We are happy to announce rTRNG 4.23.1-1, and share its journey with `valgrind`, Docker and GitHub Actions.

rTRNG is an open source package for advanced parallel Random Number Generation in **R**. It relies on TRNG (Tina's Random Number Generator), a state-of-the-art C++ pseudo-random number generator library for sequential and parallel Monte Carlo simulations.

The package has been happily living on CRAN since its first release, and we planned a maintenance release to include the latest developments of the upstream TRNG C++ library. As part of a new submission, the CRAN Team kindly pointed us to some ***valgrind issues*** reported by the CRAN package checks. `valgrind` provides tools for detecting possible memory allocation and access issues, which are very relevant to R packages including C++ code.

Despite having little experience with `valgrind`, we could address the issues by going down a path that is pretty common and very effective while investigating / fixing problems or bugs in general. We share the details of our investigations below, and how we ended up with a Docker-based solution to include `valgrind` checks in our Continuous Integration (CI) GitHub Actions workflows.

Reproduce the issue

For cases where errors are reported on a system you don't have access to, it is fundamental to have an accessible environment where the errors can be reproduced, investigated and, ultimately, fixed. This might not be trivial for `valgrind` issues reported by the CRAN checks. Luckily, and mainly inspired by a blog post by Brodie Gaslam, we could call Docker to the rescue. Using Winston Chang's `wch1/r-debug` image, `valgrind` checks and issues are reproduced as follows[1]:

```
mkdir -p valgrind-check
docker run --rm -v $(pwd)/valgrind-check:/tmp/valgrind-check wch1/r-
debug bash -c ' \
  wget https://cran.r-project.org/src/contrib/Archive/rTRNG/rTRNG_4.20-1.tar.gz \
  && RDvalgrind -e "install.packages(\"remotes\"); \
    remotes::install_deps(\"rTRNG_4.20-1.tar.gz\", dependencies =
TRUE)" \
  && RDvalgrind -d "valgrind --track-origins=yes" CMD check \
    -o /tmp/valgrind-check --use-valgrind --no-stop-on-test-error \
    rTRNG_4.20-1.tar.gz'
```

Minimal example

Working with minimal examples is often the key to an effective and efficient investigation. Running the extensive set of package checks with `valgrind` easily takes 15-20 minutes, and is not quite a minimal workable example. Therefore, we carefully isolated a specific test causing the issue (one of the many reported). From there, we could extract a few relevant and minimal examples that could be run in an (interactive) R session with `valgrind` in a Docker container. More details can be found in the GitHub issue miraisolutions/rTRNG#16.

Isolate the root cause

With the focus on individual issues reproduced using minimal examples, it was pretty clear where to locate the root cause, ultimately found in C++ code included from the upstream TRNG library.

Playground for fixes

The same minimal example proved very useful in testing a quick fix attempt to confirm our understanding of the root cause. Running the full check again made sure no other issues remained to be investigated. We could then reach out to Heiko Bauke, the maintainer of TRNG, to report the issue and coordinate on a proper fix.

Patch automation and CI

In order to ensure an effective and quick cycle for including and testing upstream fixes, we pursued two strategies:

- Automate the application of a patch to TRNG as part of our package.
- Include `valgrind` checks in our CI workflows. This was straight-forward given the out-of-the-box support for running Docker containers in GitHub Actions, where we could run `valgrind` checks from an Actions workflow.

Having all this automation in place made it very straight-forward to include, in a controlled way, the upstream patch of TRNG released by Heiko in record-time. Moreover, such setup allowed to eventually have a clean and informative commit history for rTRNG on GitHub (see miraisolutions/rTRNG#21):

- CI failure revealing the original problem.
- Assess the planned upstream fix as patch inside rTRNG.
- Smooth and confident inclusion of the properly-released upstream patch.

Happy CRAN submission

Last but not least, this put us in position for a confident CRAN release, leveraging CI to demonstrate the fix of the `valgrind` issues to the CRAN Team upon submission. CI will play an important role for future releases too, where we can capture possible new issues ahead of a CRAN submission, making life easier for all involved.