MSL, a library written in C++ and dedicated to microsimulation models

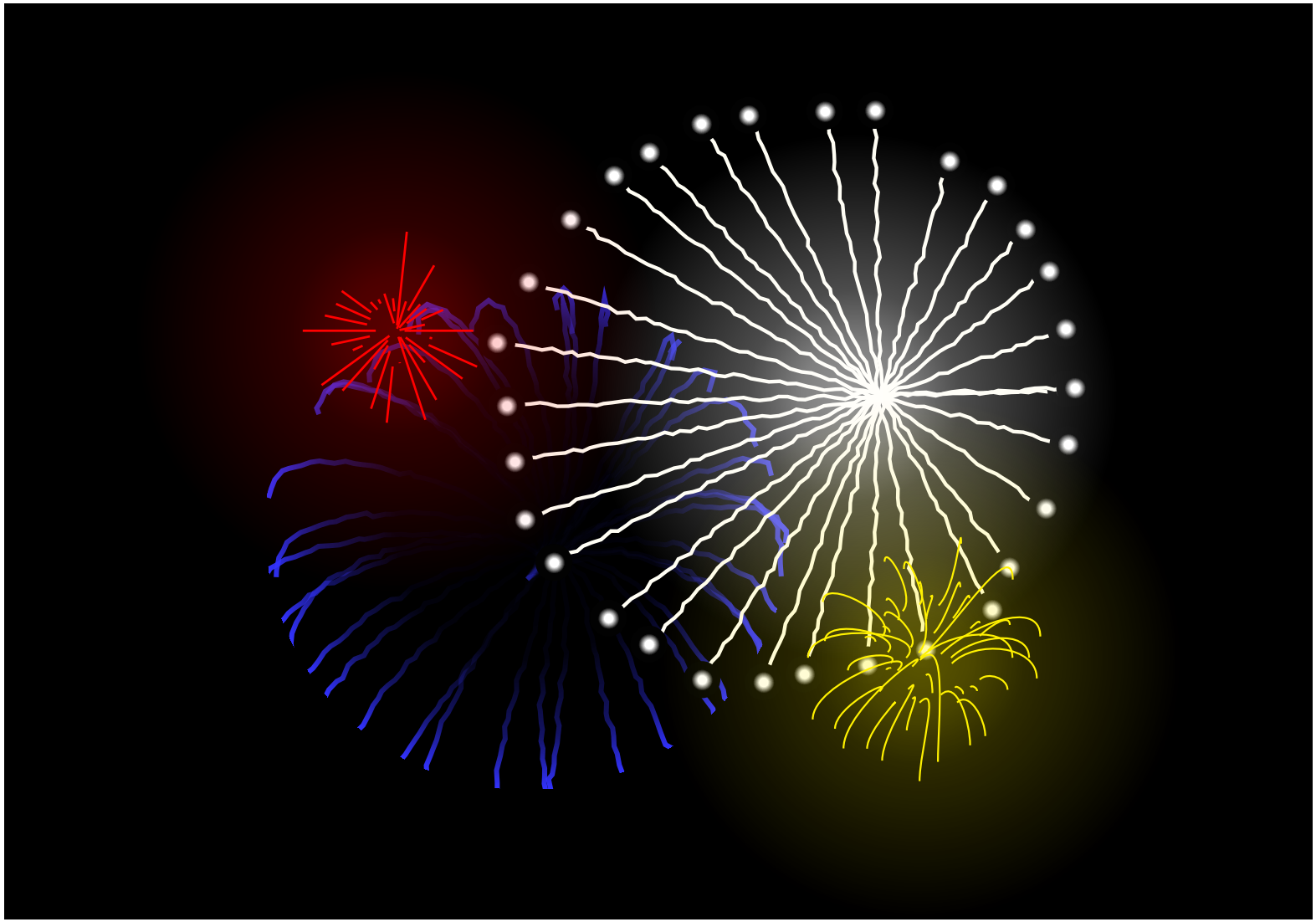
François Legendre [1]

9th World Congress of the
International Microsimulation Association
Vienna 8-10 January 2024.

[1] Univ. Paris-Est Créteil and Érudite.

Happy New Year

https://tex.stackexchange.com/questions/39485/how-can-we-display-fireworks



Outline

- Motivations
  - Modern hardware
  - Modern C++
  - Library architecture
- First session: Playing with two toy models
  - Getting the data
  - Impact assessment
  - Population projection
- Second session: Building your own model
  - Parametrize the model
  - Populate the model with survey data
  - Playing with the model

Modern hardware

Salient feature	**Recommendation**
Large main memory	Keep the entire population in memory (and refer to an individual by his address)
Various top level caches	Use highly specialised libraries Prefer simple data structures
Many cores	Promote distributed computations Provide *map and reduce* algorithms

Modern C++

- Use *STL* library, which is increasingly complete
- Use *brace initialization* and reserve the `=` sign for copy

```
int i{ 1 };
```

```
i = j;
```

- Use *reference* to access a object without copy, not *pointer*

```
auto& ref { a_big_objet };
```

```
ref.x = 0; // Uniform syntax!
```

```
auto* ptr { &a_big_objet };
```

```
ptr->x = 0;
```

- Use range-based for loop

```
for (auto& ref : a_vector) { ... }
```

- Use the `msl::enumerate(...)` function to get also the index

```
for (auto [i, ref] : msl::enumerate(a_vector)) { ... }
```

- For light processing, use *lambda* function at the call site

```
std::for_each(..., [](msl::Ref ref) {ref.x = ref.y / 2;});
```

Library architecture - I

- Main goals
  - Do not trade simplicity for safety *and* efficiency: use C++
  - Provide a framework for static models (impact assessment) and dynamic models (such as population projections)
  - Leverage, when possible, horizontal scaling
  - Give help to debug the model: check for usage of uninitialised variable, print useful error messages
  - Allow access to other libraries, like the *eigen* library
  - Core facilities only, but useful core as weighted statistics
  - Provide a clean open source code base
  - Try to supply the more efficient implementation of provided functionnalities

Library architecture - II

- Main facilities
  - Efficient data loading; elapsed time to load a $100,000$ individuals population with 50 variables: 44 ms
  - Ability to represent different subsets of the population using circular lists: families, households, those receiving benefits, etc.
  - Easy way to visit the members of a subset, circular lists mapped as C++ *forward iterators*
  - Provide basic blocks to distribute computations between cores and reduce results
  - Functions to swap "array of structs" / "struct of arrays"
  - Ability to download data as an Excel *Workbook* (transport format)

Library architecture - III

- Data layout as array of structs
  - Main data processing operations are carried out by individuals or by families.
  - Data is stored by individual, not by variable
  - As *Sas*; not as *R*
- Three steps - Fill-Build-Use
  1. Fill the "`msl-variables.cpp`" and "`msl-parameters.cpp`" files using the "`CTG`", "`FLG`", "`FLT`", "`LNK`", "`PTR`" and "`PRM`" macros
     - `CTG` : a *categorical* variable, byte-encoded variable (256 values)
     - `FLG` : a *boolean* variable (i.e. *flag*), bit-encoded variable (2 values)
     - `FLT` : a *quantitative* variable (i.e. *float*), four-byte encoded variable
     - `LNK` : a *link* variable, to represent a subset like family or household
     - `PTR` : a *pointer* variable, to get another individual like partner
     - `PRM` : a parameter whose values are stored, by period, in a Excel *Workbook*
  2. Build the specialised library for your model
  3. Use the library to developp your model

The main objects of the library

- `Individual` : the representation of an individual
- `Population` : a container for the current instances of `Individual`
- `CircIterAsForwardIter` : an adapter to transform a circular list between individuals to a STL *forward iterator*

The main free functions of the library

- `load_population` : load the population from disk
- `load_parameters` : load the parameters from a Excel *Workbook* for a period
- `transport_load_population` : load the population from a Excel *Workbook* in transport format
- `enumerate` : like the Python enumerate function
- `days_from_civil` : get a date (as a number of days since January 1, 1970) from a civil date

The `CTG` variables

Putting `CTG(std_living_10)` in the "`msl-variables.cpp`" file give you the abilities

