Tools, tips and tricks for finding a good GLLVM

Bert van der Veen

Department of Mathematical Sciences, NTNU

Questions so far?

Background

- GLLVMs are complex models
- Convergence can be difficult
- Presenting results can be challenging

Here I present some tips/tricks that can be useful
Apologies in advance, it will be a bit all over the place

GLMM FAQ

Need something like Ben Bolker's GLMM FAQ, but for GLLVMs
But most (all?) from GLMs and GLMMs also applies to GLLVMs

Outline: a collection of tools, tips and tricks

Tools

- Confidence intervals
- Information criteria
- Hypothesis testing
- Residuals (and code for `DHARMa`)
- `emmeans` and `MuMIn::dredge`
- Concurrent and r^2
- `ordiplot`, `coefplot`, `randomCoefPlot`
- `summary` and plot summary (dev)
- Using `vegan`

Tips or tricks

- LV parameterisation
- Missing response data
- Using a saved `gllvm` object
- Parallelisation
- Ggplotting a `gllvm`
- Optimisation and all that
- Starting values
- Small LV values
- Large loadings, small sigma.lv
- (Very) small standard errors
- Predicting/intervals
- Variation partitioning

More?

Information criteria

`gllvm` provides support for BIC and AIC(c)

- BIC: finds the model closest to the truth
- AIC: finds the model that predicts best
- AICc: penalizes a little stronger than AIC, but was developed for linear models

You can use these, but do so sparingly. They are sensitive to (e.g.,)

- Model misspecification
- Boundary effects (constrained parameters)
- Large sample asymptotics
- Focus issues

Information criteria

Depending on the problem, there are many ways to do selection

- The covariates?
- The random effects?
- The latent variables?
- Combinations thereof?

Information criteria: example

```
##          df      AIC
## model1   97 1266.008
## model2  125 1296.065
```

1) It seems that the model with 2 LVs is better
2) But what if one of the models has poorly converged?

Information criteria

Take-away:

1) Apply carefully
2) Avoid as much as possible
3) Use if necessary

Hypothesis testing

Likelihood ratio test (`anova`) is a similar story:

- Use carefully
- Mind convergence
- Do not use with a large difference in parameters

The same for wald-tests. Do not stare at any of these results for too long.

Confidence intervals

The `confint(.)` function provides some (approximate) wald confidence intervals

E.g.,

```
confint(model1, "theta")
```

```
##                     cilow         ciup
## theta.LV1.1    1.00000000   1.00000000
## theta.LV1.2   -3.76867965  -0.10240769
## theta.LV1.3   -0.71524991   2.28584208
## theta.LV1.4   -3.72709274   0.22885264
## theta.LV1.5   -0.16512287   3.39603618
## theta.LV1.6   -0.35179419   1.11384537
## theta.LV1.7   -0.32620856   1.84630923
## theta.LV1.8   -2.41193163   1.08913022
## theta.LV1.9   -1.65109466   1.01243809
## theta.LV1.10  -1.82590399   0.64122960
## theta.LV1.11  -4.84442267   0.08002783
## theta.LV1.12  -1.01855572   1.28235696
## theta.LV1.13  -1.11480727   2.09461387
## theta.LV1.14  -0.48629367   2.65289224
## theta.LV1.15  -2.58764458   0.19102795
```

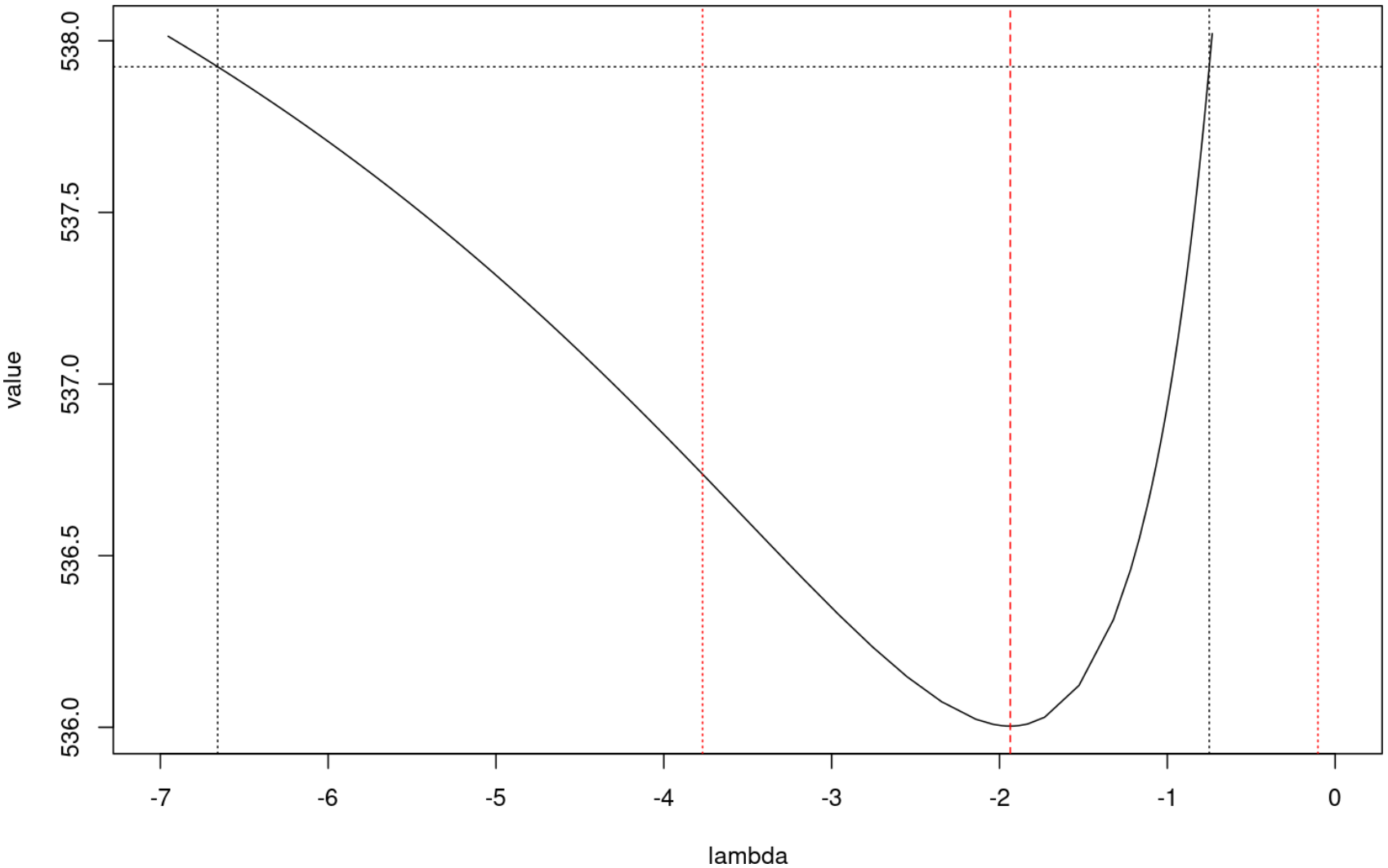
Profile CIs

You can also profile parameters thanks to the `TMB` R-package. These have better properties, but (very) slow to compute. Finding the right parameter can take some doing (see `model$TMBfn$par`).

