convergence {lme4} R Documentation

Assessing Convergence for Fitted Models

Description

`[g]lmer` fits may produce convergence warnings; these do **not** necessarily mean the fit is incorrect (see "Theoretical details" below). The following steps are recommended assessing and resolving convergence warnings (also see examples below):

- double-check the model specification and the data
- adjust stopping (convergence) tolerances for the nonlinear optimizer, using the `optCtrl` argument to `[g]lmerControl` (see "Convergence controls" below)
- center and scale continuous predictor variables (e.g. with `scale`)
- double-check the Hessian calculation with the more expensive Richardson extrapolation method (see examples)
- restart the fit from the reported optimum, or from a point perturbed slightly away from the reported optimum
- use `allFit` to try the fit with all available optimizers (e.g. several different implementations of BOBYQA and Nelder-Mead, L-BFGS-B from `optim`, `nlminb`, ...). While this will of course be slow for large fits, we consider it the gold standard; if all optimizers converge to values that are practically equivalent, then we would consider the convergence warnings to be false positives.

Details

Convergence controls

- the controls for the `nloptwrap` optimizer (the default for `lmer`) are

  ftol_abs

  : (default 1e-6) stop on small change in deviance

  ftol_rel

  : (default 0) stop on small relative change in deviance

  xtol_abs

  : (default 1e-6) stop on small change of parameter values

  xtol_rel

  : (default 0) stop on small relative change of parameter values

  maxeval

  : (default 1000) maximum number of function evaluations

Changing `ftol_abs` and `xtol_abs` to stricter values (e.g. 1e-8) is a good first step for resolving convergence problems, at the cost of slowing down model fits.

- the controls for `minqa::bobyqa` (default for `glmer` first-stage optimization) are

  rhobeg

  (default 2e-3) initial radius of the trust region

  rhoend

  (default 2e-7) final radius of the trust region

  maxfun

  (default 10000) maximum number of function evaluations

  `rhoend`, which describes the scale of parameter uncertainty on convergence, is approximately analogous to `xtol_abs`.

- the controls for `Nelder_Mead` (default for `glmer` second-stage optimization) are

  FtolAbs

  (default 1e-5) stop on small change in deviance

  FtolRel

  (default 1e-15) stop on small relative change in deviance

  XtolRel

  (default 1e-7) stop on small change of parameter values

  maxfun

  (default 10000) maximum number of function evaluations

Theoretical issues

lme4 uses general-purpose nonlinear optimizers (e.g. Nelder-Mead or Powell's BOBYQA method) to estimate the variance-covariance matrices of the random effects. Assessing the convergence of such algorithms reliably is difficult. For example, evaluating the Karush-Kuhn-Tucker conditions (convergence criteria which reduce in simple cases to showing that the gradient is zero and the Hessian is positive definite) is challenging because of the difficulty of evaluating the gradient and Hessian.

We (the `lme4` authors and maintainers) are still in the process of finding the best strategies for testing convergence. Some of the relevant issues are

- the gradient and Hessian are the basic ingredients of KKT-style testing, but (at least for now) `lme4` estimates them by finite-difference approximations which are sometimes unreliable.

- The Hessian computation in particular represents a difficult tradeoff between computational expense and accuracy. At present the Hessian computations used for convergence checking (and for estimating standard errors of fixed-effect parameters for GLMMs) follow the **ordinal** package in using a naive but computationally cheap centered finite difference computation (with a fixed step size of *1e-4*). A more

reliable but more expensive approach is to use Richardson extrapolation, as implemented in the **numDeriv** package.

- it is important to scale the estimated gradient at the estimate appropriately; two reasonable approaches are

  1. scale gradients by the inverse Cholesky factor of the Hessian, equivalent to scaling gradients by the estimated Wald standard error of the estimated parameters. `lme4` uses this approach; it requires the Hessian to be estimated (although the Hessian is required for reliable estimation of the fixed-effect standard errors for GLMMs in any case).

  2. use unscaled gradients on the random-effects parameters, since these are essentially already unitless (for LMMs they are scaled relative to the residual variance; for GLMMs they are scaled relative to the sampling variance of the conditional distribution); for GLMMs, scale fixed-effect gradients by the standard deviations of the corresponding input variable

- Exploratory analyses suggest that (1) the naive estimation of the Hessian may fail for large data sets (number of observations greater than approximately *1e5*); (2) the magnitude of the scaled gradient increases with sample size, so that warnings will occur even for apparently well-behaved fits with large data sets.

See Also

`lmerControl`, `isSingular`

Examples

```
if (interactive()) {
fm1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)

## 1. decrease stopping tolerances
strict_tol <- lmerControl(optCtrl=list(xtol_abs=1e-8, ftol_abs=1e-8))
if (all(fm1@optinfo$optimizer=="nloptwrap")) {
    fm1.tol <- update(fm1, control=strict_tol)
}

## 2. center and scale predictors:
ss.CS <- transform(sleepstudy, Days=scale(Days))
fm1.CS <- update(fm1, data=ss.CS)

## 3. recompute gradient and Hessian with Richardson extrapolation
devfun <- update(fm1, devFunOnly=TRUE)
if (isLMM(fm1)) {
    pars <- getME(fm1,"theta")
} else {
    ## GLMM: requires both random and fixed parameters
    pars <- getME(fm1, c("theta","fixef"))
}
if (require("numDeriv")) {
    cat("hess:\n"); print(hess <- hessian(devfun, unlist(pars)))
    cat("grad:\n"); print(grad <- grad(devfun, unlist(pars)))
    cat("scaled gradient:\n")
    print(scgrad <- solve(chol(hess), grad))
}
## compare with internal calculations:
fm1@optinfo$derivs

## 4. restart the fit from the original value (or
## a slightly perturbed value):
fm1.restart <- update(fm1, start=pars)
set.seed(101)
pars_x <- runif(length(pars),pars/1.01,pars*1.01)
```

```
fm1.restart2 <- update(fm1, start=pars_x,
                       control=strict_tol)

## 5. try all available optimizers

  fm1.all <- allFit(fm1)
  ss <- summary(fm1.all)
  ss$ fixef               ## fixed effects
  ss$ llik                ## log-likelihoods
  ss$ sdcor               ## SDs and correlations
  ss$ theta               ## Cholesky factors
  ss$ which.OK            ## which fits worked

}
```

[Package *lme4* version 1.1-27.1 Index]