- to use the *data member* `std_living_10` of the `Individual` *object* like that:
  `ref.std_living_10 = 8`
  or
  `if (ref.std_living_10 == 5) { ... }`
- to get tabulars, using distributed computations, where the `std_living_10` variable is the *by* variable
  `popul.sum_by(msl::var::x, msl::var::std_living_10)`

The `FLG` variables

Putting `FLG(female)` in the "`msl-variables.cpp`" file give you the abilities

- to use the *function member* `female()` of the `Individual` *object* to get the value of the boolean variable like that:
  ```
  if (ref.female()) { ... }
  ```
- to use the *function member* `set_female(boolean)` of the `Individual` *object* to set the value of the boolean like that:
  ```
  ref.set_female(true)
  ```
- to leverage distributed computations, using uniform syntax for the variable, like that:
  ```
  popul.count_if(msl::var::female)
  ```
  or
  ```
  popul.avg_if(msl::var::wages, msl::var::female)
  ```

The `FLT` variables

Putting `FLT(x)` in the "`msl-variables.cpp`" file give you the abilities

- to use the *data member* `x` of the `Individual` *object* like that:
  `ref.x = ref.y / 2;`
  or
  `if ( ref.x > 0 ) { ... }`
- to use the *function member* `x_()` of the `Individual` *object* to get the value of the variable with a check for missing value
- to use the *function member* `set_x(value)` of the `Individual` *object* to set the value of the variable
- to leverage distributed computations, using uniform syntax for the variable, like that:
  `popul.weighted_pctl(msl::var::x, msl::var:weigth)`
  or
  `popul.gini_if(msl::var::x, msl::var::female)`
  or
  `popul.fill_with_missing(msl::var::x)`

The `LNK` variables

Putting `LNK(family)` in the "`msl-variables.cpp`" file give you the abilities

- to use the *member function* `over_family()` which returns an *iterator* to visit each family member like that:
  ```
  for (msl::Ref member : ref.over_family()) { ... }
  ```
  or
  ```
  ref.over_family().sum(msl::var::wages)
  ```
- to use the `FLG` variable `family_head`, egal to `true` when the individual is the head of the family, like that:
  ```
  popul.count_if(msl::var::family_head)
  ```
- to use the *member function* `for_each_family` to start, in a distributed way, a task for each family head
- to use the *member function* `merge_family`, like that
  ```
  ref_mother.merge_family(ref_baby)
  ```

The `PTR` variables

Putting `PTR(partner)` in the "`msl-variables.cpp`" file give you the abilities

- to use the *member function* `set_partner(ref)` of the `Individual` *object* like that:
  `ref.set_partner(other)`
- to use the *member function* `get_partner()` of the `Individual` *object* like that:
  ```
  if (ref.get_partner().get_partner() == ref) { ... }
  ```
- to use the *member function* `has_partner()` which returns `true` if the individual has a partner, like that:
  ```
  if (ref.has_partner()) { x = ref.x + ref.get_partner().x; }
  ```

The main *member functions* of the `Population` *object*

- Distributed computations of standard statistics (weighted, filtered or both): count, sum, sum-by, average, min, max, percentiles, standard deviation, gini.
- Utilities promoting distributed computations: variables assignation, task launcher (by individuals, family heads, ...), swap between array of structs and struct of arrays
- Functionnalities for micro-simulation models:
  - align
  - birth (death and create are *member functions* of the `Individual` *object*)

The use of *lambda* functions

- Compute the share of top 10% for the variable x

```
1  auto const ptcls{ popul.pctl(v::x) };
2  auto const D9{ ptcls[90] };
3  auto const total{ popul.sum(v::x) };
4  auto const top10{ popul.sum_if(v::x,
5          [&](msl::RefConst ref) { return ref.x >= D9; }) };
6  std::cout << "Share of top 10% for x variable:\t" <<
7          top10 / total << '\n';
```

- Using the (not recommended) macro `MSL_LMBD`

```
auto const top10{ popul.sum_if(v::x, MSL_LMBD(ref.x >= D9)) };
```

First example: Impact Assessment

```cpp
#include "msl.hpp"

int main() {
  namespace v = msl::var;
  auto popul{ msl::load_population("population-2024") };
  msl::load_parameters("parameters.xlsx", 2024);
  msl::load_formats("formats.xlsx");
  popul.for_each_tax_fam(compute_income_tax); // First run
  std::cout << "Income tax revenues (baseline)\t"
            << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
  auto const tab0{ popul.weighted_sum_tax_fam_by(v::income_tax,
                                                 v::std_living_10) };
  msl::prm::business_expense_deduction_rate *= 1.1;
  popul.for_each_tax_fam(compute_income_tax); // Second run
  std::cout << "Income tax revenues (variant)\t"
            << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
  auto const tab1{ popul.weighted_sum_tax_fam_by(v::income_tax,
                                                 v::std_living_10) };
  tab0.save(tab1, "income-tax-variant.xlsx");
  return 0;
}
```

Set an alias for the namespace `msl::var`

```cpp
#include "msl.hpp"

int main() {
  namespace v = msl::var;
  auto popul{ msl::load_population("population-2024") };
  msl::load_parameters("parameters.xlsx", 2024);
  msl::load_formats("formats.xlsx");
  popul.for_each_tax_fam(compute_income_tax); // First run
  std::cout << "Income tax revenues (baseline)\t"
            << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
  auto const tab0{ popul.weighted_sum_tax_fam_by(v::income_tax,
                                                 v::std_living_10) };
  msl::prm::business_expense_deduction_rate *= 1.1;
  popul.for_each_tax_fam(compute_income_tax); // Second run
  std::cout << "Income tax revenues (variant)\t"
            << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
  auto const tab1{ popul.weighted_sum_tax_fam_by(v::income_tax,
                                                 v::std_living_10) };
  tab0.save(tab1, "income-tax-variant.xlsx");
  return 0;
}
```

Load the initial population from disk

```
1   #include "msl.hpp"
2
3   int main() {
4     namespace v = msl::var;
5     auto popul{ msl::load_population("population-2024") };
6     msl::load_parameters("parameters.xlsx", 2024);
7     msl::load_formats("formats.xlsx");
8     popul.for_each_tax_fam(compute_income_tax); // First run
9     std::cout << "Income tax revenues (baseline)\t"
10              << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
11    auto const tab0{ popul.weighted_sum_tax_fam_by(v::income_tax,
12                                                   v::std_living_10) };
13    msl::prm::business_expense_deduction_rate *= 1.1;
14    popul.for_each_tax_fam(compute_income_tax); // Second run
15    std::cout << "Income tax revenues (variant)\t"
16              << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
17    auto const tab1{ popul.weighted_sum_tax_fam_by(v::income_tax,
18                                                   v::std_living_10) };
19    tab0.save(tab1, "income-tax-variant.xlsx");
20    return 0;
21  }
```

Load the parameters from an Excel *Workbook* for 2024 year

```
1   #include "msl.hpp"
2
3   int main() {
4     namespace v = msl::var;
5     auto popul{ msl::load_population("population-2024") };
6     msl::load_parameters("parameters.xlsx", 2024);
7     msl::load_formats("formats.xlsx");
8     popul.for_each_tax_fam(compute_income_tax); // First run
9     std::cout << "Income tax revenues (baseline)\t"
10              << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
11    auto const tab0{ popul.weighted_sum_tax_fam_by(v::income_tax,
12                                                   v::std_living_10) };
13    msl::prm::business_expense_deduction_rate *= 1.1;
14    popul.for_each_tax_fam(compute_income_tax); // Second run
15    std::cout << "Income tax revenues (variant)\t"
16              << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
17    auto const tab1{ popul.weighted_sum_tax_fam_by(v::income_tax,
18                                                   v::std_living_10) };
19    tab0.save(tab1, "income-tax-variant.xlsx");
20    return 0;
21  }
```

