Package: TMBhelper (via r-universe)

September 12, 2024

Type Package

Title Package for basic helper functions that are not worth putting in a specialized contributed package

Version 1.4.0

Date 2022-01-31

Maintainer James Thorson <James.Thorson@noaa.gov>

Description Basic functions as suggested and contributed by users and not worth making a specific contributed R package

Imports abind, tmbstan

License GPL-3

RoxygenNote 7.1.2

Repository https://noaa-fims.r-universe.dev

RemoteUrl https://github.com/kaskr/TMB_contrib_R

RemoteRef HEAD

RemoteSha d275e5226d94a8b3b4b3417dbe89dac533d5f8c5

Contents

check_estimability *Check for identifiability of fixed effects*

Description

check_estimability calculates the matrix of second-derivatives of the marginal likelihood w.r.t. fixed effects, to see if any linear combinations are not estimable (i.e. cannot be uniquely estimated conditional upon model structure and available data, e.g., resulting in a likelihood ridge and singular, non-invertable Hessian matrix)

Usage

```
check_estimability(obj, h)
```

Arguments

obj	The compiled object
h	optional argument containing pre-computed Hessian matrix

Value

A tagged list of the hessian and the message

Check_Identifiable *Copy of check_estimability*

Description

Included for continuity when using old scripts

Usage

```
Check_Identifiable(...)
```

Details

Please use check_estimability to see list of arguments and usage

extract_fixed *Extract fixed effects*

Description

extract_fixed extracts the best previous value of fixed effects, in a way that works for both mixed and fixed effect models

Usage

```
extract_fixed(obj)
```

Arguments

obj	The compiled object

Value

A vector of fixed-effect estimates

fit_tmb *Optimize a TMB model*

Description

fit_tmb runs a TMB model and generates standard diagnostics

Usage

```
fit_tmb(
  obj,
  fn = obj$fn,
  gr = obj$gr,
  startpar = NULL,
  lower = -Inf,
  upper = Inf,
  getsd = TRUE,
  control = list(eval.max = 10000, iter.max = 10000, trace = 1),
  bias.correct = FALSE,
 bias.correct.control = list(sd = FALSE, split = NULL, nsplit = NULL, vars_to_correct
    = NULL),
  savedir = NULL,
  loopnum = 2,
  newtonsteps = 0,
  n = Inf,
  getReportCovariance = FALSE,
```

```
  getJointPrecision = FALSE,
  getHessian = FALSE,
  quiet = FALSE,
  start_time_elapsed = as.difftime("0:0:0"),
  ...
)
```

Arguments

obj	The compiled TMB object
startpar	Starting values for fixed effects (default NULL uses obj$par)
lower	vectors of lower and upper bounds, replicated to be as long as start. If unspecified, all parameters are assumed to be unconstrained.
upper	vectors of lower and upper bounds, replicated to be as long as start. If unspecified, all parameters are assumed to be unconstrained.
getsd	Boolean indicating whether to run standard error calculation; see sdreport for details
control	A list of control parameters. For details see nlminb
bias.correct	Boolean indicating whether to do epsilon bias-correction; see sdreport and fit_tmbfor details
bias.correct.control	tagged list of options for epsilon bias-correction, where vars_to_correct is a character-vector of ADREPORT variables that should be bias-corrected
savedir	directory to save results (if savedir=NULL, then results aren't saved)
loopnum	number of times to re-start optimization (where loopnum=3 sometimes achieves a lower final gradient than loopnum=1)
newtonsteps	Integer specifying the number of extra newton steps to take after optimization (alternative to loopnum). Each newtonstep requires calculating the Hessian matrix and is therefore slow. But for well-behaved models, each Newton step will typically decrease the maximum gradient of the loglikelihood with respect to each fixed effect, and therefore this option can be used to achieve an arbitrarily low final gradient given sufficient time for well-behaved models. However, this option will also perform strangely or have unexpected consequences for poorly-behaved models, e.g., when fixed effects are at upper or lower bounds.
n	sample sizes (if n!=Inf then n is used to calculate BIC and AICc)
getReportCovariance	Get full covariance matrix of ADREPORTed variables?
getJointPrecision	Optional. Return full joint precision matrix of random effects and parameters?
getHessian	return Hessian for usage in later code
quiet	Boolean whether to print additional messages results to terminal
start_time_elapsed	how much time has elapsed prior to calling fit_tmb, for use, e.g., when calling fit_tmb multiple times in sequence, where start_time_elapsed = opt_previous$time_for_run
...	list of settings to pass to sdreport