For example:

```
prof <- TMB::tmbprofile(model1$TMBfn, which(names(model1$TMBfn$par)=="lamb
plot(prof, xlim = c(-7,0))
CIs <- confint(model1);
abline(v = CIs[grepl("theta",row.names(CIs)),][2,1], col = "red", lty = "d
abline(v = CIs[grepl("theta",row.names(CIs)),][2,2], col = "red", lty = "d
abline(v=coef(model1,"loadings")[2,1],col="red",lty="dashed")
```

Profile CIs



Using vegan

The `vegan` `R`-package has a lot of useful functions. Some of these can be used with GLLVMs:

- `procrustes`: for comparison of ordinations,
- `ordiplot`: for plotting, though I have yet to work out how to get it to plot arrows
  - this does require a `scores.gllvm` function (present on github in a presentation)
- `ordisurf`? maybe not
- perhaps other functions, I have yet to discover them

scores.gllvm

```
scores.gllvm <- function(x, display  ="sites", choices = NULL){
  sol <- list()

  if(is.null(choices) || any(choices>(x$num.lv+x$num.lv.c+x$num.RR))){
    choices <- 1:(x$num.lv+x$num.lv.c+x$num.RR)
  }

  if(display%in%c("sites","both")){
    sol$sites <- gllvm::getLV(x)[,choices,drop=FALSE]
      if(display=="sites"){
        sol <- sol$sites
      }
  }

  if(display%in%c("species","both")){
    sol$species <- gllvm::getLoadings(x)[,choices,drop=FALSE]
      if(display=="species"){
        sol <- sol$species
      }
    }

  return(sol)
}
```

Using ordiplot in `gllvm`

This clashes with `vegan`'s `ordiplot`, so if it errs..

- Scaling is applied by default: does not always work well with unimodal models or with covariates in the ordination
- Arrows for covariates are **relative** so you can make them bigger
- Turn pink due to uncertainty (but you can turn it off)
- `fac.center` is new
- Prediction regions of sites
- By default shows square plots

Ordiplot

For when `num.lv`, `num.lv.c`, `num.RR` are in the model.
It is very difficult to create a plotting function that always does well.

- Remember that ordination plots are **conditional** on other effects in the model
- `vegan::ordiplot` will clash with `gllvm`
- `biplot` adds species coordinates
- Note that `ordiplot` scales and rotates, so it will not (exactly) be the same
  - But inference should not be affected

Ordiplot

- Bad scaling can occur for constrained/concurrent ordination
  - Usually due to extreme clustering: species with too few observations
- Bad scaling can occur for unimodal models
  - Optima that are far outside of the estimated gradient mess things up
  - Draw arrows with `spp.arrows` instead
- Arrows for covariates are drawn from the middle, not from 0
- `fac.center` to draw categorical effects as points
- `type` to select which scores to plot for concurrent or hybrid ordination
- `predict.region`
- use `getLV` and `getLoadings(new)` or `optima.gllvm` to create your own plot

Using ordiplot in `vegan`

- If you have `scores.gllvm` you can use `vegan`'s ordiplot
- I have not yet looked into adding the arrows
- It does not do post-hoc scaling or rotation, which is usually needed

Using procrustes

```
vegan::procrustes(gllvm::getLV(model1),
                  vegan::scores(vegan::decorana(Y), choices = 1:2),
                  symmetric = TRUE)
```

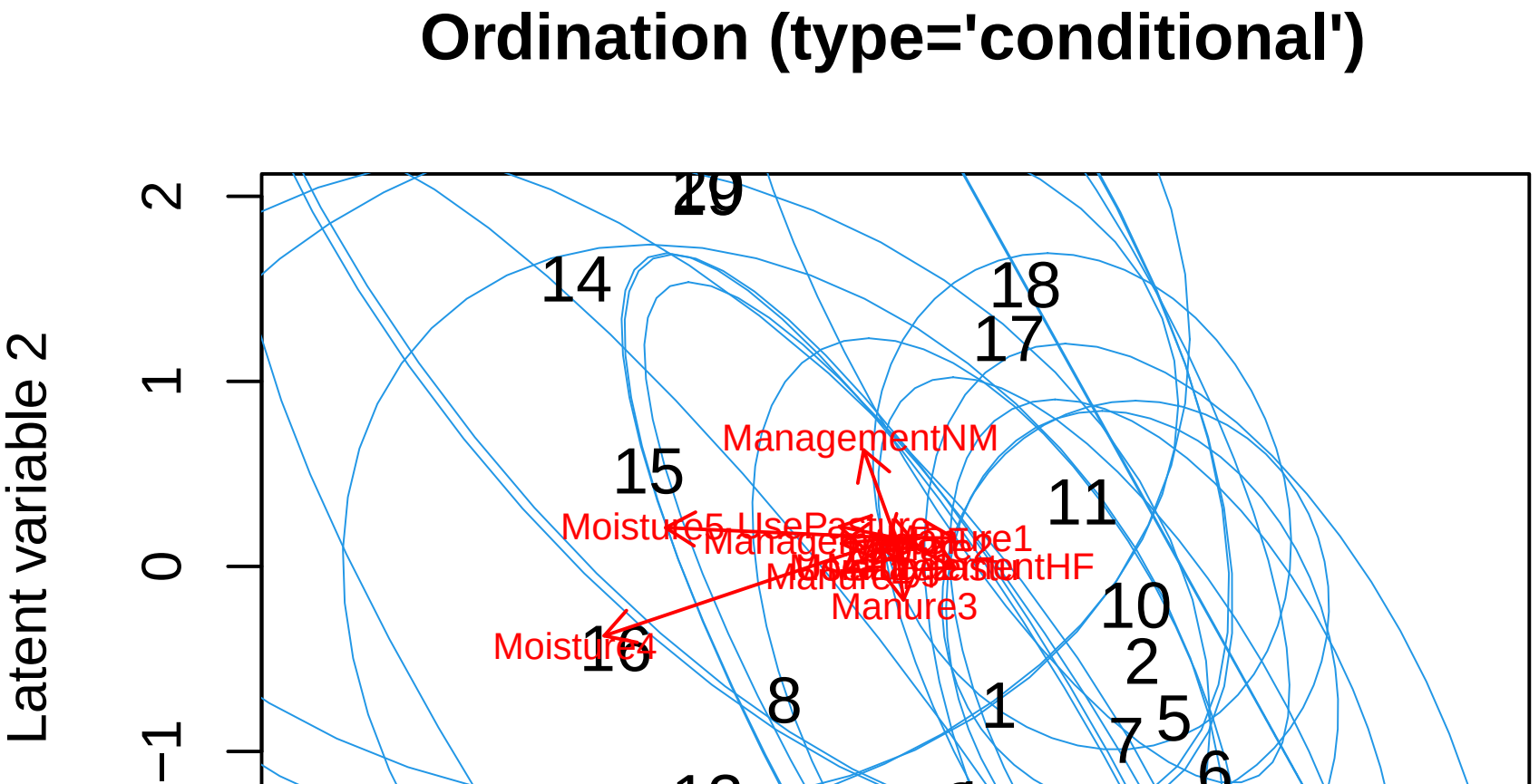
```
##
## Call:
## vegan::procrustes(X = gllvm::getLV(model1), Y = vegan::scores(vegan::de
##
## Procrustes sum of squares:
## 0.312
```