Parallelised computation of the Income Tax by tax families

```
1   #include "msl.hpp"
2
3   int main() {
4     namespace v = msl::var;
5     auto popul{ msl::load_population("population-2024") };
6     msl::load_parameters("parameters.xlsx", 2024);
7     msl::load_formats("formats.xlsx");
8     popul.for_each_tax_fam(compute_income_tax); // First run
9     std::cout << "Income tax revenues (baseline)\t"
10              << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
11    auto const tab0{ popul.weighted_sum_tax_fam_by(v::income_tax,
12                                                   v::std_living_10) };
13    msl::prm::business_expense_deduction_rate *= 1.1;
14    popul.for_each_tax_fam(compute_income_tax); // Second run
15    std::cout << "Income tax revenues (variant)\t"
16              << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
17    auto const tab1{ popul.weighted_sum_tax_fam_by(v::income_tax,
18                                                   v::std_living_10) };
19    tab0.save(tab1, "income-tax-variant.xlsx");
20    return 0;
21  }
```

Parallelised weighted sum of the Income Tax

```
1   #include "msl.hpp"
2
3   int main() {
4     namespace v = msl::var;
5     auto popul{ msl::load_population("population-2024") };
6     msl::load_parameters("parameters.xlsx", 2024);
7     msl::load_formats("formats.xlsx");
8     popul.for_each_tax_fam(compute_income_tax); // First run
9     std::cout << "Income tax revenues (baseline)\t"
10              << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
11    auto const tab0{ popul.weighted_sum_tax_fam_by(v::income_tax,
12                                                   v::std_living_10) };
13    msl::prm::business_expense_deduction_rate *= 1.1;
14    popul.for_each_tax_fam(compute_income_tax); // Second run
15    std::cout << "Income tax revenues (variant)\t"
16              << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
17    auto const tab1{ popul.weighted_sum_tax_fam_by(v::income_tax,
18                                                   v::std_living_10) };
19    tab0.save(tab1, "income-tax-variant.xlsx");
20    return 0;
21  }
```

Uniform syntax for a model variable

```
1   #include "msl.hpp"
2
3   int main() {
4     namespace v = msl::var;
5     auto popul{ msl::load_population("population-2024") };
6     msl::load_parameters("parameters.xlsx", 2024);
7     msl::load_formats("formats.xlsx");
8     popul.for_each_tax_fam(compute_income_tax); // First run
9     std::cout << "Income tax revenues (baseline)\t"
10              << popul.weighted_sum_tax_fam( v::income_tax ) << '\n';
11    auto const tab0{ popul.weighted_sum_tax_fam_by(v::income_tax,
12                                                   v::std_living_10) };
13    msl::prm::business_expense_deduction_rate *= 1.1;
14    popul.for_each_tax_fam(compute_income_tax); // Second run
15    std::cout << "Income tax revenues (variant)\t"
16              << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
17    auto const tab1{ popul.weighted_sum_tax_fam_by(v::income_tax,
18                                                   v::std_living_10) };
19    tab0.save(tab1, "income-tax-variant.xlsx");
20    return 0;
21  }
```

Parallelised sum of the Income Tax by standard of living

```cpp
#include "msl.hpp"

int main() {
  namespace v = msl::var;
  auto popul{ msl::load_population("population-2024") };
  msl::load_parameters("parameters.xlsx", 2024);
  msl::load_formats("formats.xlsx");
  popul.for_each_tax_fam(compute_income_tax); // First run
  std::cout << "Income tax revenues (baseline)\t"
            << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
  auto const tab0{ popul.weighted_sum_tax_fam_by(v::income_tax,
                                                 v::std_living_10) };
  msl::prm::business_expense_deduction_rate *= 1.1;
  popul.for_each_tax_fam(compute_income_tax); // Second run
  std::cout << "Income tax revenues (variant)\t"
            << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
  auto const tab1{ popul.weighted_sum_tax_fam_by(v::income_tax,
                                                 v::std_living_10) };
  tab0.save(tab1, "income-tax-variant.xlsx");
  return 0;
}
```

Increase by 10 % of a Income Tax parameter

```cpp
#include "msl.hpp"

int main() {
  namespace v = msl::var;
  auto popul{ msl::load_population("population-2024") };
  msl::load_parameters("parameters.xlsx", 2024);
  msl::load_formats("formats.xlsx");
  popul.for_each_tax_fam(compute_income_tax); // First run
  std::cout << "Income tax revenues (baseline)\t"
            << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
  auto const tab0{ popul.weighted_sum_tax_fam_by(v::income_tax,
                                                 v::std_living_10) };
  msl::prm::business_expense_deduction_rate *= 1.1;
  popul.for_each_tax_fam(compute_income_tax); // Second run
  std::cout << "Income tax revenues (variant)\t"
            << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
  auto const tab1{ popul.weighted_sum_tax_fam_by(v::income_tax,
                                                 v::std_living_10) };
  tab0.save(tab1, "income-tax-variant.xlsx");
  return 0;
}
```

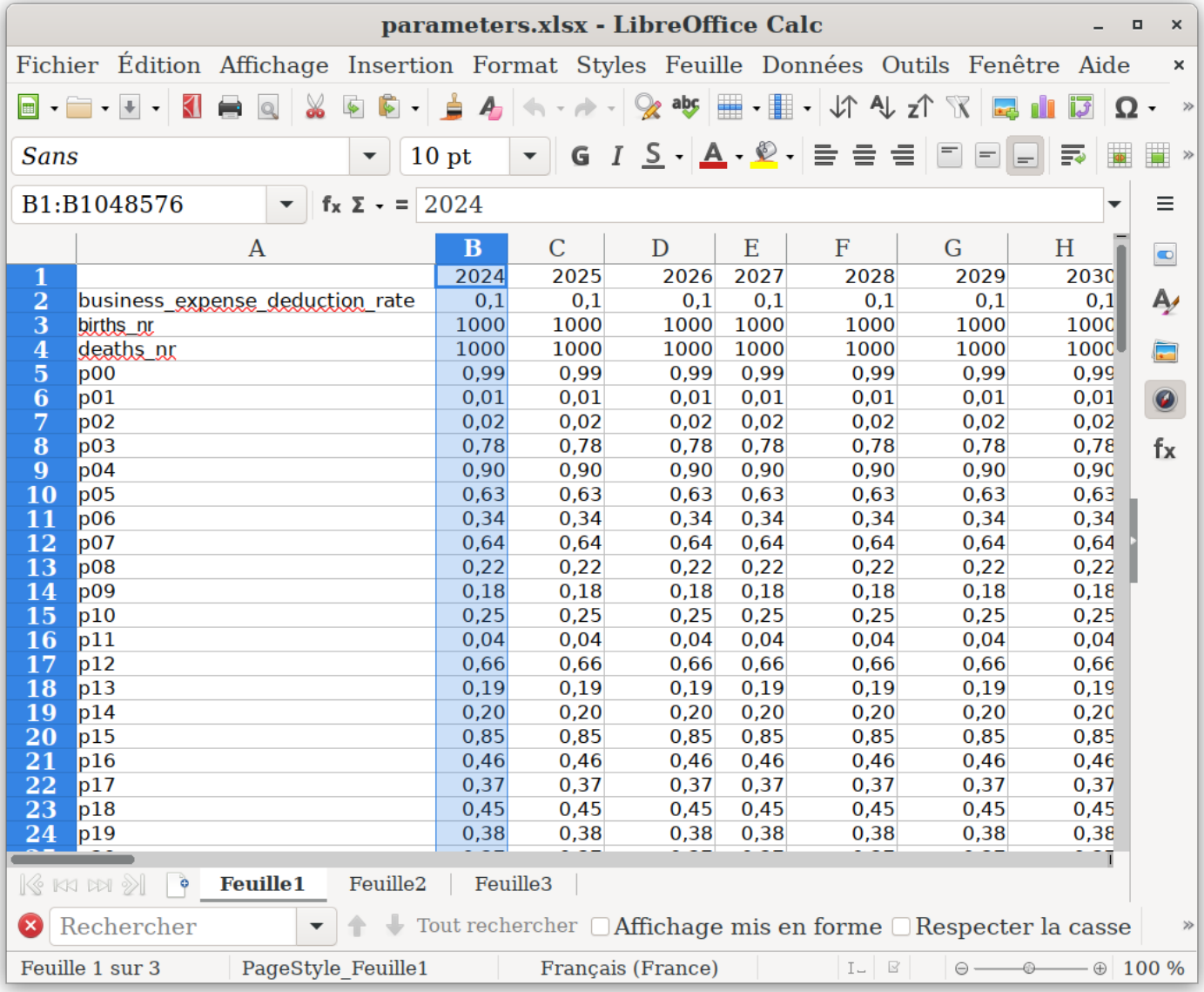
Run again the computation of the Income Tax

```
1   #include "msl.hpp"
2
3   int main() {
4     namespace v = msl::var;
5     auto popul{ msl::load_population("population-2024") };
6     msl::load_parameters("parameters.xlsx", 2024);
7     msl::load_formats("formats.xlsx");
8     popul.for_each_tax_fam(compute_income_tax); // First run
9     std::cout << "Income tax revenues (baseline)\t"
10              << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
11    auto const tab0{ popul.weighted_sum_tax_fam_by(v::income_tax,
12                                                   v::std_living_10) };
13    msl::prm::business_expense_deduction_rate *= 1.1;
14    popul. for_each_tax_fam (compute_income_tax); // Second run
15    std::cout << "Income tax revenues (variant)\t"
16              << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
17    auto const tab1{ popul.weighted_sum_tax_fam_by(v::income_tax,
18                                                   v::std_living_10) };
19    tab0.save(tab1, "income-tax-variant.xlsx");
20    return 0;
21  }
```

