TMBO FAQ:
FACETIOUS ARTIFICIAL QUESTIONS (V2.4.30)

Well, not all of these are facetious, in fact. Think of this as a Socratic dialogue, where the imaginary italic Questioner is asking the actual upright Me (not Socrates, dammit! just the TMBO author) some Questions; so "you" in a Question actually means Me, whereas "you" in an Answer actually means some generalized You which no doubt does not apply to you personally, i.e. the reader. And conversely. Got all that? Righto. NB I have added some of what I call "humour"; this may trigger some people.

Q1. *Isn't TMB wonderful? [*Not a facetious question!*]*

A1. Yes! [Not a facetious answer!] It really is amazing! It lets me write & fit exactly the type of statistical models that I like fitting: fairly complicated underlying dynamics, with lots of random effects, but nothing truly pathological or nonlinear or multimodal or needful of MCMC.

Q2. *So you enjoy using TMB, then?*

A2. Well, actually... no. Debugging seems to be an *absolute horror* (both for compilation, and during runtime), the rules for coding get pretty weird in places, documentation is somewhat patchy, and the differences with R (index offsets, for-loop syntax, ...) keep leading me into stupid mistakes. Those mistakes are my own fault for sure, but let's just realistically accept that I am never going to become a better person. Grateful though I am for TMB's existence, and mindful that some of those issues are just unavoidable side-effects of its templated-C++ nature: that doesn't mean the issues don't exist! Anyway, the only ones I can personally do much about are the first and fourth ones: hence TMBO.

Q3. *Do you have a deep understanding of how TMB works?*

A3. Nope. Well, actually I do understand the maths, and pretty much how AD works. But all the C++ wizardry of templated code and CppAD and Eigen and god-knows-what? Absolutely No Idea. And that's fine by me.

Q4. *I can't see the need for TMBO. I am a straightforward person who writes straightforward code, and I don't find it hard to use 0's in TMB, 1's in R, and completely different indices in the real dataset. It's simply a matter of basic self-discipline. Also, debugging aids are only a crutch for the morally degenerate; simply write correct code to begin with. Cold showers and a light daily flogging are good practice, too.*

A4. That's nice, dear! Well done, have a lovely biscuit. Also, in today's shock news: scientists discover that *people are different*...

Indeed, many people will not need, or will not want, to use TMBO, and that's fine. But my own perspective is that:

(i) The best chance of me understanding my own code, whenever I have to come back to it after months or years away, is if the code is fairly succinct (hence eg `FOR`) and if there are as few unnecessary mental translation gymnastics as possible (hence index offsets: not needing to remember whether to subtract 1987 or should it be 1986 from all years, etc). Same goes *a fortiori* for other people's code, if I am ever forced to look at that...

(ii) *You* might not have to code complicated stuff, but *I* sometimes do.

(iii) Bugs in TMB code are a *nightmare* to deal with, much worse than FreePascal (10/10), R (9/10 *iff* you use my `debug` package, about 4/10 otherwise), or even C (6/10), although better than "makefiles" (0/10). Thus, anything which reduces the number of TMB coding errors that I make, is very useful.

Q5. *Why is there "SystemRequirements: C++20" in DESCRIPTION? Is that a problem?*

A5. C++20 is only needed during the standalone pre-processing, for the `__VA_OPT__` macro syntax (which could be avoided, but at the cost of substantially more work for *me*). C++20 seems to be available in the `gcc` and `Clang` versions recommended for R>4.3, so it shouldn't ever be a limitation. Also note that the "main" compilation, which happens *after* the first standalone pre-processing stage, does *not* use C++20: just R's C++17 default. That's a good thing because Eigen or RcppEigen or something *does not compile* under C++20[1]!

Q6. *Is there a speed penalty when using offset indices?*

A6. I highly doubt it's noticeable. The C++ compiler should optimize away almost all of the horrid-looking macro expansions, at a guess leaving just one extra L1 cache access and one extra integer subtraction per dimension per lookup; and modern processors are so pipelined that even the subtraction may be cost-free. However, I'm too lazy to test speed. My main concern is the vast amount of *my* time saved by simplifying the TMB code I have to write and debug.

OOB-checks will slow things down a lot, so turn them off when actually fitting to data.

FWIW I suspect there are considerable speed savings possible in TMB code with heavily-nested loops (e.g. in CKMR, and age-structured population dynamics), and that I might eventually be able to auto-optimize a lot of that via extensions to TMBO (as I did once before in FreePascal)— but that's entirely another story.

Q7. *Could I run a completely normal TMB file thru `TMBO`? And can I mix normal TMB variables with `VECTORO` etc?*

A7. Yes. TMBO syntax just extends regular TMB syntax; there's no *requirement* to use `VECTORO` etc. TMBO variables *are* normal TMB variables.

