Package: microscoping (via r-universe)

September 10, 2024

Title Preliminary feasibility for CKMR

Author Mark V. Bravington <markb2@summerinsouth.net>

Description Rough-as-guts laugh-test design calculations for CKMR on fish

Maintainer Mark V. Bravington <markb2@summerinsouth.net>

Imports stats, utils, mgcv, mvbutils (>= 2.9), atease, offarray (>= 2.0)

License GPL-2

NeedsCompilation no

Version 1.0.71

Repository https://markbravington.r-universe.dev

RemoteUrl https://github.com/markbravington/microscoping

RemoteRef HEAD

RemoteSha fcaac9ca748d7354169fe3de75d871aa056a8331

Contents

`microscoping-package` *Rough feasiibility calculations for CKMR on typical fish*

Description

If you don't know what CKMR is, stop reading now.

The main function `ckmr_laugh_test` (qv) predicts how many kin-pairs of various types might be expected for Your species of fish, given Your proposed CKMR sampling scheme and Your guess as to what the current stock size etc might be. The clue is in the name `ckmr_laugh_test` (qv). It is *deliberately* crude because you are not supposed to really trust the results, and certainly not to try fine-tuning them by microtweaking the sampling scheme, because the model assumptions are bound to be somewhat wrong.

I wrote `ckmr_laugh_test` for private use by me & a few chosen colleagues, but it seems to have escaped into the wider world, and predictably to have been used more seriously than I ever intended. Be aware that AFAIK `ckmr_laugh_test` has *never been simulation-tested* and might have substantial errors (I hope not). I don't feel it's my job to test it thoroughly, since I never meant it for general use! Please **feel free** to do that simulation-testing yourself, e.g. using Shane Baylis' `fishSim` package or Eric Anderson's `ckmrSim` package (and let me know the result); it would be quite helpful..

There is a sex-specific version `clt2`, which I reluctantly added for species with unignorably strong sexual dimorphism (e.g. swordfish) where `ckmr_laugh_test` is really inadequate and cannot be tricked into giving even a semi-reliable answer. `clt2` is harder to use, and even less tested.

`ckmr_laugh_test` *Rough-as-guts close-kin scoping*

Description

How many kin-pairs might be expected in an age-structured population under a given sampling "design"? You have to specify a few things about the species and about the sampling design. So the idea is that you play around with the "sampling design" to see how samples of which type (age, year) you might have to do to get a useful number of kin-pairs. If that looks promising, you should go off and do a more formal statistical design. `ckmr_laugh_test` is the non-sexually-dimorphic version, and is substantially easier to set up; if you *really* need a two-sex version because of major differences in e.g. catchability or fecundity or abundance, then look at `clt2`. Most of this documentation applies to both. Note also the **health warnings** in `package?microscoping`.

To run `ckmr_laugh_test`, you will need to start from an existing age-structured stock assesssment. If you don't have such a thing, you'll need a different approach— "watch this space". Also, the assumptions behind `ckmr_laugh_test` are deliberately crude (see below for details). They might be too crude for your problem; in that case, you'd need to go straight to a more formal design anyway, based on a bespoke CKMR model.

Even if your `ckmr_laugh_test` results suggest that you can get plenty of kin-pairs, it does NOT follow that you will actually be able to estimate everything you really really want, because the spread of samples and the ability to accurately measure covariates (eg age) can have a big impact. However, if after playing around with `ckmr_laugh_test` you decide that you *can't* get enough kin pairs, then you are simply dead in the water. And again: if you can, then it's time to do that more detailed and formal design.

The following types of kin-pairs are reported: parent-offspring POPs, separated according to whether the offspring are "adult" or not at time of sampling; half-sibling HSPs (juve-juve only) *except* from the same cohort; and grandparent-grandchild GGPs (juve-juve). The most useful are generally `POP_offJ` (i.e. where offspring is still juvenile) and `HSP_JJ`. When interpreting the numbers, bear in mind an implicit assumption that age is known fairly accurately for juveniles, at least. Offspring-age is really important for POPs (and HSPs); parent-age less so.