Save in a Excel *Workbook* the two tabulars

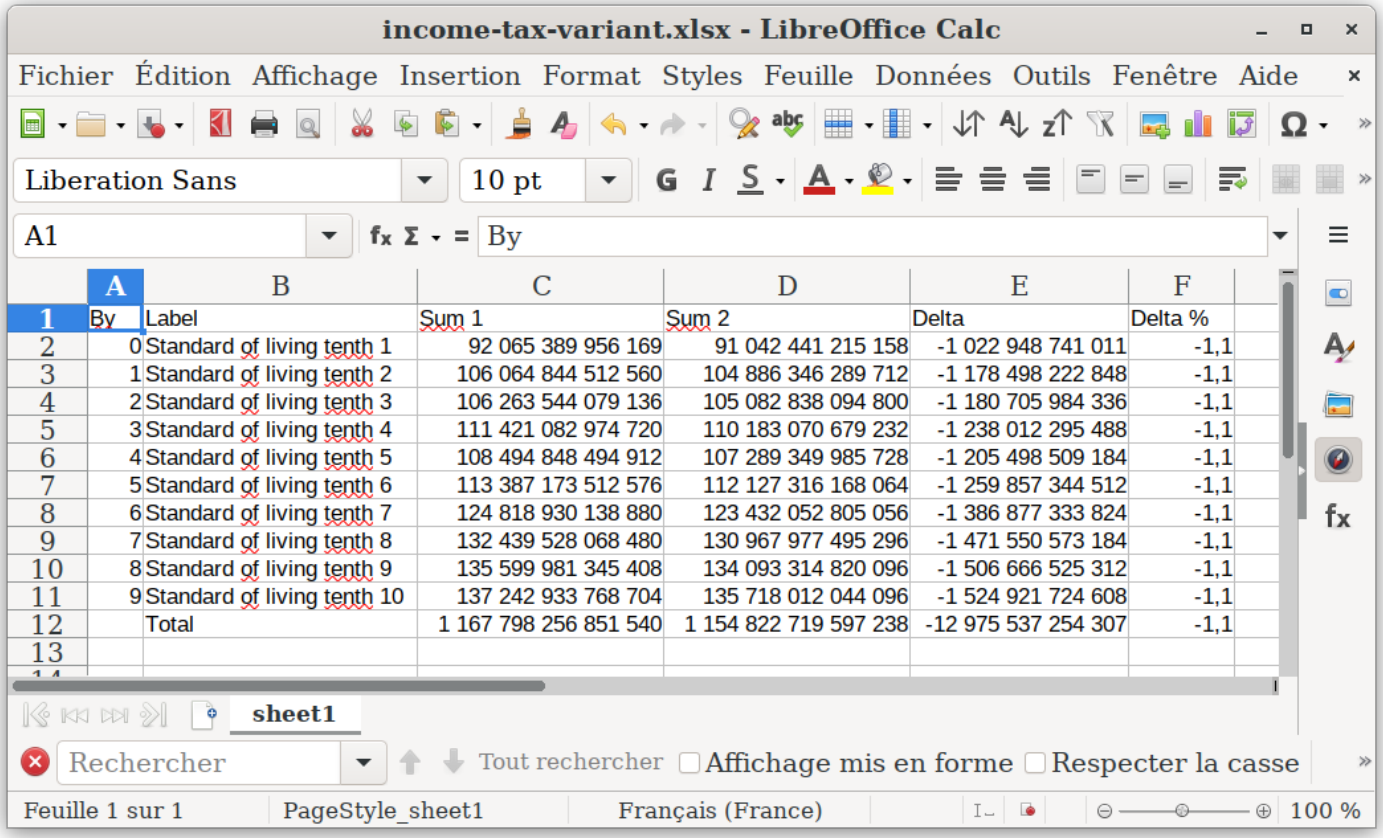
```
1   #include "msl.hpp"
2
3   int main() {
4     namespace v = msl::var;
5     auto popul{ msl::load_population("population-2024") };
6     msl::load_parameters("parameters.xlsx", 2024);
7     msl::load_formats("formats.xlsx");
8     popul.for_each_tax_fam(compute_income_tax); // First run
9     std::cout << "Income tax revenues (baseline)\t"
10              << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
11    auto const tab0{ popul.weighted_sum_tax_fam_by(v::income_tax,
12                                                   v::std_living_10) };
13    msl::prm::business_expense_deduction_rate *= 1.1;
14    popul.for_each_tax_fam(compute_income_tax); // Second run
15    std::cout << "Income tax revenues (variant)\t"
16              << popul.weighted_sum_tax_fam(v::income_tax) << '\n';
17    auto const tab1{ popul.weighted_sum_tax_fam_by(v::income_tax,
18                                                   v::std_living_10) };
19    tab0.save(tab1, "income-tax-variant.xlsx");
20    return 0;
21  }
```

The parameters Excel *Workbook* for impact assessment



	A	B	C	D	E	F	G	H
1		2024	2025	2026	2027	2028	2029	2030
2	business_expense_deduction_rate	0,1	0,1	0,1	0,1	0,1	0,1	0,1
3	births_nr	1000	1000	1000	1000	1000	1000	1000
4	deaths_nr	1000	1000	1000	1000	1000	1000	1000
5	p00	0,99	0,99	0,99	0,99	0,99	0,99	0,99
6	p01	0,01	0,01	0,01	0,01	0,01	0,01	0,01
7	p02	0,02	0,02	0,02	0,02	0,02	0,02	0,02
8	p03	0,78	0,78	0,78	0,78	0,78	0,78	0,78
9	p04	0,90	0,90	0,90	0,90	0,90	0,90	0,90
10	p05	0,63	0,63	0,63	0,63	0,63	0,63	0,63
11	p06	0,34	0,34	0,34	0,34	0,34	0,34	0,34
12	p07	0,64	0,64	0,64	0,64	0,64	0,64	0,64
13	p08	0,22	0,22	0,22	0,22	0,22	0,22	0,22
14	p09	0,18	0,18	0,18	0,18	0,18	0,18	0,18
15	p10	0,25	0,25	0,25	0,25	0,25	0,25	0,25
16	p11	0,04	0,04	0,04	0,04	0,04	0,04	0,04
17	p12	0,66	0,66	0,66	0,66	0,66	0,66	0,66
18	p13	0,19	0,19	0,19	0,19	0,19	0,19	0,19
19	p14	0,20	0,20	0,20	0,20	0,20	0,20	0,20
20	p15	0,85	0,85	0,85	0,85	0,85	0,85	0,85
21	p16	0,46	0,46	0,46	0,46	0,46	0,46	0,46
22	p17	0,37	0,37	0,37	0,37	0,37	0,37	0,37
23	p18	0,45	0,45	0,45	0,45	0,45	0,45	0,45
24	p19	0,38	0,38	0,38	0,38	0,38	0,38	0,38