The procrustes error is very useful if you want to compare ordinations.

It is a kind of RMSE that accounts for the different in rotation and scale.

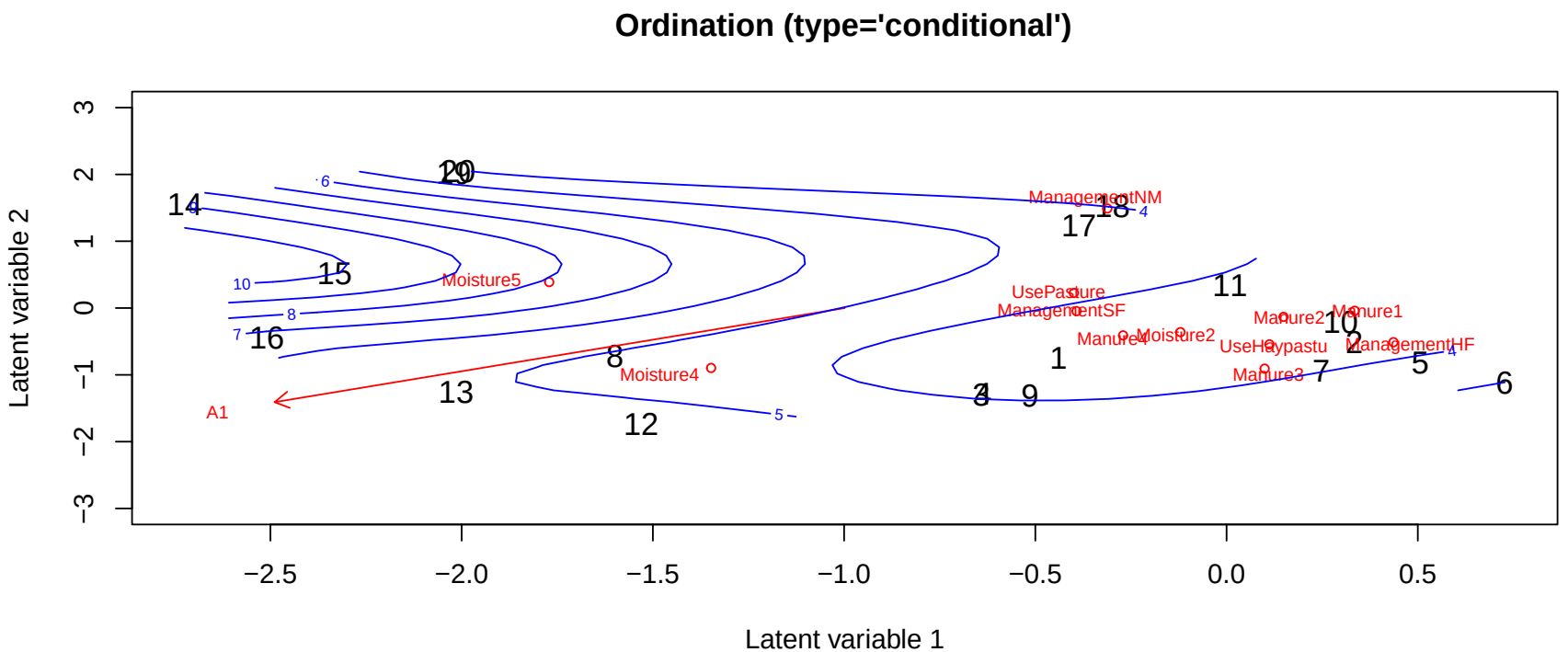
Example with Dune data

```
cnord <- gllvm::gllvm(Y,X, num.lv.c=2,randomB="LV",family="ordinal", seed
gllvm::ordiplot(cnord, predict.region = TRUE, arrow.ci = FALSE)
```



Example with Dune data

```
gllvm::ordiplot(cnord, predict.region = FALSE, fac.center = TRUE, arrow.ci = FALSE, ylim = c(-3,3))
LV = gllvm::getLV(cnord)
rot <- svd(LV)$v
alpha <- 0.5
norms <- sqrt(colSums(LV^2))*sqrt(colSums(gllvm::getLoadings(cnord)^2))
LV <- sweep(LV, 2, (norms^alpha)/sqrt(colSums(LV^2)), "*")%*%rot
vegan::ordisurf(LV, dune.env$A1, add = TRUE, col = "blue")
```



Residuals and DHARMa

gllvm natively provides randomized quantile residuals. It is possible to use the DHARMa package if preferred, though there is no official support. DHARMa has various tests that can be useful:

1) Zero-inflation
2) Overdispersion
3) Other?