Value

the standard output from nlminb, except with additional diagnostics and timing info, and a new slot containing the output from sdreport

References

For more details see https://doi.org/10.1016/j.fishres.2015.11.016

Examples

```
TMBhelper::fit_tmb( Obj ) # where Obj is a compiled TMB object
```

---

oneStepPredict_deltaModel

Calculate one-step-ahead (OSA) residuals for a mixed-effects delta-model.

Description

oneStepPredict_deltaModel is a wrapper for oneStepPredict for distributions with a mixture of discrete and continuous distributions

Usage

```
oneStepPredict_deltaModel(obj, observation.name, deltaSupport = 0, ...)
```

Arguments

obj	Output from MakeADFun.
observation.name	Character naming the observation in the template.
deltaSupport	integer-vector, listing values that have a dirac-delta within an otherwise continuous distribution
...	list of arguments to pass to oneStepPredict

Details

It is convenient to compute one-step-ahead residuals for data that arise as a mixture of continuous and discrete distributions. One common example is a delta-model, which arises as a mixture of an encounter probability and a continuous distribution for biomass given an encounter. In these cases, it is possible to apply oneStepPredict twice, once for observations falling within the continuous domain, and again for observations in the discrete domain, and then combining the two. This function provides an example of doing so. It is designed to use the 'method="cdf"' feature in oneStepPredict, and code changes in the CPP side are shown in the example script 'deltaModel.R' loaded within directory 'system.file("tmb",package="TMBhelper")'. This example also shows a proof-of-concept for uniform residuals under a (sufficiently-close-to) correctly specified model.

Value

the standard output from oneStepPredict

Examples

```
## Not run:

library(TMB)
library(RandomFields)
library(INLA) # FROM: http://www.r-inla.org/download

###################
# Poisson-link gamma distribution
###################

# n = numbers density
# w = weight-per-number
# cv = CV of gamma
dpoislinkgamma = function(x, n, w, cv){
  pow = function(a,b) a^b
  enc_prob = 1 - exp(-n)
  posmean = n * w / enc_prob
  if( x==0 ){
    dens = 1 - enc_prob
  }else{
    dens = enc_prob * dgamma(x, shape=pow(cv,-2), scale=posmean*pow(cv,2))
  }
  if(log==FALSE) return(dens)
  if(log==TRUE) return(log(dens))
}
ppoislinkgamma = function(x, n, w, cv){
  pow = function(a,b) a^b
  enc_prob = 1 - exp(-n)
  posmean = n * w / enc_prob
  dist = 1 - enc_prob
  if( x>0 ){
    posmean = n*w
    dist = dist + enc_prob * pgamma(x, shape=pow(cv,-2), scale=posmean*pow(cv,2))
  }
  return(dist)
}
rpoislinkgamma = function(n, w, cv){
  pow = function(a,b) a^b
  enc_prob = 1 - exp(-n)
  posmean = n * w / enc_prob
  enc = rbinom(n=1, prob=enc_prob, size=1)
  x = enc * rgamma(n=1, shape=pow(cv,-2), scale=posmean*pow(cv,2))
  return(x)
}

###################
# Simulate data
```

```
####################

Dim = c("n_x"=10, "n_y"=10)
loc_xy = expand.grid("x"=1:Dim['n_x'], "y"=1:Dim['n_y'])
Scale = 2
Sigma2 = (0.5) ^ 2
beta0 = 1
w = 1
cv = 0.1

# Simulate spatial process
RMmodel = RMgauss(var=Sigma2, scale=Scale)
epsilon_xy = array(RFsimulate(model=RMmodel, x=loc_xy[,'x'], y=loc_xy[,'y'])@data[,1], dim=Dim)

# Simulate samples
c_xy = array(NA, dim=dim(epsilon_xy))
for(x in 1:nrow(c_xy)){
for(y in 1:ncol(c_xy)){
  c_xy[x,y] = rpoislinkgamma( n=exp(beta0 + epsilon_xy[x,y]), w=w, cv=cv )
}}

#' ####################
#' # SPDE-based
####################

# create mesh
mesh = inla.mesh.create( loc_xy, plot.delay=NULL, refine=FALSE)
# Create matrices in INLA
spde <- inla.spde2.matern(mesh, alpha=2)

# COmpile
setwd( system.file("tmb",package="TMBhelper") )
compile( "deltaModel.cpp" )
dyn.load( dynlib("deltaModel") )

# Build object
Data = list("c_i"=as.vector(c_xy), "j_i"=mesh$idx$loc-1, "M0"=spde$param.inla$M0, "M1"=spde$param.inla$M1, "M2"=
Params = list( "beta0"=0, "ln_tau"=0, "ln_kappa"=0, "ln_w"=0, "ln_cv"=0, "epsilon_j"=rep(0,nrow(spde$param.inla$M
Map = list( "ln_tau"=factor(NA), "ln_kappa"=factor(NA), "epsilon_j"=factor(rep(NA,length(Params$epsilon_j))) )
Obj = MakeADFun( data=Data, parameters=Params, random="epsilon_j", map=Map )

# Optimize
Opt = TMBhelper::fit_tmb( obj=Obj, newtonsteps=0, getsd=FALSE )
report = Obj$report()

# Run
osa = oneStepPredict_deltaModel( obj=Obj, observation.name="c_i", method="cdf",
 data.term.indicator="keep", deltaSupport=0, trace=TRUE, seed=1 ) #discreteSupport = seq(0,max(Data$c_i),by=1) )
qqnorm(osa$residual); abline(0,1)

# should be uniform from 0 to mean(c_xy==0) when mapping off random effects
qresid = NULL
for(i in 1:1000){
```

```
  osa = oneStepPredict_deltaModel( obj=Obj, observation.name="c_i", method="cdf",
   data.term.indicator="keep", deltaSupport=0, trace=FALSE, seed=i ) #discreteSupport = seq(0,max(Data$c_i),by=1)
  qresid = c( qresid, pnorm(osa[which(Obj$env$data[["c_i"]]==0),'residual']) )
}
hist(qresid)
abline( v=mean(c_xy==0), lwd=3, lty="dotted" )


## End(Not run)
```

