Recent Changes in HMSC

https://github.com/hmsc-r/HMSC/

Jari Oksanen 19 Aug 2022

Hmsc 3.0-13

CRAN Release Aug 11, 2022

- Recover from bad updaters
- Add more chains to old models
- More aggressive parallelization in pcomputePredictedValues and predict
- Improved support for spatial models defined *via* distances
- Read complete news with **R** command `news(package=“Hmsc”)`
- Always when we have courses, students find hidden bugs: next release expected after this course

Three Kind of Errors

- Evil Errors: Function runs smoothly and gives results – but the results are wrong!
- Good Errors: We inspect the data before users do something wrong and warn about their mistakes
  - We try to be friendly and informative (but may fail in our attempt)
- Nasty Errors: That come from the abyss of **R** and are incomprehensible
  - We try to catch these and make to “Good Errors” – to serve and protect people

An Example of Nasty Error

Bad Updater

- Nasty error: absolutely cryptic error messages and sampling is trashed

```
Error in chol.default(iV) :
  the leading minor of order 3 is not positive definite
```

- Mathematically should not happen, but can occur in numerical computation
- Not deterministic but can occur after days or weeks of successful calculation
- From latest CRAN release, these errors are caught and handled – almost
- If an updater fails, keep the old values of those parameters and try again on the next iteration

What to do after updater failures?

- If any failures, the numbers are printed for each updater and chain
- If there are not many failures, it is safe to use the result
  - What is many after trying Chains × (Samples × Thin + ~~Transient~~) times?
- If there are failures only in some chains, these can be removed

```
model$postList[[2]] <- NULL
```

- If all fails, reconsider model specification
  - Recently found out that updateGammaEta propagates errors elsewhere
  - Using `sampleHmsc(…, updater = list(GammaEta = FALSE))` may help

How to report an error

- Errors may still appear: Otso came across one case on Sunday morning
- Report in https://github.com/hmsc-r/HMSC/issues
- Crucial to copy the exact error message in your message
- After error: `traceback()`, copy and paste the full output in your message
    - Unfortunately traceback is useless in parallel processing
- Reproducible examples are valuable: it is difficult to fix things if you do not know what is broken
- use `set.seed()` to reproduce random sequences

Need More Samples? Add Chains!

- New support function `c()` combines models by adding their chains

  ```
  m1 <- sampleMcmc(…, nChains=2)
  m2 <- sampleMcmc(…, nChains=2)
  m4 <- c(m1, m2) # now 4 chains
  ```

- Tries to check that the models are similar and warns if detects differences – but may still combine

- Do not start different models from the same random seed: these duplicate data, but do not add *new* samples

pcomputePredictedValues

- Cross validation runs `sampleMcmc` for each fold, and this can be **very** slow
- Old `computePredictedValues` can run each chain in parallel, but folds are run serially
- New `pcomputePredictedValues` can run nChains × nFolds parallel chains
  - Does not preschedule, but starts new chain when any CPU becomes free
- We plan to ditch old function, but now both are provided
- Species cross-validation is very slow with `mcmcStep`: but `predict` can be parallelized over posterior samples

A Word about Speed

- Hmsc spends most of its time in matrix algebra, especially in matrix inversions
    - Most of time Hmsc is in function `chol()` doing Cholesky decomposition
- R does matrix algebra in BLAS (Basic Linear Algebra Subprograms) and ships with an internal “Reference BLAS” – which is not built for speed
- BLAS can be optimized for **your** computer architecture to use parallel processing and vectorized instructions and these can give **enormous** speed-up
    - OpenBLAS, Intel MKL (Math Kernel Library), Apple Accelerate Framework
- How to take faster BLAS in use depends on your system and hardware: study documentation in CRAN

Spatial Models & Distances

- Spatial models work *via* distances, but still spatial coordinates are often needed
- Support for distance matrices is improved, and in most cases models are identical with spatial coordinates and their distances as an input
    - If you find differences or glitches, please report – some cases can be fixed (not all)
- Gaussian Predictive Process can be run only with spatial coordinates
- `constructGradient` allows now user-set or infinite coordinates: infinite means that ignore spatial location
- Hmsc does work with Infinite distances (or Infinite spatial coordinates – almost): Spatial dependence over ∞ distance is 0

Little-known features: getCall & update

- **R** function `getCall` finds the call of Hmsc model or random levels:

```
> getCall(m)
Hmsc(Y = TD$Y, XFormula = ~x1 + x2, XData = TD$X, studyDesign = TD$studyDesign,
    ranLevels = list(sample = TD$rL2, plot = TD$rL1), TrFormula = ~T1 +
        T2, TrData = TD$Tr, phyloTree = TD$phy, distr = c("probit"))
> getCall(m$ranLevels[[1]])
HmscRandomLevel(units = TD$studyDesign$sample)
```

- These can be `update`d:

```
> m <- update(m, XFormula = ~ x1, distr = "normal")
> getCall(m)
Hmsc(Y = TD$Y, XFormula = ~x1, XData = TD$X, studyDesign = TD$studyDesign,
    ranLevels = list(sample = TD$rL2, plot = TD$rL1), TrFormula = ~T1 +
        T2, TrData = TD$Tr, phyloTree = TD$phy, distr = "normal")
```