This requires use of createDHARMa

Example

We do simulations and predictions with `gllvm`:

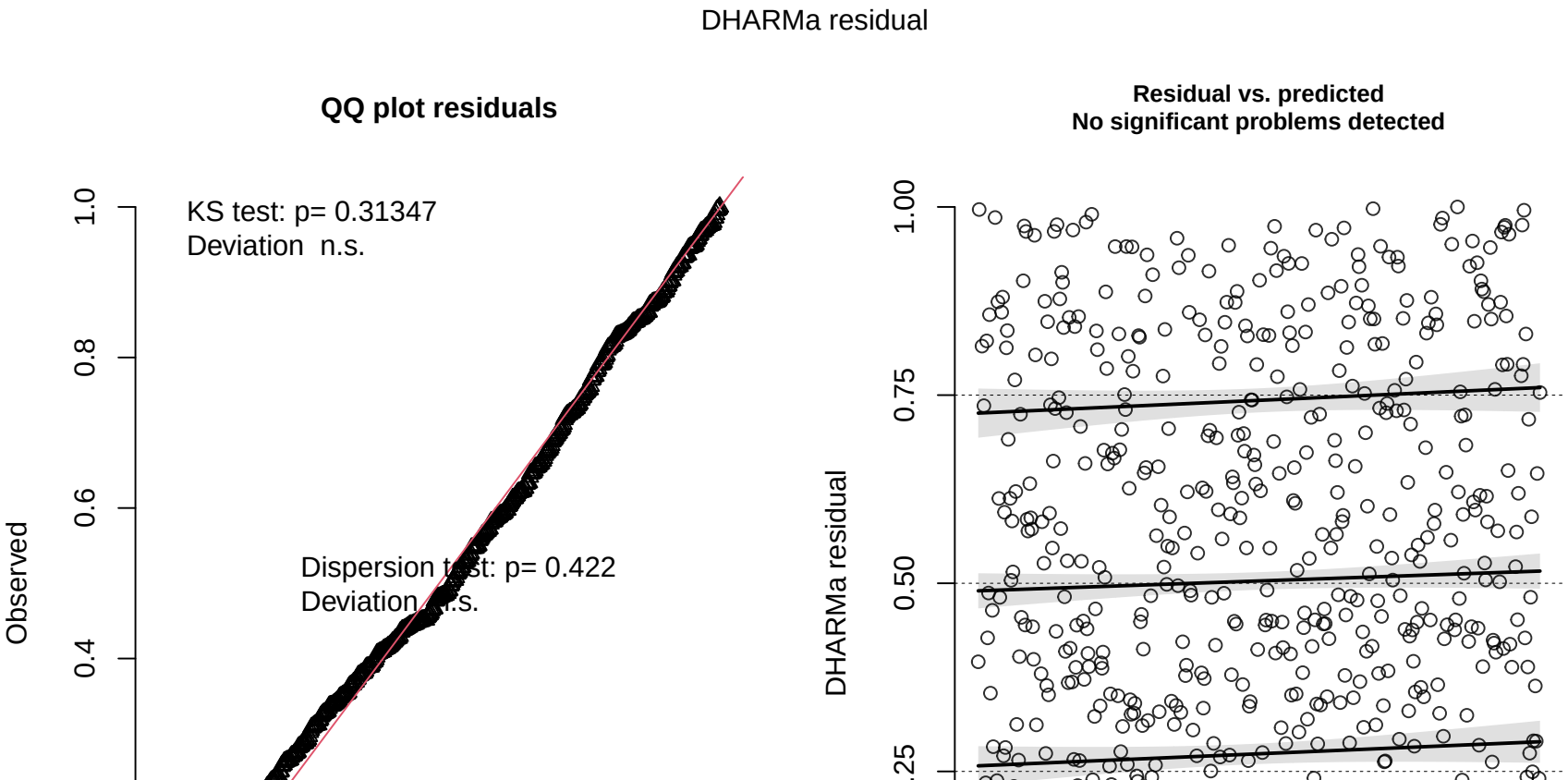
```
# Simulate with model 1000x
sim <- do.call("cbind", replicate(1000,
        c(as.matrix(gllvm::simulate(cnord, conditional=TRUE))), simplify=F
preds <- c(predict(cnord))
obs <- c(as.matrix(Y))
```

Note that these need to be in long format. `DHARMa` handles the rest:

```
dharma <- DHARMa::createDHARMa(
  simulatedResponse=sim,
  observedResponse=obs,
  integerResponse=TRUE,
  fittedPredictedResponse= preds)
```

Example

```
plot(dharma)
```



Residuals and DHARMa

Do not attempt to use the functions that require refitting models.

- GLLVMs (at present) are not suitable for this
- Refitting will take long
- Refitting without monitoring convergence is a recipe for disaster

emmeans

When using categorical variables, R by default uses treatment contrasts. So, the first category is dropped.

A popular choice for inspecting categorical variables is the emmeans package. I recently developed some code to support this (not part yet of gllvm)

- Can be used with JSDM/Residual ordination
- Can be used with constrained ordination (without random slopes)

The functions you need are available on the gllvm github discussions page

Example with Dune data

```
library(emmeans)
coord <- gllvm:::gllvm(Y, X, lv.formula = ~A1+Manure+Moisture+Management, num.RR=2,family="ordinal")
emmeans(coord, ~Manure:species, component = "LV")
```

```
## NOTE: A nesting structure was detected in the fitted model:
##     Manure %in% species, Moisture %in% species, Management %in% species
```

```
##  Manure species     emmean     SE  df asymp.LCL asymp.UCL
##  0      Achimill   -1.3266  0.325 Inf   -1.9641   -0.6890
##  1      Achimill    0.0476  0.327 Inf   -0.5940    0.6892
##  2      Achimill   -0.5447  0.307 Inf   -1.1471    0.0577
##  3      Achimill   -0.6291  0.309 Inf   -1.2352   -0.0231
##  4      Achimill   -1.7443  0.360 Inf   -2.4505   -1.0381
##  0      Agrostol    1.6780  0.477 Inf    0.7435    2.6125
##  1      Agrostol   -1.7412  0.997 Inf   -3.6946    0.2121
##  2      Agrostol   -0.3434  0.580 Inf   -1.4796    0.7928
##  3      Agrostol   -0.1409  0.539 Inf   -1.1982    0.9163
##  4      Agrostol    2.5432  0.831 Inf    0.9154    4.1710
##  0      Airaprae  -22.1585  0.783 Inf  -23.6931  -20.6239
##  1      Airaprae  -16.6781  2.147 Inf  -20.8858  -12.4703
##  2      Airaprae  -22.5258  0.719 Inf  -23.9349  -21.1167
##  3      Airaprae  -23.2119  0.866 Inf  -24.9091  -21.5147
##  4      Airaprae  -31.8141  1.817 Inf  -35.3763  -28.2519
##  0      Alopgeni    0.7223  0.579 Inf   -0.4134    1.8580
##  1      Alopgeni   -3.4468  1.221 Inf   -5.8408   -1.0527
##  2      Alopgeni   -1.1336  0.565 Inf   -2.2410   -0.0263
##  3      Alopgeni   -0.8257  0.556 Inf   -1.9148    0.2633
##  4      Alopgeni    3.1727  0.924 Inf    1.3615    4.9840
##  0      Anthodor   -1.2340  0.405 Inf   -2.0271   -0.4410
##  1      Anthodor    0.2996  0.689 Inf   -1.0499    1.6491
##  2      Anthodor   -0.3959  0.535 Inf   -1.4446    0.6529
##  3      Anthodor   -0.4936  0.523 Inf   -1.5194    0.5322
##  4      Anthodor   -1.7792  0.591 Inf   -2.9382   -0.6201
##  0      Bellpere   -1.2009  0.477 Inf   -2.1361   -0.2657
##  1      Bellpere   -0.4086  0.631 Inf   -1.6448    0.8276
##  2      Bellpere   -0.5935  0.479 Inf   -1.5327    0.3457
```