`Optimize` *Copy of fit_tmb*

Description

Included for continuity when using old scripts

Usage

```
Optimize(...)
```

Details

Please use `?fit_tmb` to see list of arguments and usage

`sample_re` *Sample random effects to correct for re-transformation bias*

Description

`sample_re` calculates MCMC samples of random effects conditional upon estimated MLE for fixed effects, and then uses each sample to calculate objects in the report. This is useful e.g., in correcting for re-transformation bias (by calculating the posterior mean of a nonlinear transformation of random effects) or visualizing random-effect variance (which often can be time-consuming using the delta-method in models with many random effects)

Usage

```
sample_re(
  obj,
  warmup = 50,
  iter = 150,
  report_names = NULL,
  dat = obj$env$data,
  ...
)
```

Arguments

obj	the TMB object after parameter estimation
warmup	A positive integer specifying the number of warmup (aka burnin) iterations per chain. If step-size adaptation is on (which it is by default), this also controls the number of iterations for which adaptation is run (and hence these warmup samples should not be used for inference). The number of warmup iterations should be smaller than iter and the default is iter/2.
iter	A positive integer specifying the number of iterations for each chain (including warmup). The default is 2000.
report_names	which elements of obj$report() should be recorded; default report_names=NULL uses report_names=names(obj$report())
...	adding arguments to pass to tmbstan

Value

A tagged list containing:

stan_out output from tmbstan

report_full A list of output from obj$report()[report_names], except with extra dimension for each MCMC sample

run_time total run time

References

For a discussion of the epsilon-estimator as alternative method to correct for re-transformation bias see https://doi.org/10.1016/j.fishres.2015.11.016

TMBAIC *Calculate marginal AIC for a fitted model*

Description

TMBAIC calculates AIC for a given model fit

Usage

```
TMBAIC(opt, p = 2, n = Inf)
```

Arguments

opt	the output from nlminb or optim
p	the penalty on additional fixed effects (default=2, for AIC)
n	the sample size, for use in AICc calculation (default=Inf, for which AICc=AIC)

Value

AIC, where a parsimonious model has a AIC relative to other candidate models

Index