The resulting Excel *Workbook* for impact assessment



	A	B	C	D	E	F
1	By	Label	Sum 1	Sum 2	Delta	Delta %
2	0	Standard of living tenth 1	92 065 389 956 169	91 042 441 215 158	-1 022 948 741 011	-1,1
3	1	Standard of living tenth 2	106 064 844 512 560	104 886 346 289 712	-1 178 498 222 848	-1,1
4	2	Standard of living tenth 3	106 263 544 079 136	105 082 838 094 800	-1 180 705 984 336	-1,1
5	3	Standard of living tenth 4	111 421 082 974 720	110 183 070 679 232	-1 238 012 295 488	-1,1
6	4	Standard of living tenth 5	108 494 848 494 912	107 289 349 985 728	-1 205 498 509 184	-1,1
7	5	Standard of living tenth 6	113 387 173 512 576	112 127 316 168 064	-1 259 857 344 512	-1,1
8	6	Standard of living tenth 7	124 818 930 138 880	123 432 052 805 056	-1 386 877 333 824	-1,1
9	7	Standard of living tenth 8	132 439 528 068 480	130 967 977 495 296	-1 471 550 573 184	-1,1
10	8	Standard of living tenth 9	135 599 981 345 408	134 093 314 820 096	-1 506 666 525 312	-1,1
11	9	Standard of living tenth 10	137 242 933 768 704	135 718 012 044 096	-1 524 921 724 608	-1,1
12		Total	1 167 798 256 851 540	1 154 822 719 597 238	-12 975 537 254 307	-1,1
13						

Elapsed times: 100,000 individuals & 50 variables

Function	ms	%
`msl::load_population`	44	74
`msl::load_parameters`	2	3
`msl::load_formats`	1	2
`Population::for_each_tax_fam` †	4	7
`Population::weighted_sum_tax_fam` †	ε	1
`Population::weighted_sum_tax_fam_by` †	1	2
`Population::for_each_tax_fam` ‡	4	7
`Population::weighted_sum_tax_fam` ‡	ε	1
`Population::weighted_sum_tax_fam_by` ‡	1	2
`Tabular::save`	1	2
Total	59	100

ms: milliseconds; † first run; ‡ second run.

Second example: Population Projection

```cpp
#include "msl.hpp"

int main() {
  namespace v = msl::var;
  std::mt19937 gnr{}; std::uniform_real_distribution<> unif{0, 1};
  auto popul{ msl::load_population("population-2024") };
  for (int t{ 2025 }; t < 2025 + 50; ++t) {
    msl::to_day = msl::days_from_civil(t, 12, 31); // Last of December
    msl::load_parameters("parameters.xlsx", t);
    popul.for_each(MSL_FNCTN(ref.prob = (ref.female() &&
      msl::within(20, ref.age(), 40)) ? 2. / (40 - 20) : 0));
    popul.align(msl::prm::births_nr, v::prob, [&](msl::Ref mother) {
      msl::Ref baby{ popul.birth(mother, msl::to_day - gnr() % 365) };
      baby.set_female(unif(gnr) < .48);
    });
    popul.for_each(MSL_FNCTN(ref.prob = (ref.age() > 60) ? .05 : 0));
    popul.align(msl::prm::deaths_nr, v::prob, MSL_FNCTN(ref.death()));
    std::cout << t << '\t' << popul.count_if(v::one) << '\t'
              << popul.avg(MSL_LMBD(ref.age())) << '\n';
  }
  return 0;
}
```

Set an alias for the namespace `msl::var`

```
1   #include "msl.hpp"
2
3   int main() {
4     namespace v = msl::var;
5     std::mt19937 gnr{}; std::uniform_real_distribution<> unif{0, 1};
6     auto popul{ msl::load_population("population-2024") };
7     for (int t{ 2025 }; t < 2025 + 50; ++t) {
8       msl::to_day = msl::days_from_civil(t, 12, 31); // Last of December
9       msl::load_parameters("parameters.xlsx", t);
10      popul.for_each(MSL_FNCTN(ref.prob = (ref.female() &&
11        msl::within(20, ref.age(), 40)) ? 2. / (40 - 20) : 0));
12      popul.align(msl::prm::births_nr, v::prob, [&](msl::Ref mother) {
13        msl::Ref baby{ popul.birth(mother, msl::to_day - gnr() % 365) };
14        baby.set_female(unif(gnr) < .48);
15      });
16      popul.for_each(MSL_FNCTN(ref.prob = (ref.age() > 60) ? .05 : 0));
17      popul.align(msl::prm::deaths_nr, v::prob, MSL_FNCTN(ref.death()));
18      std::cout << t << '\t' << popul.count_if(v::one) << '\t'
19                << popul.avg(MSL_LMBD(ref.age())) << '\n';
20    }
21    return 0;
22  }
```

Manage pseudo-random number generators

```
1   #include "msl.hpp"
2
3   int main() {
4     namespace v = msl::var;
5     std::mt19937 gnr{}; std::uniform_real_distribution<> unif{0, 1};
6     auto popul{ msl::load_population("population-2024") };
7     for (int t{ 2025 }; t < 2025 + 50; ++t) {
8       msl::to_day = msl::days_from_civil(t, 12, 31); // Last of December
9       msl::load_parameters("parameters.xlsx", t);
10      popul.for_each(MSL_FNCTN(ref.prob = (ref.female() &&
11        msl::within(20, ref.age(), 40)) ? 2. / (40 - 20) : 0));
12      popul.align(msl::prm::births_nr, v::prob, [&](msl::Ref mother) {
13        msl::Ref baby{ popul.birth(mother, msl::to_day - gnr() % 365) };
14        baby.set_female(unif(gnr) < .48);
15      });
16      popul.for_each(MSL_FNCTN(ref.prob = (ref.age() > 60) ? .05 : 0));
17      popul.align(msl::prm::deaths_nr, v::prob, MSL_FNCTN(ref.death()));
18      std::cout << t << '\t' << popul.count_if(v::one) << '\t'
19                << popul.avg(MSL_LMBD(ref.age())) << '\n';
20    }
21    return 0;
22  }
```

Load the initial population from disk

```
1   #include "msl.hpp"
2
3   int main() {
4     namespace v = msl::var;
5     std::mt19937 gnr{}; std::uniform_real_distribution<> unif{0, 1};
6     auto popul{ msl::load_population("population-2024") };
7     for (int t{ 2025 }; t < 2025 + 50; ++t) {
8       msl::to_day = msl::days_from_civil(t, 12, 31); // Last of December
9       msl::load_parameters("parameters.xlsx", t);
10      popul.for_each(MSL_FNCTN(ref.prob = (ref.female() &&
11        msl::within(20, ref.age(), 40)) ? 2. / (40 - 20) : 0));
12      popul.align(msl::prm::births_nr, v::prob, [&](msl::Ref mother) {
13        msl::Ref baby{ popul.birth(mother, msl::to_day - gnr() % 365) };
14        baby.set_female(unif(gnr) < .48);
15      });
16      popul.for_each(MSL_FNCTN(ref.prob = (ref.age() > 60) ? .05 : 0));
17      popul.align(msl::prm::deaths_nr, v::prob, MSL_FNCTN(ref.death()));
18      std::cout << t << '\t' << popul.count_if(v::one) << '\t'
19                << popul.avg(MSL_LMBD(ref.age())) << '\n';
20    }
21    return 0;
22  }
```