Dredge

Similar to `emmeans` very limited functionality is supported for `MuMIn::dredge`:

- Not for `lv.formula` type arguments (unless you can figure it out and tell me how)
- not for effects in `formula`
- Repeated disclaimer: blindly fitting without convergence check is icky
- If you figure out more, let me know!

Example with Dune data

```
fit <- gllvm:::gllvm(y = Y, X = X, formula = ~Manure+A1, family = "ordinal
```

```
## Standard errors for parameters could not be calculated, due to singular
```

```
tab <- MuMIn::dredge(fit,varying=list(num.lv=1:2), rank="AICc")
```

```
## Fixed term is "(Intercept)"
```

```
## Standard errors for parameters could not be calculated, due to singular
## Standard errors for parameters could not be calculated, due to singular
```

```
tab
```

Example with Dune data

These warnings are fine

```
fit <- gllvm:::gllvm(y = Y, X = X, formula = ~Manure+A1, family = "ordinal
```

```
## Warning in gllvm.iter(y = y, X = X, lv.X = lv.X.design, formula = fo
## Can't fit ordinal model if there are species with missing classes. S
## 'zeta.struc = `common`'.

## Warning in nlminb(objr$par, objr$fn, objr$gr, control = list(rel.tol
## : NA/NaN function evaluation
## Warning in nlminb(objr$par, objr$fn, objr$gr, control = list(rel.tol
## : NA/NaN function evaluation
## Warning in nlminb(objr$par, objr$fn, objr$gr, control = list(rel.tol
## : NA/NaN function evaluation
## Warning in nlminb(objr$par, objr$fn, objr$gr, control = list(rel.tol
## : NA/NaN function evaluation
## Warning in nlminb(objr$par, objr$fn, objr$gr, control = list(rel.tol
## : NA/NaN function evaluation
```

Variable selection generally

Variable selection is a very difficult problem. What do we do it for?

- Do not get too obsessed with "a good" model
- Use the magnitude of parameter estimators
- In combination with statistical uncertainties

There is (not quite) one exception: adaptive shrinkage

Adaptive shrinkage

- Fit the full model
- Penalize the parameters
- Let a method work its magic

Random effects kind of do this with their quadratic penalty.
This is also what the `randomB` argument is for.