There are two reasons for restricting HSPs to JJ. The first is that HSPs and GGPs cannot be distinguished genetically, which unfortunately tends to make HSPs between "adults" not useful; that's why `HSP_AA` etc is not reported, because they might actually be GGPs which just kinda mess up the signal for population dynamics. (There is some information, but I'm not sure how useful it is.) Hence the restriction to juve-juve HSPs, which are certainly useful. You are required to specify the `MAX_JUVE_AGE` up to which animals still count as "juvenile". Sometimes, raising `MAX_JUVE_AGE`, so as to include some age-classes with very low individual fecundity, can increase the number of HSPs a lot. `ckmr_laugh_test` reports the number of GGPs, so you just need to make sure that the ratio of GGPs to HSPs stays small enough to ignore (eg 1%). The second reason for restricting HSPs to JJ, is that HSPs are not useful when the offspring-age is very imprecise. If that is already happening before sexual maturity, you should push down `MAX_JUVE_AGE` to a lower value.

There are two reasons why `POP_offA` *may* not be useful (ie why you may just want to concentrate on `POP_offJ`). The first is if age estimates are imprecise for "adults"; if the birth-date of an offspring is vague, then that POP is just not very informative for CKMR purposes. The second reason is that, even if adult age is accurate, you may not want to push your pop dyn model back far enough in time to cover all potential adult *offspring*. At some point it becomes more trouble than it's worth to push the model back into the past in order to accommodate diminishing returns of information from a handful of distant-past POPs.

The parameter `MAX_JUVE_AGE` is thus doing "double duty" here. First, it lets you restrict comparisons so that GGPs are rare compared to HSPs; that's all about the maturity schedule of the species. Second, it lets you separate between young animals with fairly precise age estimates (based on body length, and/or otoliths, and/or vertebral sections), and older ones where age may be vague; that's all about ageing-error, and may or may not relate to sexual maturity. So the two meanings of `MAX_JUVE_AGE` may be different. If that seems really important to you, then I suppose you can always run `ckmr_laugh_test` twice with different `MAX_JUVE_AGE`, and mix-and-match the results, .

The assumptions are deliberately crude and inflexible, because the idea is that you should do a **proper** design *if-and-only-if* this crude calculation looks promising. For the same reason, there's no attempt to automatically "optimize" the sample sizes in this code to get the overall numbers big, although that is entirely do-able in a full design; with `ckmr_laugh_test`, you just have to poke around manually and try different things (eg age-breakdown especially, and duration of survey), which hopefully also gives you some insight into trade-offs between different types of sample. If the assumptions are just too crude for you, then it might be possible to modify this code simply enough going to full design; try it yourself, or by all means ask me, though I may well not have time to help. Anyway, the assumptions are:

- steady-state population;
- males and females have identical reproductive and demographic parameters;
- age, and nothing but age, affects fecundity and selectivity.

Plus group: `ckmr_laugh_test` and `clt2` are not really intended for stocks with a large proportion of fecundity sitting in the plus-group. There's no perfect way to make the plus-group consistent with both (i) steady-state pop dyn, and (ii) "constant" mortality rate (ie equal to the pre-plus-group-age mortality rate, inferred from the log-linear regression). You gotta pick one. So if `extraplus=FALSE` (the default), I calculate a separate mortality rate for the plus-group that keeps the input plus-group numbers in balance with the other age classes. However, this may not reflect true mortality (e.g. if the plus-group is big because it contains a residue of more-lightly-exploited days) and that will distort HSP probabilities. If instead `extraplus=TRUE`, the actual numbers supplied for the plus-group are replaced by an extrapolation based on the pre-plus-group age classes, with mortality equal pre- and post-plus-group. That might be more realistic mortalitywise, but could misfire in terms of the actual TRO that a CKMR study would encounter. I added the `extraplus` option so that it's not My fault when Your numbers go wrong... but with some trepidation. For this laugh-test stuff, I really want to *avoid* adding options, because it will only make people believe in results that they shouldn't (the more effort You put in, the more You are inclined believe the results, but for no good reason!).