50 years of projection

```
1  #include "msl.hpp"
2
3  int main() {
4    namespace v = msl::var;
5    std::mt19937 gnr{}; std::uniform_real_distribution<> unif{0, 1};
6    auto popul{ msl::load_population("population-2024") };
7    for (int t{ 2025 }; t < 2025 + 50; ++t) {
8      msl::to_day = msl::days_from_civil(t, 12, 31); // Last of December
9      msl::load_parameters("parameters.xlsx", t);
10     popul.for_each(MSL_FNCTN(ref.prob = (ref.female() &&
11       msl::within(20, ref.age(), 40)) ? 2. / (40 - 20) : 0));
12     popul.align(msl::prm::births_nr, v::prob, [&](msl::Ref mother) {
13       msl::Ref baby{ popul.birth(mother, msl::to_day - gnr() % 365) };
14       baby.set_female(unif(gnr) < .48);
15     });
16     popul.for_each(MSL_FNCTN(ref.prob = (ref.age() > 60) ? .05 : 0));
17     popul.align(msl::prm::deaths_nr, v::prob, MSL_FNCTN(ref.death()));
18     std::cout << t << '\t' << popul.count_if(v::one) << '\t'
19               << popul.avg(MSL_LMBD(ref.age())) << '\n';
20   }
21   return 0;
22 }
```

Set the date using the `days_from_civil` function

```
1   #include "msl.hpp"
2
3   int main() {
4     namespace v = msl::var;
5     std::mt19937 gnr{}; std::uniform_real_distribution<> unif{0, 1};
6     auto popul{ msl::load_population("population-2024") };
7     for (int t{ 2025 }; t < 2025 + 50; ++t) {
8       msl::to_day = msl::days_from_civil( t , 12, 31); // Last of December
9       msl::load_parameters("parameters.xlsx", t);
10      popul.for_each(MSL_FNCTN(ref.prob = (ref.female() &&
11        msl::within(20, ref.age(), 40)) ? 2. / (40 - 20) : 0));
12      popul.align(msl::prm::births_nr, v::prob, [&](msl::Ref mother) {
13        msl::Ref baby{ popul.birth(mother, msl::to_day - gnr() % 365) };
14        baby.set_female(unif(gnr) < .48);
15      });
16      popul.for_each(MSL_FNCTN(ref.prob = (ref.age() > 60) ? .05 : 0));
17      popul.align(msl::prm::deaths_nr, v::prob, MSL_FNCTN(ref.death()));
18      std::cout << t << '\t' << popul.count_if(v::one) << '\t'
19                << popul.avg(MSL_LMBD(ref.age())) << '\n';
20    }
21    return 0;
22  }
```

Load the parameters for the current year

```
1   #include "msl.hpp"
2
3   int main() {
4     namespace v = msl::var;
5     std::mt19937 gnr{}; std::uniform_real_distribution<> unif{0, 1};
6     auto popul{ msl::load_population("population-2024") };
7     for (int t{ 2025 }; t < 2025 + 50; ++t) {
8       msl::to_day = msl::days_from_civil(t, 12, 31); // Last of December
9       msl::load_parameters("parameters.xlsx", t);
10      popul.for_each(MSL_FNCTN(ref.prob = (ref.female() &&
11        msl::within(20, ref.age(), 40)) ? 2. / (40 - 20) : 0));
12      popul.align(msl::prm::births_nr, v::prob, [&](msl::Ref mother) {
13        msl::Ref baby{ popul.birth(mother, msl::to_day - gnr() % 365) };
14        baby.set_female(unif(gnr) < .48);
15      });
16      popul.for_each(MSL_FNCTN(ref.prob = (ref.age() > 60) ? .05 : 0));
17      popul.align(msl::prm::deaths_nr, v::prob, MSL_FNCTN(ref.death()));
18      std::cout << t << '\t' << popul.count_if(v::one) << '\t'
19                << popul.avg(MSL_LMBD(ref.age())) << '\n';
20    }
21    return 0;
22  }
```

Set the probability of birth for each fertile woman

```
1   #include "msl.hpp"
2
3   int main() {
4     namespace v = msl::var;
5     std::mt19937 gnr{}; std::uniform_real_distribution<> unif{0, 1};
6     auto popul{ msl::load_population("population-2024") };
7     for (int t{ 2025 }; t < 2025 + 50; ++t) {
8       msl::to_day = msl::days_from_civil(t, 12, 31); // Last of December
9       msl::load_parameters("parameters.xlsx", t);
10      popul.for_each(MSL_FNCTN(ref.prob = (ref.female() &&
11        msl::within(20, ref.age(), 40)) ? 2. / (40 - 20) : 0));
12      popul.align(msl::prm::births_nr, v::prob, [&](msl::Ref mother) {
13        msl::Ref baby{ popul.birth(mother, msl::to_day - gnr() % 365) };
14        baby.set_female(unif(gnr) < .48);
15      });
16      popul.for_each(MSL_FNCTN(ref.prob = (ref.age() > 60) ? .05 : 0));
17      popul.align(msl::prm::deaths_nr, v::prob, MSL_FNCTN(ref.death()));
18      std::cout << t << '\t' << popul.count_if(v::one) << '\t'
19                << popul.avg(MSL_LMBD(ref.age())) << '\n';
20    }
21    return 0;
22  }
```

Align the number of births on an exogenous parameter

```cpp
#include "msl.hpp"

int main() {
  namespace v = msl::var;
  std::mt19937 gnr{}; std::uniform_real_distribution<> unif{0, 1};
  auto popul{ msl::load_population("population-2024") };
  for (int t{ 2025 }; t < 2025 + 50; ++t) {
    msl::to_day = msl::days_from_civil(t, 12, 31); // Last of December
    msl::load_parameters("parameters.xlsx", t);
    popul.for_each(MSL_FNCTN(ref.prob = (ref.female() &&
      msl::within(20, ref.age(), 40)) ? 2. / (40 - 20) : 0));
    popul.align(msl::prm::births_nr, v::prob,[&](msl::Ref mother) {
      msl::Ref baby{ popul.birth(mother, msl::to_day - gnr() % 365) };
      baby.set_female(unif(gnr) < .48);
    });
    popul.for_each(MSL_FNCTN(ref.prob = (ref.age() > 60) ? .05 : 0));
    popul.align(msl::prm::deaths_nr, v::prob, MSL_FNCTN(ref.death()));
    std::cout << t << '\t' << popul.count_if(v::one) << '\t'
              << popul.avg(MSL_LMBD(ref.age())) << '\n';
  }
  return 0;
}
```

Simulate the births, set the birthday and the gender

```
1   #include "msl.hpp"
2
3   int main() {
4     namespace v = msl::var;
5     std::mt19937 gnr{}; std::uniform_real_distribution<> unif{0, 1};
6     auto popul{ msl::load_population("population-2024") };
7     for (int t{ 2025 }; t < 2025 + 50; ++t) {
8       msl::to_day = msl::days_from_civil(t, 12, 31); // Last of December
9       msl::load_parameters("parameters.xlsx", t);
10      popul.for_each(MSL_FNCTN(ref.prob = (ref.female() &&
11        msl::within(20, ref.age(), 40)) ? 2. / (40 - 20) : 0));
12      popul.align(msl::prm::births_nr, v::prob, [&](msl::Ref mother) {
13        msl::Ref baby{ popul.birth(mother, msl::to_day - gnr() % 365) };
14        baby.set_female(unif(gnr) < .48);
15      });
16      popul.for_each(MSL_FNCTN(ref.prob = (ref.age() > 60) ? .05 : 0));
17      popul.align(msl::prm::deaths_nr, v::prob, MSL_FNCTN(ref.death()));
18      std::cout << t << '\t' << popul.count_if(v::one) << '\t'
19                << popul.avg(MSL_LMBD(ref.age())) << '\n';
20    }
21    return 0;
22  }
```

Set the probability of death for each individual

```cpp
#include "msl.hpp"

int main() {
  namespace v = msl::var;
  std::mt19937 gnr{}; std::uniform_real_distribution<> unif(0, 1);
  auto popul{ msl::load_population("population-2024") };
  for (int t{ 2025 }; t < 2025 + 50; ++t) {
    msl::to_day = msl::days_from_civil(t, 12, 31); // Last of december
    msl::load_parameters("parameters.xlsx", t);
    popul.for_each(MSL_FNCTN(ref.prob = (ref.female() &&
      msl::within(20, ref.age(), 40)) ? 2. / (40 - 20) : 0));
    popul.align(msl::prm::births_nr, v::prob, [&](msl::Ref mother) {
      msl::Ref baby{ popul.birth(mother, msl::to_day - gnr() % 365) };
      baby.set_female(unif(gnr) < .48);
    });
    popul.for_each(MSL_FNCTN(ref.prob = (ref.age() > 60) ? .05 : 0));
    popul.align(msl::prm::deaths_nr, v::prob, MSL_FNCTN(ref.death()));
    std::cout << t << '\t' << popul.count_if(v::one) << '\t'
              << popul.avg(MSL_LMBD(ref.age())) << '\n';
  }
  return 0;
}
```