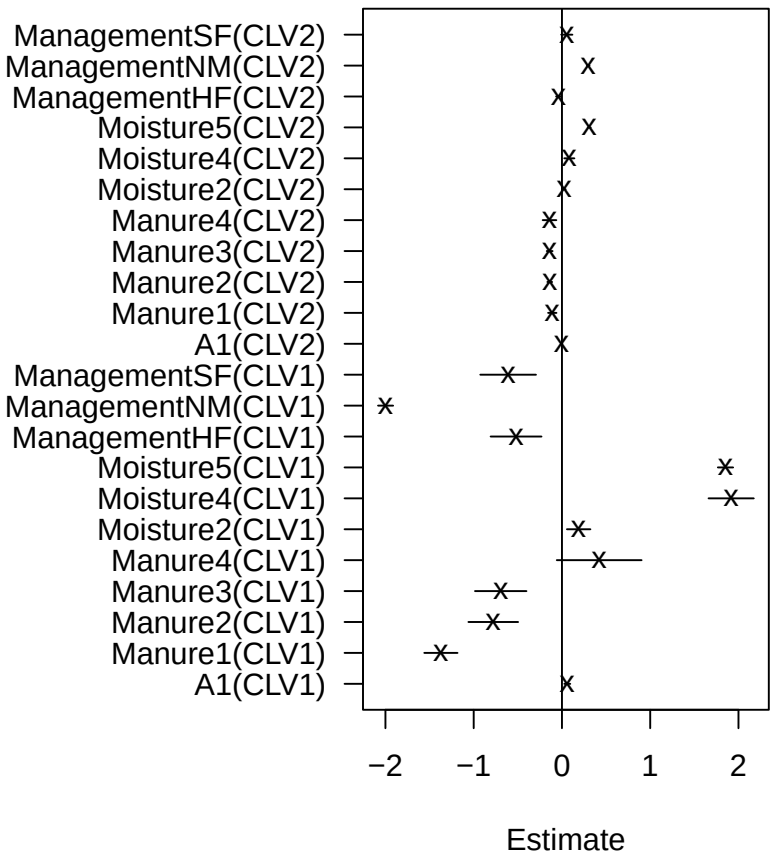
Summary

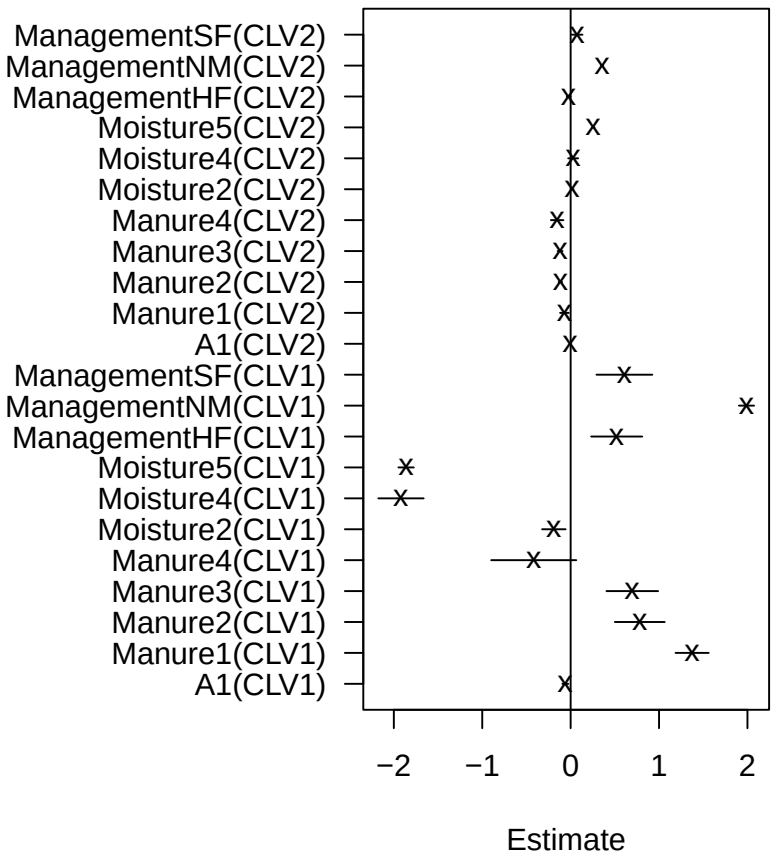
The summary function is a work in progress

- You can plot it (new - dev version)
- It shows correlations of the random effects
- Canonical coefficients
- Other parameter estimates
- But not if there are no standard errors, or for random effects
- It also has a `rotate` arguments for ordination with covariates; rotates to the same direction as the ordination (which also has `rotate`)

Summary example

```
plot(summary(coord), mar=c(4,10,2,4))
plot(summary(coord, rotate = FALSE), mar = c(4,10,2,4))
```

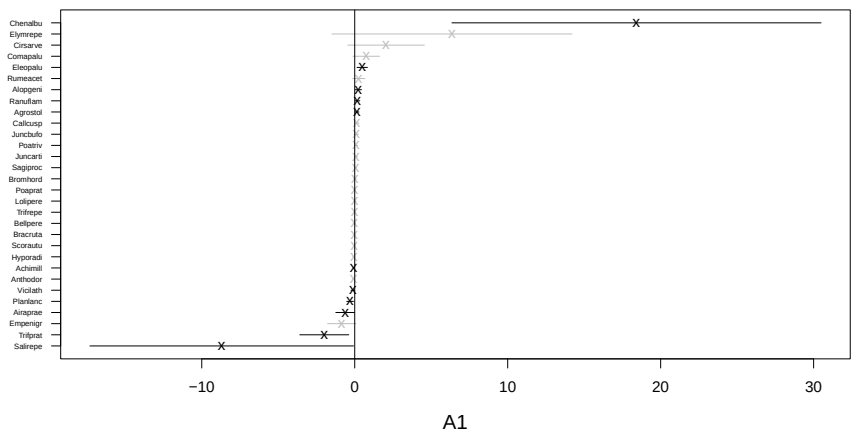




Coefplot

For models with fixed effects

- Makes a "Caterpillar" plot with 95% confidence intervals
- If `which.Xcoef` does not work, double check names (e.g., categorical covariates have one effect per factor level)
- Grey effects have confidence intervals that cross zero
- Also for constrained/concurrent ordination - effects inside or outside of the ordination