Q8. *Can I do OOB-checks on non-TMBO variables, eg ones I declare with `vector<int>`?*

A8. Nope.

[1] I spent quite a long time trying to enforce C++20 in `tools:::.shlib_internal`, which is what TMB calls for compilation. It was very and unnecessarily difficult to do, and/because R's "documentation" is pretty dreadful on this sort of thing. Anyway, after all that effort, it didn't work anyway...

Q9. *Remind me again— how do I turn on OOB-checking?*

A9. A good start is to read the documentation... since I'm in a benevolent mood, let me point you to `?compile`.

Q10. *Harrrumph! I turned on OOB-checks and I* still *got a crashy error. What have you got to say about* that*?*

A10. Well, that's not overly generous with information, but at a guess: you copied some code direct from R, where you had eg `myvec[i]`, and forgot to change the square-brackets into parentheses[2]. Square brackets in C++ should be mightily avoided[3], but the otherwise-monstrously-pedantic compiler will happily let you shoot [yourself] in the [foot]. Harrrumph yerself.

Q11. *TMBO sounds like it would help me a lot with debugging, but I have a sh**load of existing vanilla R/TMB code, where indices are 1-based in R but 0-based in TMB, and I don't want to change it all just yet. Can I keep things that way while starting my conversion journey?*

A11. Yes. As of v2.4.30, you should just need to change your TMB declarations, but no code. For new variables declared on the TMB side via eg `vector<Type> stuff(30)` change to `VECTORZ( Type, stuff, 30)`. The indices of `stuff` will start at 0 and end at 29 in C, so you don't need to change your "classic" `for`-loops in C. And if you `REPORT(stuff)` then it will be a normal R vector (not an `offarray`) so it have elements from 1 to 30.

The original version of this FAQ suggested that you just redeclare things as eg `VECTORO( stuff, 0, 29)`. That's true, but the problems are that (i) you have to remember to adjust the size, and (ii) if you `REPORT(stuff)` then you will have an `offarray` in R with 0-based indexing, which you might not want.

For `DATA_` and `PARAMETER_` variables that come from R, you *might* even get away with leaving their TMB declarations unchanged. But I think it's best to change their declarations to eg `DATA_IMATRIXZ`; that should have no implications for existing C code, but you can use all TMBO features such as `VECTORO_sameshapeas`.

Ultimately, of course, you would have to be stark raving mad to want to *keep* your code that way indefinitely; why on earth not have consistent indices??? But, as we discussed earlier, I am not your mother...

Q12. *Is there support for index-offsets and names on sparse matrices, or other more-esoteric TMB objects such as `DATA_ARRAY_INDICATOR`, `DATA_STRUCT`, ..?*

A12. No, though I might be able to add them if there's demand. I doubt that sparse-oriented code would want offsets (of course, you can still use sparse matrices alongside the TMBO variables). It's possible that offsets on `DATA_ARRAY_INDICATOR` *might* be useful, but I didn't bother yet since I have not used myself it and I don't know how people do use it in practice. `DATA_STRUCT` sounds a bit over-complicated to me. Note that TMBO does have

[2]Yes, of course I've made that mistake myself...

[3]Unless you really know what you're doing. I had to use some in the OOB code. But, as it says elsewhere in these FAQ: do as I say, not as I do...

`DATA_FACTOR1`, added only because it was trivially easy; again, I don't *know* how factors typically get used inside TMB, but I *guessed* that `DATA_FACTORO` would *not* be useful, so that's not in there (yet).

Q13. *I want to refer to a* `DIMRANGE` *or* `ICONSEQ` *or* `CHIND` *in normal code, rather than just inside a declaration or* `FOR` *loop. Can I?*

A13. Nope, but see next.

Q14. *Can I specify a subrange of a range, eg in a* `FOR`*-loop?*

A14. Not directly. But you can refer to `FIRSTEL(vec)` and `LASTEL(range_or_chind)` and `LASTEL( ar, 3)` as if they were regular variables, so for a subrange of `AGES` you can always resort to eg

```
FOR( a,FIRSTEL(AGES), LASTEL(AGES)-1)
```

However, for such things I usually prefer to have variables which define the ends of the range, and refer to those instead:

```
FOR( a, Amin, Aplus-1)
```

where `Amin` comes either from R via `DATA_INTEGER(Amin)`, or perhaps inside TMB from `int Amin=FIRSTEL(AGES)`. I usually go the R way, but of course it's then my responsibility to make sure `Amin` coincides with `FIRSTEL(AGES)`.