Align the number of deaths on an exogeneous parameter

```cpp
#include "msl.hpp"

int main() {
  namespace v = msl::var;
  std::mt19937 gnr{}; std::uniform_real_distribution<> unif(0, 1);
  auto popul{ msl::load_population("population-2024") };
  for (int t{ 2025 }; t < 2025 + 50; ++t) {
    msl::to_day = msl::days_from_civil(t, 12, 31); // Last of december
    msl::load_parameters("parameters.xlsx", t);
    popul.for_each(MSL_FNCTN(ref.prob = (ref.female() &&
      msl::within(20, ref.age(), 40)) ? 2. / (40 - 20) : 0));
    popul.align(msl::prm::births_nr, v::prob, [&](msl::Ref mother) {
      msl::Ref baby{ popul.birth(mother, msl::to_day - gnr() % 365) };
      baby.set_female(unif(gnr) < .48);
    });
    popul.for_each(MSL_FNCTN(ref.prob = (ref.age() > 60) ? : 0));
    popul.align(msl::prm::deaths_nr, v::prob,MSL_FNCTN(ref.death()));
    std::cout << t << '\t' << popul.count_if(v::one) << '\t'
              << popul.avg(MSL_LMBD(ref.age())) << '\n';
  }
  return 0;
}
```

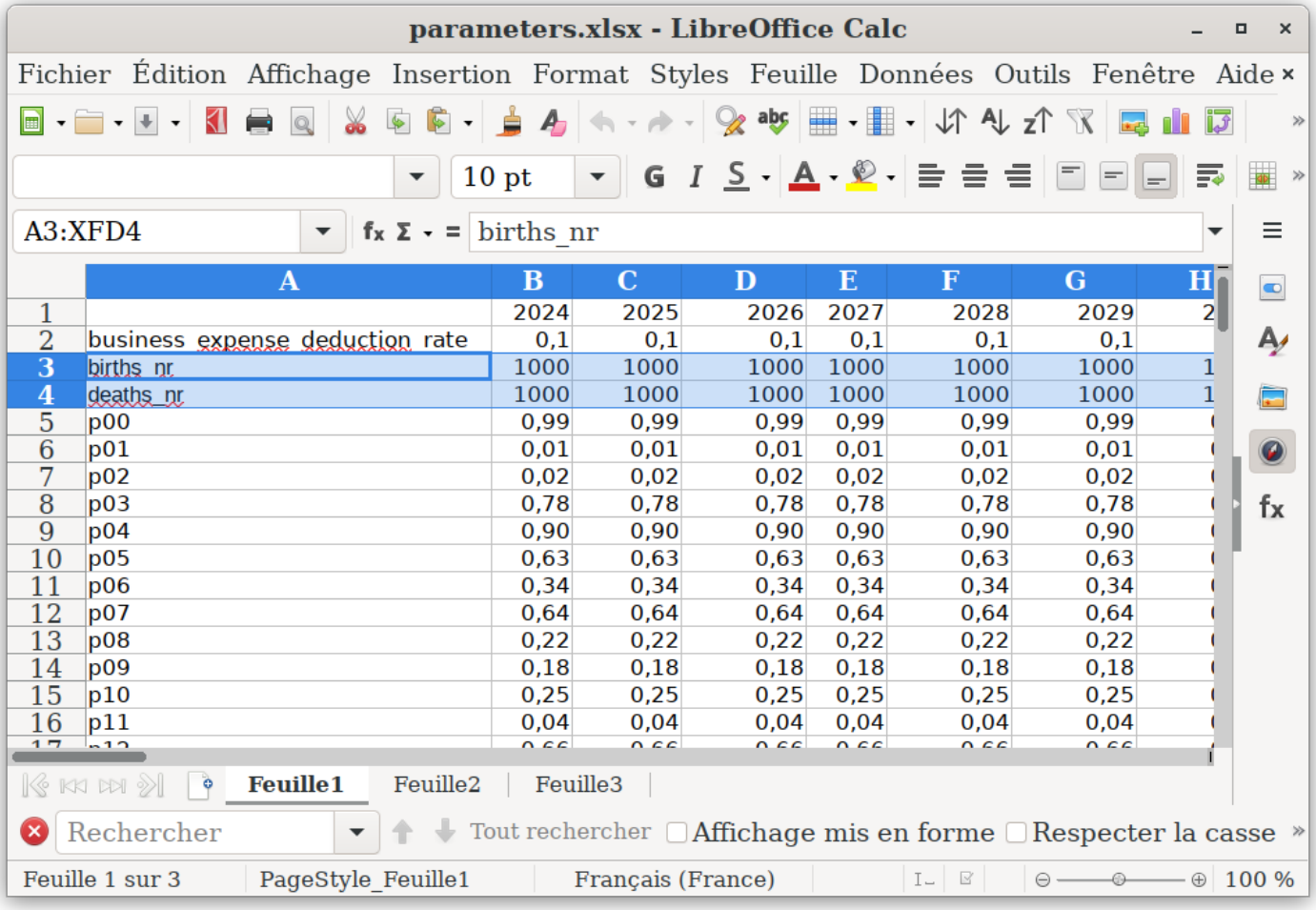
Simulate the deaths

```cpp
#include "msl.hpp"

int main() {
  namespace v = msl::var;
  std::mt19937 gnr{}; std::uniform_real_distribution<> unif(0, 1);
  auto popul{ msl::load_population("population-2024") };
  for (int t{ 2025 }; t < 2025 + 50; ++t) {
    msl::to_day = msl::days_from_civil(t, 12, 31); // Last of december
    msl::load_parameters("parameters.xlsx", t);
    popul.for_each(MSL_FNCTN(ref.prob = (ref.female() &&
      msl::within(20, ref.age(), 40)) ? 2. / (40 - 20) : 0));
    popul.align(msl::prm::births_nr, v::prob, [&](msl::Ref mother) {
      msl::Ref baby{ popul.birth(mother, msl::to_day - gnr() % 365) };
      baby.set_female(unif(gnr) < .48);
    });
    popul.for_each(MSL_FNCTN(ref.prob = (ref.age() > 60) ? : 0));
    popul.align(msl::prm::deaths_nr, v::prob, MSL_FNCTN( ref.death() ));
    std::cout << t << '\t' << popul.count_if(v::one) << '\t'
              << popul.avg(MSL_LMBD(ref.age())) << '\n';
  }
  return 0;
}
```

Print the size of the population and the average age

```cpp
#include "msl.hpp"

int main() {
  namespace v = msl::var;
  std::mt19937 gnr{}; std::uniform_real_distribution<> unif(0, 1);
  auto popul{ msl::load_population("population-2024") };
  for (int t{ 2025 }; t < 2025 + 50; ++t) {
    msl::to_day = msl::days_from_civil(t, 12, 31); // Last of december
    msl::load_parameters("parameters.xlsx", t);
    popul.for_each(MSL_FNCTN(ref.prob = (ref.female() &&
      msl::within(20, ref.age(), 40)) ? 2. / (40 - 20) : 0));
    popul.align(msl::prm::births_nr, v::prob, [&](msl::Ref mother) {
      msl::Ref baby{ popul.birth(mother, msl::to_day - gnr() % 365) };
      baby.set_female(unif(gnr) < .48);
    });
    popul.for_each(MSL_FNCTN(ref.prob = (ref.age() > 60) ? : 0));
    popul.align(msl::prm::deaths_nr, v::prob, MSL_FNCTN(ref.death()));
    std::cout << t << '\t' << popul.count_if(v::one) << '\t'
              << popul.avg(MSL_LMBD(ref.age())) << '\n';
  }
  return 0;
}
```