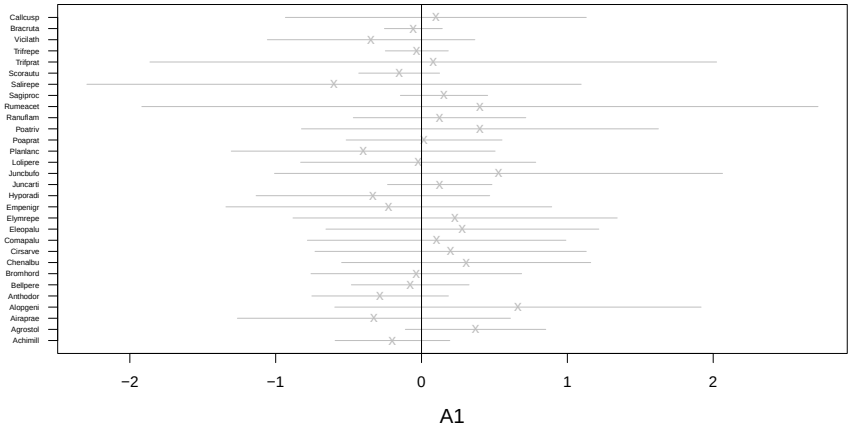


randomCoefplot

For models with random effects

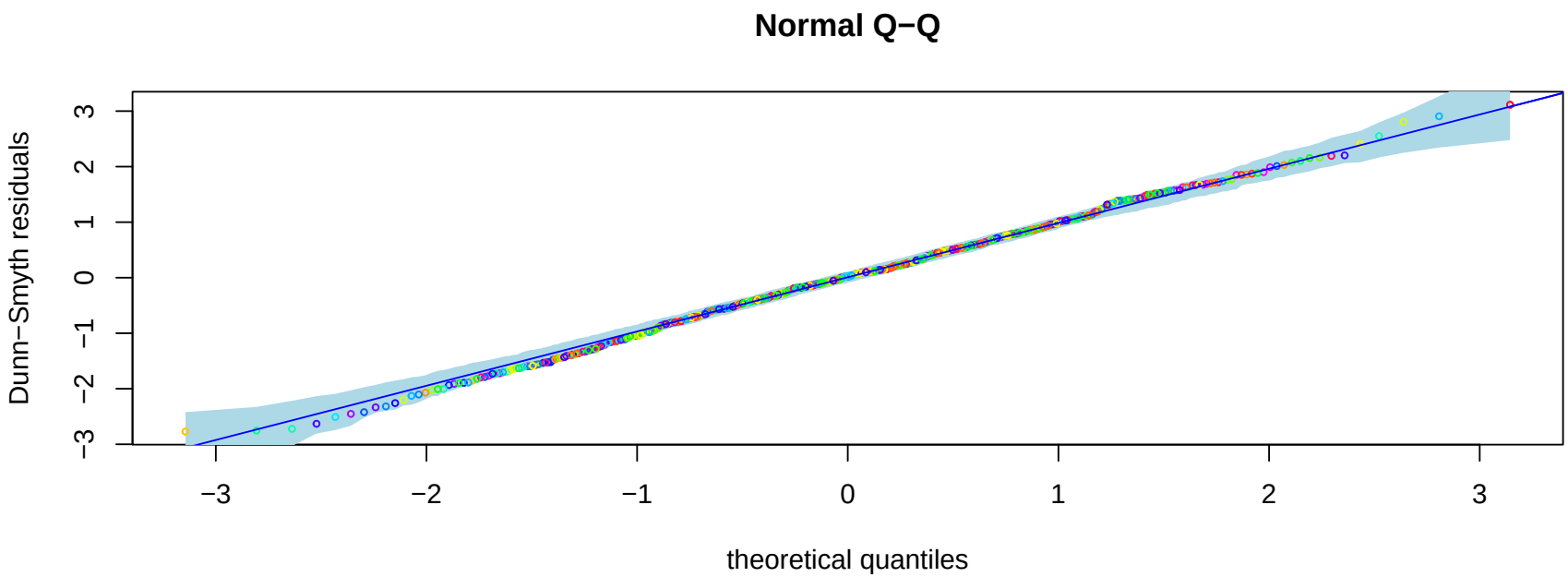
- Makes a "Caterpillar" plot with 95% prediction intervals
- If `which.Xcoef` does not work, double check names (e.g., categorical covariates have one effect per factor level)
- Grey effects have prediction intervals that cross zero
- Also for constrained/concurrent ordination - effects inside or outside of the ordination

```
## Warning in sweep(sweep(covM
## does not recycle exactly ac
```



Residual diagnostic plots in `gllvm`

```
plot(coord, which = 2)
```



- Always check the residuals and make corrections if needed
- Envelopes are included to help check the plots
- QQ-plot: points should fall in the envelopes (more or less)

NAs in responses

It is possible to have missing data in the responses **Y**

- Maybe an experiment went wrong
- Perhaps to "trick" the Phylogenetic model into predicting for ancestors
- Or another reason

`gllvm` will skip the likelihood evaluation and predict the observation instead (i.e., incorporate uncertainty).

Missing data in covariates is not allowed

Using a saved gllvm object

You might want to save a model object when you have finished your analysis. I.e.,:

```
save(cnord,file="cnord.RData")
```

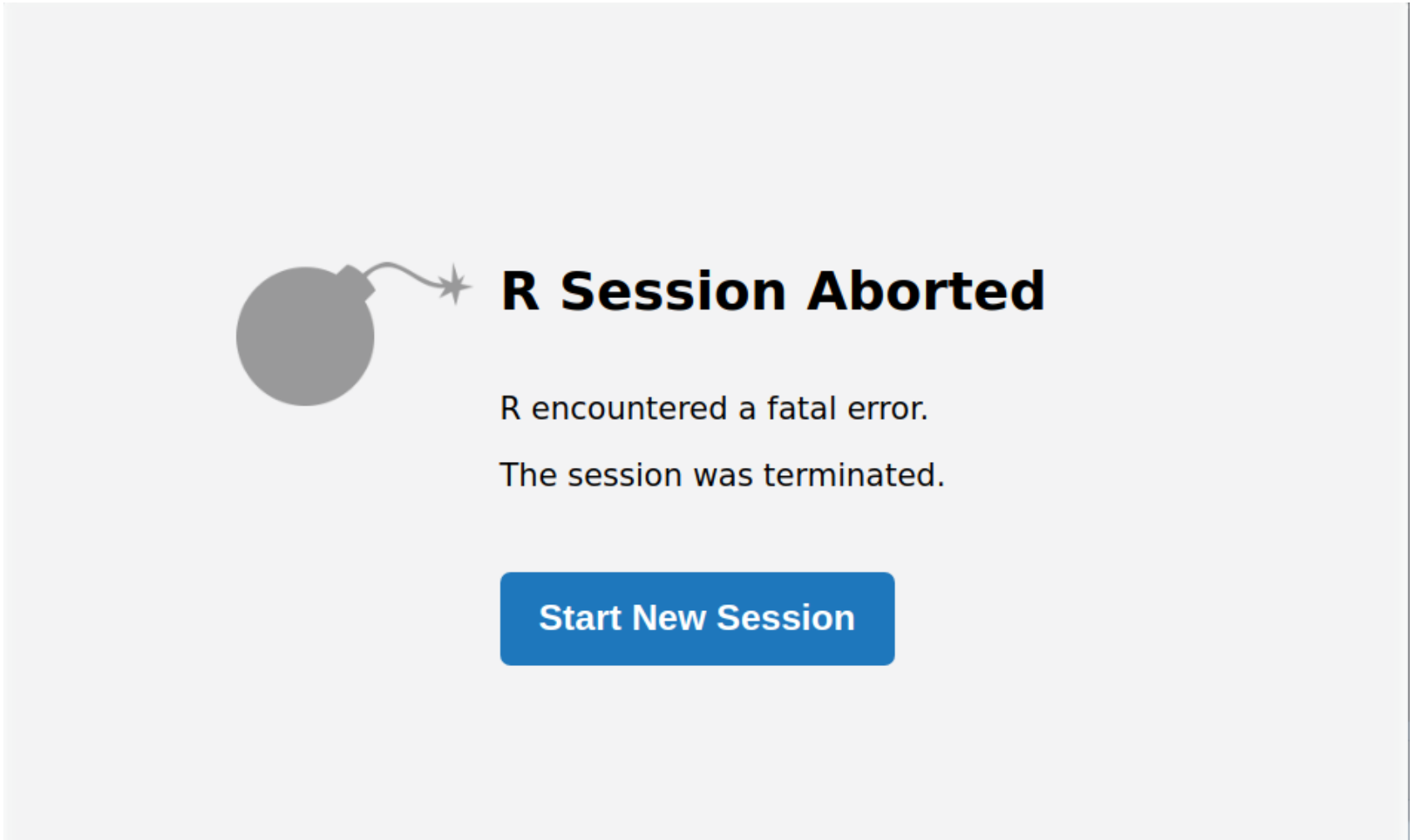
and use it another time:

```
load("cnord.RData")
```

perhaps you ran the model with `sd.errors = FALSE` (as it is faster) and still need to calculate the standard errors with `se.gllvm`.

