# ctools - Action #354

Feature # 353 (Closed): ctobssim: Parallelize event simulations

## Set deterministic seed values for each observation.

07/20/2012 10:42 PM - Knödlseder Jürgen

| | | | |
|---|---|---|---|
| **Status:** | Closed | **Start date:** | 07/30/2012 |
| **Priority:** | Normal | **Due date:** | |
| **Assigned To:** | | **% Done:** | 100% |
| **Category:** | | **Estimated time:** | 24.00 hours |
| **Target version:** | 00-05-00 | | |

**Description**

In the actual implementation, the state of the random number generator at the start of observation N+1 depends on the state of the random number generator at the end of observation N. This has to be modified to allow a deterministic parallelization of the code.

For the moment, the ctobssim class has one global random number generator m_ran:

```
GRan        m_ran;      //!< Random number generator
```

This should be replaced by a vector of random number generators, one for each observation:

```
std::vector<GRan>  m_ran;  //!< Random number generators
```

The assignment of the seed values for the random number generators is done in ctobssim::get_parameters. Actually the code is:

```
    // Get other parameters
    m_seed = (*this)["seed"].integer();

    // Initialise random number generator
    m_ran.seed(m_seed);
```

What is needed here is a code that initializes the vector of random number generators with deterministic seed values, that differ however from observation to observation. This is a tricky task, as different seed values should always lead to statistically different datasets.

If one would for example simply increment seed by 1 from one observation to the next, a ctobssim run for which the seed parameter is incremented by one with respect to another run would virtually produce the same data (only two observations would be different).

I guess a pretty clean method would be to use a random number generator to assign seed values for all random number generators, avoiding that a given seed value is used twice (if a given seed value would be used twice we would have two simulated observations with an identical set of events). Something like:

```
GRan master(m_seed);
std::vector<unsigned long long int> seed;
for (int i = 0; i < m_obs.size(); ++i) {
  unsigned long long int new_seed;
  do {
    new_seed = (unsigned long long int)(master * 1.0e20);
    bool repeat = false;
    for (int j = 0; j < seed.size(); ++j) {
      if (new_seed == seed[j]) {
        repeat = true;
        break;
```

```
    }
   }
  } while(repeat);
  seed.push_back(new_seed);
}
```

Now we should have a set of seed values that is fully deterministic, but that guarantees at the same time that each observation is simulated with a different seed value. In a subsequent loop, we can initialise random number generators with these seed values and push them on the vector, e.g.

```
m_ran.push_back(GRan(seed[i]));
```

This code should be okay, as the GRan copy operator copies the seed value, but it should be checked that the random number generator in the vector produces the same sequence of numbers as the random number generator that was pushed.

Finally, we have to modify the ctobssim::run method to explicitly select a random number generator from the vector. For this purpose, one probably has to add the random number generator as parameter to the ctobssim::simulate_source and ctobssim::simulate_background methods, and modify these routines so that the use the random number generator passed in the argument.

**Related issues:**

| | | |
|---|---|---|
| Blocks ctools - Action # 355: Parallelize simulation loop | **Closed** | **07/30/2012** |

### History

**#1 - 07/20/2012 11:06 PM - Knödlseder Jürgen**

*- Description updated*

*- Estimated time changed from 8.00 to 24.00*

*- Remaining (hours) changed from 8.0 to 24.0*

**#2 - 07/30/2012 10:44 AM - Anonymous**

*- Status changed from New to In Progress*

*- Start date set to 07/30/2012*

**#3 - 07/30/2012 03:03 PM - Anonymous**

*- Status changed from In Progress to Resolved*

*- % Done changed from 0 to 100*

Juste for the new_seed:

new_seed = (unsigned long long int)(master.int64() * 1.0e20);

1.0e20 is too big and with the multiplication, new_seed is always at 0. The random number is too big so I just multiplicate by 1.0e2.

**#4 - 09/14/2012 11:48 PM - Knödlseder Jürgen**

*- Status changed from Resolved to Closed*

*- Remaining (hours) changed from 24.0 to 0.0*