Code notes: The code starts with some "housekeeping" to expand out the plus-group, smooth the age distribution, make sure enough years are "simulated", etc; I recommend ignoring it until/unless you can't. The CKMR action starts around the line `nfata <- ....` The main part of CKMR is the probability calculations for different kinships depending on the potential covariates of each pair in a comparison.; something like this is always needed for real data, as well as for scoping. Then, specifically for design/scoping, there is a calculation of the likely number of pairwise comparisons of each type, from the line starting `ncomps <- ...` onwards. With real data, you'd know the actual number of comparisons.

The code uses the **offarray** package to allow arrays where the first index is not 1; in this case, ages start at 0. Generally, the code is about as straightforward as it could be, but does make use of one novel feature: `offarray::autoloop` (qv). This "loops" over whatever named parameter ranges you tell it, evaluating an expression that you pass it for each combination of the parameters, and putting the results into one or more arrays. `autoloop` is like writing for-loops without the "for", but is **far** quicker than R's built-in for-loops, and saves you having to work out how to vectorize things according to R's own rules, which requires hideous mental contortions even when possible, which isn't always. `offarray::sumover` is also used, and it does exactly as its name suggests.

Probability calculations: The code shows the basics of all POP, HSP, and GGP calculations. It only deals with females- the final answers are obtained by scaling up- and lacks an explicit time dimension because of the steady-state assumption, but the changes required for application to real data are not enormous. Mainly, the TROF would need to be a vector across years, rather than a scalar (which corresponds to steady-state). For really real data, though, length *as well as* age ought to be considered, certainly for HSPs.

Steady state: Steady-state obviously needs to have applied over a generation or two. Hence we need to eliminate cohort effects, so the input numbers-at-age (eg from a recent stock assessment) are first smoothed by fitting a mildly-nonlinear "catch curve"; from that, we can also estimate age-specific steady-state survival. The "catch curve esque" model for smoothing numbers at age is this:

```
  glm( nata ~ age + sqrt( age), family=Tweedie( p=1.7, link="log"), ...
```

and the final input numbers-at-age is disregarded, in case it's really a plus-group. If that model leads to any estimated annual survivals above 1, then a purely (log-) linear fit is used instead. The point of the Tweedie is to be somewhere between "noise purely due to cohort variability" (which tends to have var propto mean-squared, ie Tweedie(2)) and "noise purely due to counting-type error" (which has var propto mean, ie Tweedie(1)). Not a big deal, just a minor improvement.

Sex differences: It's perfectly possible to amend the algorithm so that the two sexes can have different reproductive and/or sampling schedules by age. But in the interests of simplicity and of not making the results seem more reliable than they really are, I haven't done so here. When a species really does have strong sex differences, I suggest running the code twice, once using the schedules for females and once using the schedules for males, and averaging the results. It's crude, it's wrong— but the whole point of `ckmr_laugh_test` is to be simple, crude, and unlikely to be mistaken for "correct". If the "twice and average" just seems **too** crude— eg if the sex ratio of adult samples is likely to be strongly sex-biased— then you really need to do a proper CKMR design.

FWIW: I did consider including an example to show how sex can be handled a bit better than "twice and average", by running `ckmr_laugh_test` several times with adjusted sex-specific sample sizes and sex-specific age schedules, and then combining the results. But in the end, I decided not to, because (i) the process was getting complicated to explain, and (ii) there is in any case no really satisfactory solution without making *all* the `ckmr_laugh_test` code sex-specific (which actually wouldn't be that difficult). The issue is that the sex of potential *parents* does matter in CKMR (including unobserved parents in HSPs, and the unobserved "middle generation" of GGPs), but the sex of *offspring* usually doesn't. That means that a proper sex-specific version should split the sample sizes of potential parents by sex, but not the sample sizes of offspring. However, the code as currently written allows "juveniles" (i.e. age up to `MAX_JUVE_AGE`) to be parents as well as offspring— since `MAX_JUVE_AGE` might just be a filter to discriminate "ageable" age-classes from "unageable" ones, rather than an indication of maturity per se. So there's no obviously clean way to split the sample sizes. Of course all this could be handled properly in fully-sex-structured code, but not really within the current framework.