The parameters Excel *Workbook* for population projection



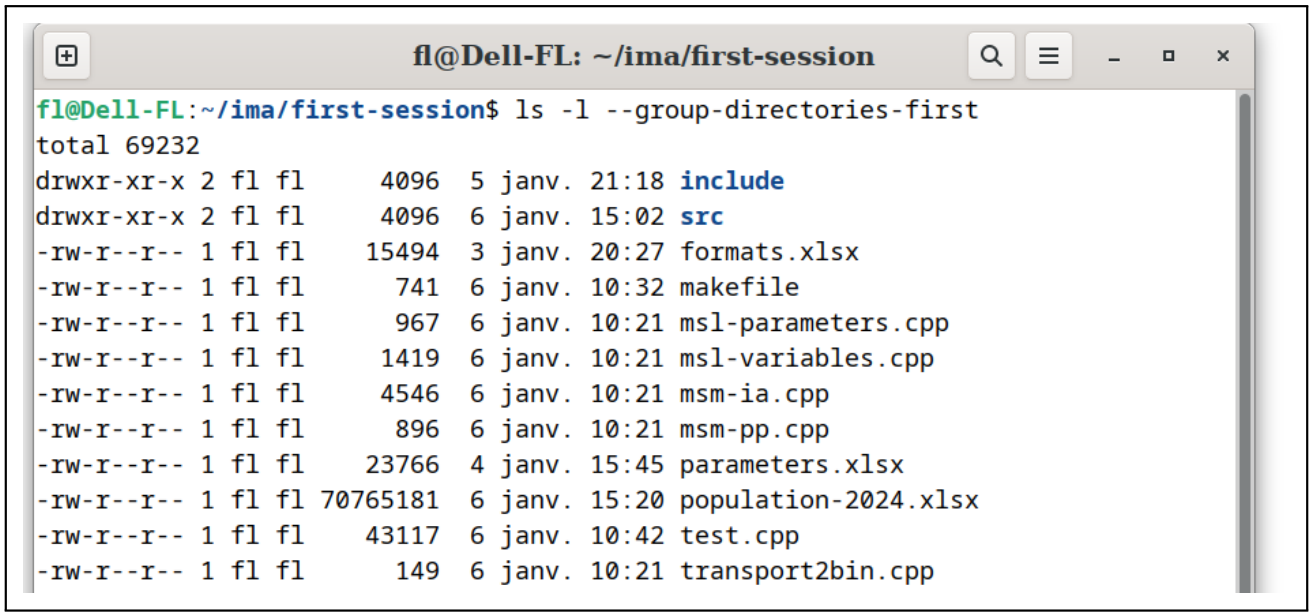
	A	B	C	D	E	F	G	H
1		2024	2025	2026	2027	2028	2029	2
2	business_expense_deduction_rate	0,1	0,1	0,1	0,1	0,1	0,1	
3	births_nr	1000	1000	1000	1000	1000	1000	1
4	deaths_nr	1000	1000	1000	1000	1000	1000	1
5	p00	0,99	0,99	0,99	0,99	0,99	0,99	
6	p01	0,01	0,01	0,01	0,01	0,01	0,01	
7	p02	0,02	0,02	0,02	0,02	0,02	0,02	
8	p03	0,78	0,78	0,78	0,78	0,78	0,78	
9	p04	0,90	0,90	0,90	0,90	0,90	0,90	
10	p05	0,63	0,63	0,63	0,63	0,63	0,63	
11	p06	0,34	0,34	0,34	0,34	0,34	0,34	
12	p07	0,64	0,64	0,64	0,64	0,64	0,64	
13	p08	0,22	0,22	0,22	0,22	0,22	0,22	
14	p09	0,18	0,18	0,18	0,18	0,18	0,18	
15	p10	0,25	0,25	0,25	0,25	0,25	0,25	
16	p11	0,04	0,04	0,04	0,04	0,04	0,04	

Elapsed times per year: 100,000 individuals & 50 variables

Function	ms	%
`msl::load_parameters`	6	10
`Population::for_each` †	1	1
`Population::align` †	39	74
`Population::for_each` ‡	ε	1
`Population::align` ‡	6	11
`Population::count_if` and `Population::avg`	1	2
Total	53	100

ms: milliseconds; † birth; ‡ death.

First session: files layout

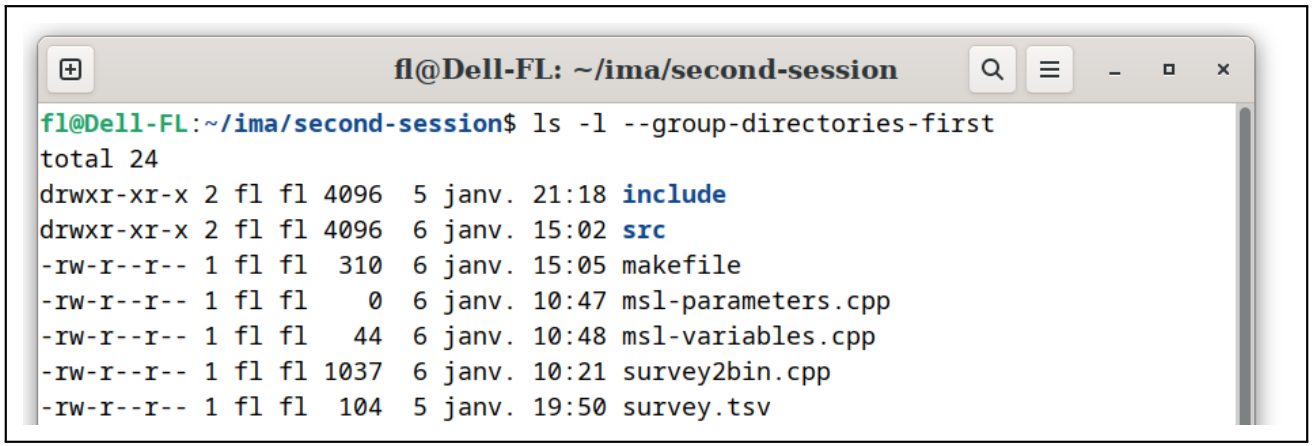


File	Description
`formats.xlsx`	Excel *Workbook* for the `CTG` variable formats
`makefile`	File for the *make* build utility
`msl-parameters.cpp`	File to fill with the model parameters
`msl-variables.cpp`	File to fill with the model variables
`msm-ia.cpp`	Source of the *impact assessment* micro-simulation model
`msm-pp.cpp`	Source of the *population projection* micro-simulation model
`parameters.xlsx`	Excel *Workbook* to fill with the parameter values
`population-2024.xlsx`	Excel *Workbook* of the initial population (transport format)
`test.cpp`	Source to run a series of checks to test the library
`transport2bin.cpp`	Source to transform the population transport ⟶ binary

First session: the road map

1. Go to `http://legendre.ovh/` to get the library source files and the data
2. Unzip the archives to obtain the files layout of the previous slide in a fresh directory
3. In a terminal, run the CLI command `make lib` to build the library
4. Run the command `make test` to compile the `test` program
5. Run the command `./test` to check the library
6. Run the command `make transport2bin` to compile the `transport2bin` program
7. Run the command `./transport2bin` to execute this program
8. Run the command `make msm-ia` to compile the `msm-ia` program – micro-simulation model impact assessment
9. Run the command `./msm-ia` to execute this program
10. Run the command `make msm-pp` to compile the `msm-pp` program – micro-simulation model population projection
11. Run the command `./msm-pp` to execute this program

Second session: files layout



File	Description
`makefile`	File for the *make* build utility
`msl-parameters.cpp`	File to fill with the model parameters
`msl-variables.cpp`	File to fill with the model variables
`survey.tsv`	Data to populate the model – tabular separated values
`survey2bin.cpp`	Source of the program to populate the model

Second session: the road map

1. Go to `http://legendre.ovh/` to get the source files of the second session
2. Unzip the archives to obtain the files layout of the previous slide in a fresh directory
3. In a terminal, run the CLI command `make lib` to build the library
4. Fill with another variables the "`msl-variables.cpp`" file
5. Fill with parameters the "`msl-parameters.cpp`" file
6. Run the command `make survey2bin` to compile the `survey2bin` program
7. Run the command `./survey2bin` to execute this program
8. Play with the model

Thank you for your attention