Using a saved `gllvm` object

If you try to do that you will be met with this screen:



Using a saved `gllvm` object

We first need to retape:

```
cnord$TMBfn$retape()
ses <- gllvm:::se(cnord)
```

And can then safely use the TMB object again.

Parallelisation

Not yet on CRAN, since parallel computation is still very new in `gllvm`

- Parallel computation: use multiple CPU for model fitting
- Can also slow model fitting down (e.g., small models)
- Creating the TMB object is by default sequential due to memory spikes
  - Can be changed via `TMB::config`
- It works by setting `TMB::openmp(...)`
  - ... is the number of CPU to use (see `parallel::detectCores()`)
- Needs to be reset after re-taping

Optimisation and convergence

`gllvm` uses numerical optimisation. There is no guarantee for finding "the best" solution.

- Switching optimisation routine can help
  - `optimizer` and `optim.method`
- Sometimes you need to increase the number of iterations
- Sometimes you need to increase `reltol`
- Constrained/concurrent ordination has an additional constraint on **B**
  - This has separate optimisation routines
  - Separate convergence criterium `reltol.c`
    - If this is not reached, you will get a warning
- Good starting values can make a big difference `starting.val`
- Fit your model repeatedly with `n.init` and `n.init.max`

There are many other control arguments for the approximation, and otherwise

Optimisation and convergence

I am currently trying to find more robust optimisation routines for improved usability.

Starting values

PLOS ONE

RESEARCH ARTICLE

Efficient estimation of generalized linear latent variable models

Jenni Niku[1]*, Wesley Brooks[2☯], Riki Herliansyah[3☯], Francis K. C. Hui[4☯], Sara Taskinen[1☯], David I. Warton[2,5☯]

1 Department of Mathematics and Statistics, University of Jyväskylä, Jyväskylä, Finland, **2** School of Mathematics and Statistics, The University of New South Wales, Sydney, Australia, **3** Department of Mathematics, Kalimantan Institute of Technology, Kalimantan, Indonesia, **4** Research School of Finance, Actuarial Studies & Statistics, Australian National University, Canberra, Australia, **5** Evolution & Ecology Research Centre, The University of New South Wales, Sydney, Australia

☯ These authors contributed equally to this work.
* jenni.m.e.niku@jyu.fi

Niku et al. article looked into starting values.

Starting values

Generally, repeated fits with different starting values is recommended.

`starting.val` has three options: `zero`, `random`, and `res` (default):

- `zero`: Start everything at zero (or 1)
- `random`: Start at randomly generated values
- `res`: Start at smartly generated values

`jitter.var` to add a little noise to `res`

Multiple starts

`n.init`: number times to refit the model (and pick best)

`n.init.max`: `n.init` exit after this many unsuccessful fits (defaults quite high)

Small LV values

- It can happen that you get a lot of small values for the LVs
- Usually in NB or binomial models
- Indicates either lack of convergence, or little data
- For NB you can try to switch to `method = "EVA"` or `"LA"`
- For EVA you can try to switch to `link = "logit"`, or change method
- Also repeatedly fit the model with `n.init`
- If nothing works, omit an LV

Large loadings or small sigma.lv

Usually indicates lack of convergence

- Try reordering the responses: most frequently observed species first
  - Especially for constrained ordination
- Refit the model multiple times
- Sometimes it is normal
- If nothing works, omit an LV

Checking convergence

Checking arrival at the maximum is challenging

- Use `gradient.check = TRUE`
- Or try `hist(model$TMBfn$gr())` - these should near zero
  - If not, refit or change model (lack of convergence)
- `gllvm` might tell you standard errors could not be calculated
  - This indicates lack of convergence, the model **might** be poor
  - Unclear if a **singular** Hessian should be accepted
- We could use more checks on the Hessian
  - If you get a warning about the determinant of the Hessian, something is wrong (simplify/refit)

Standard error calculation in `gllvm` is really quite robust (two fail safes). If it fails, something is wrong

Small standard errors

For a reason that we do not -quite- understand yet the SEs sometimes get very small

- Worst case: overfitting
- Best case: lack of convergence
- Intermediate case: Hessian is being weird

Usually: repeatedly refit the model, change the model, or change `method`

On the calculation of standard errors: an improvement is coming

Prediction and intervals

The predict function works as with a `glm`. It is a work in progress, especially for random effects. There are no intervals available (yet).

You can try simulation.

Variance partitioning in gllvms

Jenni Niku is working on this for `gllvm`, but:

- van der Veen et al (2023) has an R^2 for concurrent ordination
- Nakagawa and Schielzeth (2012) can be constructed for GLLVMs
- `getResidualCov` and `getEnvironmentalCov`

Citing `gllvm`

Please cite the software version `citation("gllvm")`.
Always cite the research articles that developed the method.
Package development and maintenance is **a lot** of work that is not otherwise rewarded.

This is good practice for all packages that you use.

Citing gllvm

To cite the 'gllvm' package in publications use:

Niku, J., Brooks, W., Herliansyah, R., Hui, F. K. C., Korhonen, P., Taskinen, S., van der Veen, B., and Warton, D. I. (2023). gllvm: Generalized Linear Latent Variable Models.R package version 1.4.3.

accompanied by any of associated research articles, as applicable:

Niku, J., Hui, F. K. C., Taskinen, S., and Warton, D. I. (2019). gllvm - Fast analysis of multivariate abundance data with generalized linear latent variable models in R. Methods in Ecology and Evolution, 10, 2173-2182.

Niku, J., Hui, F. K. C., Taskinen, S., and Warton, D. I. (2021). Analyzing environmental-trait interactions in ecological communities with fourth-corner latent variable models. Environmetrics, 32, 1-17.

van der Veen, B., Hui, F. K. C., Hovstad, K.A., Solbu, E.B., and O'Hara, R.B. (2021). Model-based ordination for species with unequal niche widths. Methods in Ecology and Evolution, 12, 1288-1300.

van der Veen, B., Hui, F. K. C., Hovstad, K.A., and O'Hara, R.B. (2023). Concurrent ordination: simultaneous unconstrained and constrained latent variable modelling. Methods in Ecology and Evolution, 14, 683-695.

Korhonen, P., Hui, F. K. C., Niku, J., and Taskinen, S. (2023). Fast and universal estimation of latent variable models using extended variational approximations. Statistics and Computing, 33, 1-16.

Conclusion

- More points, let me know
- Suggestions to improve usability, let us know
- Questions on github discussions
- Bug reports on github under issues