Q15. *Can I resize a* `VECTORO/MATRIX1` *etc* after *they've been declared?*

A15. Nope. So you do need to know their dimensions before you declare them, yes. Get it right first time.

Q16. *Do I have to have inner parentheses when declaring "manual" ranges, as in* `VECTORO(Type,smurfs,(14, 28))`*?*

A16. Yep.

Q17. *Can I have* lots *of inner parentheses, like this?* `FOR(i,((((14,28))))` *or* `VECTORO(int,nasty_nesty,(`

A17. Nope. Just one pair.

Q18. *What about in* `FOR` *loops? Is* `FOR(i,1,9)` *OK? How about* `FOR(i,(1,9))`*? Or* `FOR(i,((1,9)))`*?*

A18. Yep; yep, but why would you?; and nope, respectively.

Q19. *Why (not)?*

A19. Becoz macros. Enough already! Next questioner, please...

Q20. *I did something wrong and now have a logfile full of totally incomprehensible compiler error messages. What next?*

A20. Welcome to TMB... :/

But it might help to remember the two-phase nature of TMBO compilation. If you get an error during the first (preprocessing) pass, then it will probably happen very quickly, and you should rejoice! You will get a character vector return-value in R, containing the partly-preprocessed output with an error message(s) somewhere in the middle (so `.Last.value` might be handy). The preprocessor is not very fussy and will let thru code that is a complete turdfest, but it certainly doesn't like it if you supply the wrong number of parameters to

a macro; one easy way to do that, is to forget the first `Type` or `int` argument when you declare a variable via `VECTOR0` etc. Preprocessor errors are usually much easier to find and fix than errors in the second, main, compilation.

Also, carefully read the documentation of `TMBO::compile`, including the `flags` argument and the Value section. And see next FAQ. And good luck!

Q21. *Can I save those* dreadful *incomprehensible compiler error/warning/note messages into eg an R character vector, so that I can at least sift thru them after the fact, to pan for a nugget of sense amongst the spoil-heap of verbiage?*

A21. On windows, try setting the `flags` argument to `TMBO::compile`; read the doco! Maybe there's a Linux and Mac equivalent, dunno. Other than that, I don't know how. It would certainly be useful! The problem is that hardcore compilation in `TMB::compile()` is done by `tools:::.shlib_internal()` which calls `base::system()`, and does not bother saving its output. I had a fair go at hacking around that but so far without success. My guess is that different front-ends to R (RGui, rterm, VSCode, emacs, Rstudio) and platforms (Windows, Linux, Mac) might differ here. I use RGui and those messages *only* appear on-screen (unless I use a logfile... I should always use a logfile!), requiring very finicky mouse-based cut'n'paste, which is a PITA. It's certainly annoying. Anyway, see `flags`, like I said.

Note that preprocessing errors *are* automatically saved, as per previous; this FAQ is just about second-phase compiler errors.

Q22. *Will the macros in TMBO clash with other variables, and/or macros lurking in the bowels of RcppEigen etc?*

A22. I highly doubt it. The macros you use, ie those in "TMBO.h", disappear after the first preprocessor pass, replaced by others in "TMBO2.h". Those others (which you never see) mostly have deliberately odd names, usually starting with underscores and containing double-underscores and the letters "TMBO". (Though I haven't been completely systematic about disappearances and name-changing yet.) Just let me know if you find any problem, and I will obfuscate some more names (without changing the interface).

Q23. *Can I put double-underscores in my variable names, and also start and end the names with underscores? Becoz I gotta be me.*

A23. Don't. Just don't. Be "you" by all means, but somewhere else please.

Q24. *I tried to do something very very clever, and it didn't work. What now?*

A24. Don't try so hard to be clever, at least not with TMB and TMBO. Stick to what's in the examples. Google "Icarus". And do as I say, not as I do.

Q25. *I tried to do `fancycast(myar)(i,j)` and it didn't respect the indexing. Why not?*

A25. See previous question :/

Q26. *Is TMBO completely idiot-proof?*

A26. Probably not. So, don't be that idiot.

Q27. *I am a serious-minded person and prefer to write* `Type objective_function<Type>::operator()()` *instead of* `TMB_MAGIC`*. Also I prefer to write eg* `for(int i=0;i<x.size();int ix=indx(i),i++)` *instead of* `FOR( ix, XINDS)` *etc. What have you got to say about* that*?*

A27. I'm not your mother; you can do what you like! Feel free to keep rubbing those sticks together to try to get the fire going... However, if you use TMBO but insist on avoiding the word `TMB_MAGIC`, then you will also have to add the following very serious lines immediately before it:

```
int flubbadub = 0;
int flubbadubbadoo = 0;
```

otherwise your code won't compile[4]. Seriously: it won't!

