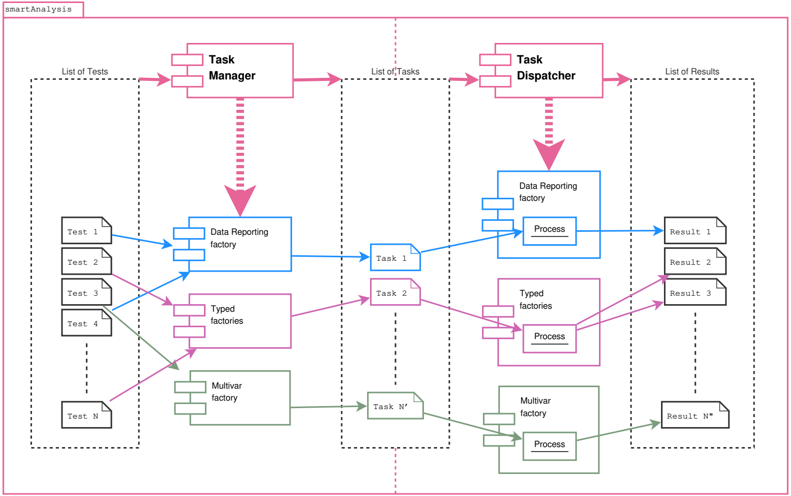
api-impl.hpp

```
class MyPublic::impl : public Pimpl::impl {
public:
  static PTR build(...) { return make_pimpl<impl>(...); }

  double asNumber() const { return ...; }
  std::string asString() const { return ...; }

protected:
```

Pimpl: use

```
typedef std::vector<Test> TESTS;

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



smartAnalysis
List of Tests
Task Manager
List of Tasks
Task Dispatcher
List of Results
Test 1
Test 2
Test 3
Test 4
Test N
Data Reporting factory
Typed factories
Multivar factory
Task 1
Task 2
Task N'
Data Reporting factory
Process
Typed factories
Process
Multivar factory
Process
Result 1
Result 2
Result 3
Result N"

Fail-fast/Fail-safe idioms

9GAG is your best source of fun.

Fail-fast/Fail-safe idioms

- Check constraints on input/output

```
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
}
```

Fail-fast/Fail-safe idioms

- Check constraints on input/output

```
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

```
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
}
```

Fail-fast/Fail-safe idioms

- Check constraints on input/output

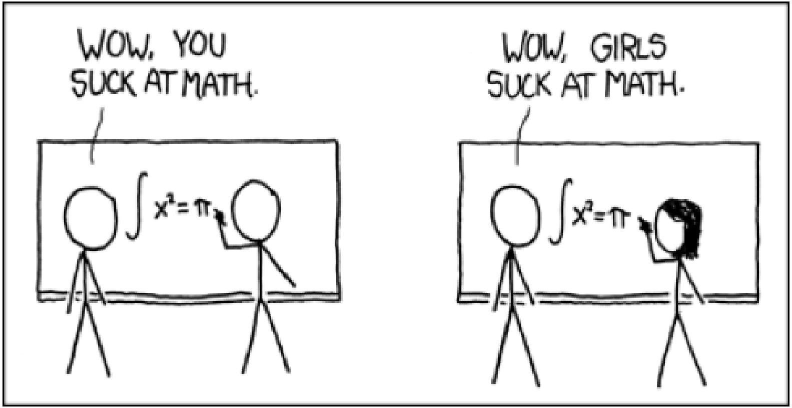
```
double foo(const std::vector<size_t>& l, const std::vector<double>& x, const std::vector<bool>& w)
{
  CP_ASSERT(l.size() == x.size());
  CP_ASSERT(l.size() == w.size());
  // Rest of the code
}
```

- Fitting of statistical models might fails

```
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
}
```

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

Numerical instabilities



Credit xkcd

Numerical instabilities

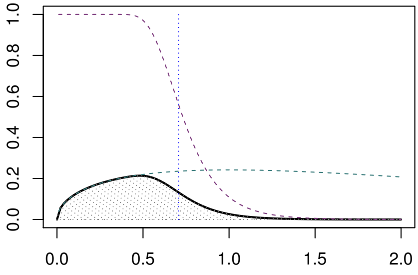
Test on standard deviations

- P values computed from the integration of two functions:

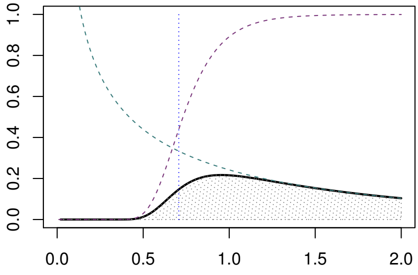
$$f_1(x) = pchisq(s/x^2; N, left.tail) \times dgamma(x; scale, shape)$$

$$f_2(x) = pchisq(s/x^2; N, right.tail) \times dgamma(x; scale, shape)$$

$f_1(x)$



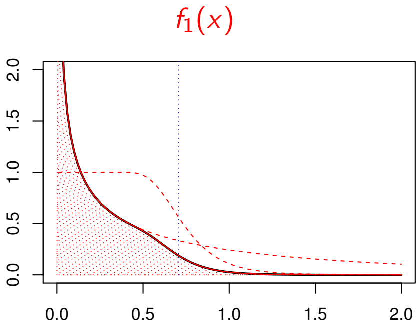
$f_2(x)$



Numerical instabilities

calcPsd: test on standard deviations

- $f_1(x)$ is unstable in case *shape* < 1



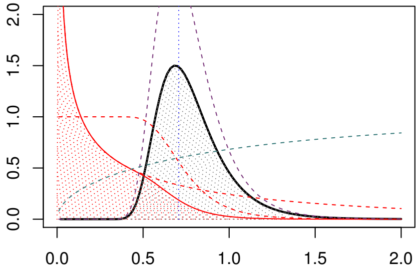
Numerical instabilities

calcPsd: test on standard deviations

- $f_1(x)$ is unstable in case $shape < 1$
- $f_1(x)$ can be rewritten by using the integration by parts theorem $\int_0^a udv = [uv]_0^a - \int_0^a vdu$

$$f_{1'}(x) = \frac{2s}{x^3} \times dchisq(s/x^2; N) \times pgamma(x; scale, shape, left.tail)$$

$f_1(x)$; $f_{1'}(x)$



Minimizing numerical errors

- Sample variance – Standard formula

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

Minimizing numerical errors

- Sample variance – Standard formula

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

  - 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

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

  - 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

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

  - 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

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

  - 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 $-1.0666667\mathrm{e}{+}01$ 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*

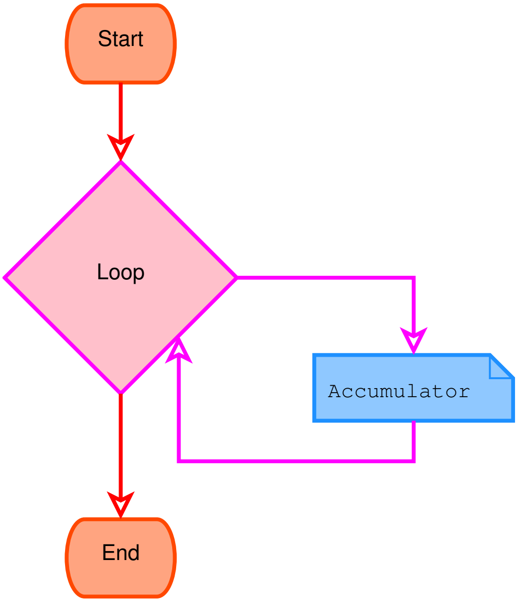
$$\begin{aligned} M_k &= M_{k-1} + \frac{x_k - M_{k-1}}{k} \\ S_k &= S_{k-1} + (x_k - M_{k-1})(x_k - M_k), \end{aligned}$$

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



Accumulators

- Separation of the operations on the elements from the iteration leads to smaller testable code.
- Statisticals tests involve operations (agregation, 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

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

Accumulators as a OO pattern

Mean of the elements of a vector

- without accumulator

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

- with accumulator

```
using namespace boost::accumulators;
accumulator_set<double, stats<tag::mean> > acc;
for(const auto x: myvector) acc(x);
mean(acc);
```