Usage

```
ckmr_laugh_test(nata, wata, pmatata, samp_pata, samp_ny,
    MAX_JUVE_AGE, MIN_CATCH_CURVE_AGE=0, beqy_POP=FALSE,
    etriv=0.01, extraplus=FALSE, want_gory_details=FALSE)
```

Arguments

`nata`	numbers-at-age in population (both sexes combined). Starts at age 0. Final age-class is assumed to be a plus-group and is *not* used (because if things are steady-state, you can work out what's in the plus-group just from the non-plus numbers).
`wata, pmatata`	weight-at-age and proportion-mature-at-age (starts age 0). Fec-at-age is assumed to be `wata*pmatata`.
`samp_pata`	proportion of sample by age class (starts age 0).
`samp_ny`	total numbers of samples per year. In each year, this is pro-rated by `samp_pata` to determine numbers-sampled-by-age-and-year.
`MAX_JUVE_AGE`	what's the greatest age that will still be treated as "juvenile" ie included in HSP/GGP comparisons, and as "potential offspring" in adult/juve POP comparisons. (Adult-adult POP comparisons still happen, but are summarized separately.)
`MIN_CATCH_CURVE_AGE`	what age to start fitting a log-linear "catch curve" regression? Use this to skip earliest ages if their mortality is verrry high. Be warned you could get into

trouble with overstretching steady-state assumptions here. Rough-as-guts, like I said...

`beqy_POP`
: should it allow comparisons between "adults" caught in year Y and "juves" born in the same year (ie B==Y) ? Normally this is a bad idea because adults caught *during* a spawning season have not had a fair go; however, if you are sure that "adult" fishing happens after spawning (within the cycle of whatever "year" you are using) then you could set it to TRUE. The real reason for having it at all, is to check that `ckmr_laugh_test` is matching the "POP cartoon", which requires adult sampling shortly after juve birth; see **Examples**.

`etriv`
: Probably leave this alone. It controls how far out to expand the plus group, ie the AMAX at which all animals are forced to die. Setting `etriv=0.01` means that only 1% of the total fecundity will be "lost" due to animals being forced to die. Mainly there to avoid calculations taking forever with 10000 age classes... if things take too long, try increasing `etriv`.

`extraplus`
: It should not matter much what you choose here— if it does, this simple laugh-test code might not be right for you (see argument `etriv` too). Anyway:if TRUE, the input plus-group (represented by the final age-class in the input) will be ignored, and will be replaced by extrapolation from the pre-plus-group based on estimated survival from `Aplus-2` to `Aplus-1`. If FALSE, the input plus-group will be assigned an annual survival (from `Aplus-1` onwards) so that it smoothly matches the fitted `N[Aplus-1]`. Neither option is correct; nothing ever could be for an equilibrium plus-group out to age Infinity.

`want_gory_details`
: if TRUE then the usual result will acquire an attribute `gory` which is the environment where everything was evaluated— all the intermediate calculands are kept in there. It might be big. `ls(attr(<result>,"gory"))`— or `ls( <result>@gory)` if package **atease** is loaded— will show what's available, e.g. covariate-specific probabilities and number-of-comps. I've used it for debugging, and for simulation-checking the code.

Value

A numeric vector with fairly self-explanatory elements "POP_offJ", "POP_offA, "HSP_JJ", "GGP_JJ", plus "nsamp_J" and "nsamp_A" which are totals across all years. For POPs, "offA"/"offJ" describes whether-or-not offspring is above MAX_JUVE_AGE. Same-cohort HSPs and GGPs are excluded. See DESCRIPTION. There are also attributes `call` (obvious) and `params`. The latter is a list with the actual values of all parameters; by default, it does not print (it has class `nullprint`) since it is mostly clutter because you know what they were since you called it, but you can get the values if you need them via eg `attr( result, "params")$nata`, or show them all via `unclass( attr( result, "params"))`. If you have library `atease` loaded, then you can just type `result@params$nata` etc.