Q28. *I sense you aren't particularly keen on C++ as a language. Can you provide any historical information?*

A28. With pleasure! It is a little-known fact that the "C" of C actually stands for "Chucky", the evil killer doll of movie fame (disclaimer: I have never seen the movie, it sounds terrifying!). One simply has to admire, however grudgingly, the brutal and (mostly) concise power of the C language; but at the same time, one should acknowledge its *many* design flaws (your Google journey starts here...).

C++, on the other hand, is Chucky after he's been dressed up in a smart suit and sent out on various Sensitivity Awareness courses etc. He has learnt to speak in long turgid corporatesque phrases, and to spout pious platitudes about mindful this and respectful not-that. He has also become incredibly, sometimes lethally, pedantic. But in the end, he's *still the spawn of the devil* and will rip your organs out if given half a chance. Beware.

Q29. *Now I suppose you are going to launch into some lengthy technical diatribe about how bad C++ is? Yawn...*

A29. No, you can look it up yourself on The Internet. All I'm gonna say is: FreePascal[5] .

Q30. *I suppose you think this sort of childish and self-indulgent drivel is "amusing". Well, let me tell you that millions of hardworking families worldwide use C++ which is designed in accordance with ISO principle 134873.735.388b and has a hardworking committee of real people devoted to adding ever-more baroque features in order to paper over serious earlier design deficiencies with cumbersome layers of grotesquery. This superficial and frivolous so-called "FAQ" simply betrays your profound lack of understanding of autochthonous lambda-iterators and prototyped deltoid predicates (which are coming in C++29) and hardly helps the pro-C-ess of world domination. You are merely a caterpillar trying to crawl across the superhighway of progress. Adapt or die, peasant scum!*

A30. That's not technically a question, is it? Next!

[4]OK, why the stupid names? Because the compiler warnings they are designed to defeat are themselves stupid. Fight stupid with stupid!

[5]And yes, it is possible to implement automated Laplace Approximation in FreePascal linked to R. I had this working between about 2006 and 2014, but the maintenance was too much for me.

Q31. *Well that's all very well, but honestly all this so-called "humour" is just going to deter potential users, isn't it?*

A31. I gotta be me.

Q32. *Is this* really *all "just" macros?*

A32. Well... it *was* entirely macros, until I got fancy with the OOB checks in v2.1+. It still uses macros to trigger all the behaviour, but the OOB-check code itself is in C. I got basic OOB-checking to work fine just with macros (it told you where, and R didn't just crash), but it didn't report *all* the index values of the offending subscript operation, and for arrays it could not check the number of parameters (which becomes an appreciable risk when dealing with 9D arrays etc). To fix the latter, I had to add a bit of C code, delving *slightly* into how TMB handles arrays; then I realized I could report the actual index values within that C code, so why not; then I decided that would be useful to extend the actual-index stuff to matrices and vectors too... so in the end, there *is* a bit of C code in "TMBO2.h", but it's only for OOB checks. It uses strings; ugggh. It reminded me of how disgusting C can be... (and no C++ does not solve it, before you are tempted to blurt that out! IMO. Yes: In My Opinion). This answer is pretty boring, sorry.

Q33. *Still, I bet there's lots of clever C macro tricks in TMBO! Are there?*

A33. Ooooh yes! Recursive macros with `__VA_OPT__`, overloading via number of arguments, multiline expansions, sneaky concatenation workarounds, deparenthisation (?), and probably more. In fact it is a veritable smörgåsbord of lightly-seasoned macro *amuse-bouches*. Enjoy!

Q34. *Can you explain them?*

A34. Not really, sorry. I got them all from The Internet, and klunged them together haphazardly until stuff worked. I do now understand *bits* of them, but not very well. The people who figure this macro stuff out are *brilliant*, IMO; there's acknowledgements in the code of `macro_utils_TMBO.h`. It really is very hard to figure out why the tricks work, but they self-evidently do!

Q35. *C macros are evil! I read so on the internet! We should all be using non-abelian prosthetic autotopologies instead, as in the forthcoming C++29 standard! Why aren't you? The termiggsd kdsich jgim gsie*

A35. Feel free to bring up that *fascinating* point with the authors of TMB and R itself, who have gone full macro. Unfortunately your question seems to have degenerated into gibberish by the end, if not well before.

Q36. *TMBO works well, but it sure looks like a lot of work for whoever programmed it; wasn't it a mighty pain sorting out all those macros, just to shift all a few indices by an offset and trap OOBs?*

A36. Well, you may have a point... that is one way to look at it! But not the only way :)