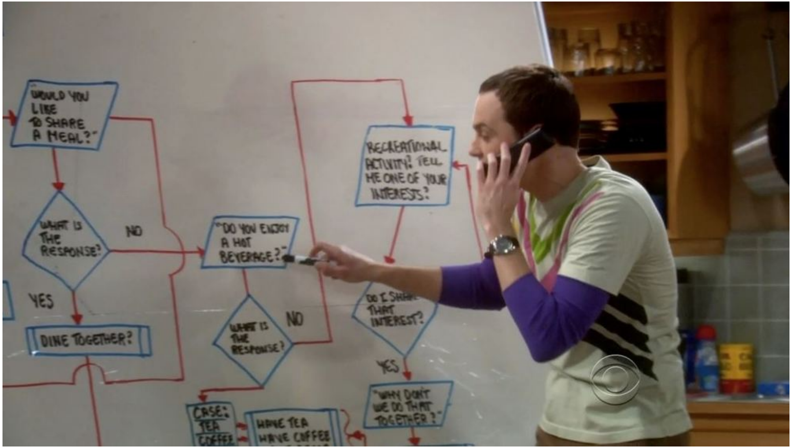
Accumulator implementation

Sample variance – Welford's recursion method

$$
\begin{aligned}
M_k &= M_{k-1} + \frac{x_k - M_{k-1}}{k} \\
S_k &= S_{k-1} + (x_k - M_{k-1})(x_k - M_k) \\
s_k^2 &= \frac{S_k}{k-1},
\end{aligned}
$$

```
template<typename T> class Variance {
  size_t k; /**< Number of elements */
  T m; /**< 0th order moment, i.e. average */
  T s; /**< 1st order moment */
public:
  Variance() : k(0), m(0), s(0) {}
  void operator()(const T x) {
    if(std::isnan(x)) return;
    ++k;
    const T pm(m);
    m += (x-pm) * (ONE/k);
    s += (x-pm) * (x-m);
  }
  T average() const noexcept { return m; }
  T s2() const noexcept { return k > 1 ? s / (k-1) : ZERO; }
};
```

std::algorithms, boost, GSL, BLAS, LAPACK, . . .



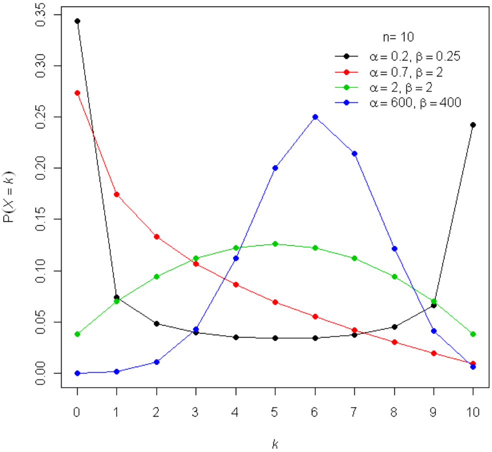
Credit Big Bang Theory

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 $k <= n$
- α and β 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 α and β are small
- When α and β 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 α and β are small
- When α and β are not small, $B(\alpha,\beta)$ tends toward zero.

Trick: Do the calculation in the log scale:

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

$$\begin{aligned}\log\binom{n}{k} &= \mathrm{l\_binomial\_coefficient}(n,k)\\ &= \mathrm{lgamma}(n+1) - \mathrm{lgamma}(k+1) - \mathrm{lgamma}(-k+n+1)\end{aligned}$$

$$\begin{aligned}\log B(x,y) &= \mathrm{lbeta}(x,y)\\ &= \mathrm{lgamma}(x) + \mathrm{lgamma}(y) - \mathrm{lgamma}(x+y)\end{aligned}$$

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

```
/// 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_lower : m_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

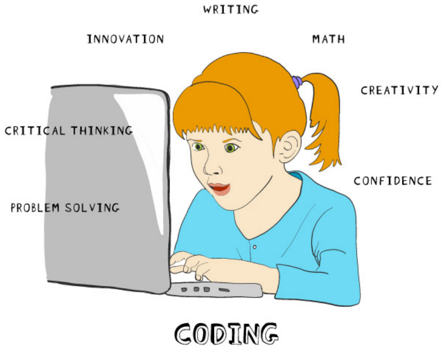
```
template<M_TYPE> struct M_API; /// Generic template holding the API

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); }
};
```

Minimizers – Example

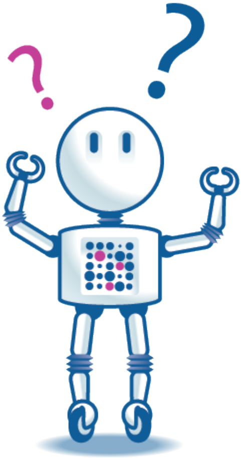
```
class MyFctNoGradient : public gsl::MinimizerNoGradient::FCT::Base {
public:
  MyFctNoGradient(...) { ... }

  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!