Examples

```
N <- c( 1000, 780, 690,
    420, 400, 282,
    210, 150, 100,
    900)
W <- c( 0.1, 1, 2,
    3, 3.5, 4,
    4.5, 5, 5.2,
    5.5)
Pmat <- c( 0, 0.1, 0.4,
    0.8, 1, 1,
    1, 1, 1, 1)
Sampa <- c( 0.05, 0.2, 0.7,
    1, 1, 1,
    0.8, 0.6, 0.4,
    0.2)
Sampy <- c( 100, 500, 100)
MJA <- 3
testio <- ckmr_laugh_test( nata=N, wata=W, pmatata=Pmat, MAX_JUVE_AGE=MJA, samp_p=Sampa, samp_ny=Sampy)
#
## Cartoon:
N <- c( 4e6, 400, 0)
#  ie very high mortality, so in effect only one year of adulthood.
#  Needs one extra "plus-group" age-class to keep machinery happy,
#  but the number in that final class is not used directly.
#
W <- c( 1, 1, 1)
Pmat <- c( 0, 1, 1)
Sampa <- c( 1, 1, 0)
Sampy <- c( 10 * sqrt(400)) # one year, equally split between juves and adults
# Should give 50 POPs...
MJA <- 0 # 0-group are juves; 1+ are adult
cartoonio <- ckmr_laugh_test( nata=N, wata=W, pmatata=Pmat, MAX_JUVE_AGE=MJA,
    samp_pata=Sampa, samp_ny=Sampy,
    beqy_POP=TRUE) # phew, POP_offJ == 50 !
```

---

clt2 *Sex-aware close-kin scoping*

Description

Like `ckmr_laugh_test` (qv) but everything is split by sex, for better handling of strongly dimorphic species. This two-sex version `clt2` entails considerably more tedium when setting up the input data, and runs about eight times slower, so stick to the single-sex version `ckmr_laugh_test` if you can bear to. It's all very rough anyway. And I haven't really tested it. Feel free to do so by simulation (not sarcastic!).

Usage

```
clt2( n_sa, w_sa, pmat_sa, nsamp_sya= NULL, samp_p_sa,
  nsamp_y, MAX_JUVE_AGE, MIN_CATCH_CURVE_AGE= 0, beqy_POP= FALSE,
  etriv= 0.01, extraplus= FALSE, want_gory_details= FALSE,
  .BLOCKSIZE= 1e4, .REPORTSEC= 5)
```

Arguments

`n_sa`, `w_sa`, `pmat_sa`, `samp_p_sa`
: all as per `ckmr_laugh_test`, except now sex-specific; so they should each be a two-row matrix or `offarray`, one row for each sex. Rownames, if present, must be "F" and "M" in either order, but must be consistent. With `offarray` arguments, I *think* that AGES *must* start at 0, but I'm not sure this is checked...

`nsamp_sya`
: Sample sizes by sex, year, age (3D `offarray`, or possibly normal `array`). A value of NULL, the default, means it will be constructed from `samp_p_sa[S,A]*nsamp_y[Y]` in the obvious notation.

`nsamp_y`, `MAX_JUVE_AGE`, `MIN_CATCH_CURVE_AGE`, `beqy_POP`, `etriv`, `extraplus`, `want_gory_details`
: as per `ckmr_laugh_test`

`.BLOCKSIZE`, `.REPORTSEC`
: passed to `autoloop`. If the main loop is slow, you can set these to get a progress update every `.REPORTSEC` seconds. I wouldn't recommend messing with `.BLOCKSIZE` but it exists for a reason— and so does the `autoloop` documentation ;).

Value

Mostly as per `ckmr_laugh_test`. Results are not disaggregated by sex (though the computations internally are sex-specific) because the focus is on total kin-pairs. Attributes `call`, `params`, and `nsamp_sya` are attached. The last two won't display in full if `mvbutils` has been explicitly loaded, because they are class `nullprint`. That is entirely deliberate. If you want to see them, use `unclass` but that will zap their offarray-ness too; safer is eg

`res <- clt2(...)`

`nsampo <- res@nsamp_sya # requires library( atease); already imported`

`oldClass( nsampo) <- oldClass( nsampo)[-1]`

`# can now print nsampo`

Examples

```
# This really SHOULD have an example
# but it doesn't yet
```

Index