Package 'RegressionFactory'

January 20, 2025

Type Package

Title Expander Functions for Generating Full Gradient and Hessian from Single-Slot and Multi-Slot Base Distributions

Version 0.7.4

Date 2020-10-26

Author Alireza S. Mahani, Mansour T.A. Sharabiani

Maintainer Alireza S. Mahani <alireza.s.mahani@gmail.com>

Description

The expander functions rely on the mathematics developed for the Hessian-definiteness invariance theorem for linear projection transformations of variables, described in authors' paper, to generate the full, high-dimensional gradient and Hessian from the lower-dimensional derivative objects. This greatly relieves the computational burden of generating the regression-function derivatives, which in turn can be fed into any optimization routine that utilizes such derivatives. The theorem guarantees that Hessian definiteness is preserved, meaning that reasoning about this property can be performed in the low-dimensional space of the base distribution. This is often a much easier task than its equivalent in the full, high-dimensional space. Definiteness of Hessian can be useful in selecting optimization/sampling algorithms such as Newton-Raphson optimization or its sampling equivalent, the Stochastic Newton Sampler. Finally, in addition to being a computational tool, the regression expansion framework is of conceptual value by offering new opportunities to generate novel regression problems.

License GPL (>= 2)

Suggests sns, MfUSampler

NeedsCompilation no

Repository CRAN

Date/Publication 2020-10-26 06:30:07 UTC

Contents

`fbase1.binomial.logit` *Single-Parameter Base Log-likelihood Function(s) for Binomial GLM*

Description

Vectorized, single-parameter base log-likelihood functions for binomial GLM using various link functions. These base functions can be supplied to the expander function `regfac.expand.1par` in order to obtain the full, high-dimensional log-likleihood and its derivatives.

Usage

```
fbase1.binomial.logit(u, y, fgh=2, n=1)
fbase1.binomial.probit(u, y, fgh=2, n=1)
fbase1.binomial.cauchit(u, y, fgh=2, n=1)
fbase1.binomial.cloglog(u, y, fgh=2, n=1)
```

Arguments

`u`	Varying parameter of the base log-likelihood function. This parameter is intended to be projected onto a high-dimensional space using the familiar regression transformation of `u <- X%*%beta`. In the typical use-case where the caller is `regfac.expand.1par`, a vector of values are supplied, and return objects will have the same length as `u`.
`y`	Fixed slot of the base distribution, corresponding to the response variable in the regression model. For `binomial` family, it must be an integer vector with values between `0` and `n`.
`fgh`	Integer with possible values 0,1,2. If `fgh=0`, the function only calculates and returns the log-likelihood vector and no derivatives. If `fgh=1`, it returns the log-likelihood and its first derivative in a list. If `fgh=2`, it returns the log-likelihood, as well as its first and second derivatives in a list.
`n`	Number of trials in the binomial model. This parameter is assumed to be fixed, and must be supplied by the user. If `n==1`, the model is reduced to binary logit/probit/cauchit/cloglog regression.

Value

If `fgh==0`, the `logit` version returns `-(n*log(1+exp(-u))+(n-y)*u)`, the `probit` returns `y*log(pnorm(u))+(n-y)*log(1`
the `cauchit` returns `y*log(pcauchy(u))+(n-y)*log(1-pcauchy(u))`, and the `cloglog` returns `y*log(1-exp(-exp(u)))-(n-y)*exp(u)` . If `fgh==1`, a list is returned with elements `f` and `g`, where the latter is a vector of length `length(u)`, with each element being the first derivative of the above expressions. If `fgh==2`, the list will include an element named `h`, consisting of the second derivatives of `f` with respect to `u`.

Note

In all base log-likelihood functions, we have dropped any additive terms that are independent of the distribution parameter, e.g. constant terms or those terms that are dependent on the response variable only. This is done for computational efficiency. Therefore, these functions cannot be used to obtain the absolute values of log-likelihood functions but only in the context of optimization and/or sampling. Users can write thin wrappers around these functions to add the constant terms to the function value. (Derivatives do not need correction. For binomial family, all factorial terms are ignored since they only depend on `n` and `y`.)

Author(s)

Alireza S. Mahani, Mansour T.A. Sharabiani

See Also

`regfac.expand.1par`

Examples

```
## Not run:
library(sns)
library(MfUSampler)

# using the expander framework and binomial base log-likelihood
# to define log-likelihood function for binary logit regression
loglike.logit <- function(beta, X, y, fgh) {
  regfac.expand.1par(beta, X, y, fbase1.binomial.logit, fgh, n=1)
}

# generate data for logistic regression
N <- 1000
K <- 5
X <- matrix(runif(N*K, min=-0.5, max=+0.5), ncol=K)
beta <- runif(K, min=-0.5, max=+0.5)
y <- 1*(runif(N) < 1.0/(1+exp(-X%*%beta)))

# obtaining glm coefficients for comparison
beta.glm <- glm(y~X-1, family="binomial")$coefficients

# mcmc sampling of log-likelihood
nsmp <- 100
```

```
# Slice Sampler (no derivatives needed)
beta.smp <- array(NA, dim=c(nsmp,K))
beta.tmp <- rep(0,K)
for (n in 1:nsmp) {
  beta.tmp <- MfU.Sample(beta.tmp
    , f=function(beta, X, y) loglike.logit(beta, X, y, fgh=0), X=X, y=y)
  beta.smp[n,] <- beta.tmp
}
beta.slice <- colMeans(beta.smp[(nsmp/2+1):nsmp,])

# Adaptive Rejection Sampler
# (only first derivative needed)
beta.smp <- array(NA, dim=c(nsmp,K))
beta.tmp <- rep(0,K)
for (n in 1:nsmp) {
  beta.tmp <- MfU.Sample(beta.tmp, uni.sampler="ars"
    , f=function(beta, X, y, grad) {
        if (grad)
          loglike.logit(beta, X, y, fgh=1)$g
        else
          loglike.logit(beta, X, y, fgh=0)
      }
    , X=X, y=y)
  beta.smp[n,] <- beta.tmp
}
beta.ars <- colMeans(beta.smp[(nsmp/2+1):nsmp,])

# SNS (Stochastic Newton Sampler)
# (both first and second derivative needed)
beta.smp <- array(NA, dim=c(nsmp,K))
beta.tmp <- rep(0,K)
for (n in 1:nsmp) {
  beta.tmp <- sns(beta.tmp, fghEval=loglike.logit, X=X, y=y, fgh=2)
  beta.smp[n,] <- beta.tmp
}
beta.sns <- colMeans(beta.smp[(nsmp/2+1):nsmp,])

# compare results
cbind(beta.glm, beta.slice, beta.ars, beta.sns)

## End(Not run)
```

`fbase1.exponential.log` *Single-Parameter Base Log-likelihood Function for Exponential GLM*

Description

Vectorized, single-parameter base log-likelihood functions for exponential GLM using log link function. The base function(s) can be supplied to the expander function `regfac.expand.1par` in order to obtain the full, high-dimensional log-likleihood and its derivatives.

Usage

```
fbase1.exponential.log(u, y, fgh=2)
```

Arguments

u	Varying parameter of the base log-likelihood function. This parameter is intended to be projected onto a high-dimensional space using the familiar regression transformation of `u <- X%*%beta`. In the typical use-case where the caller is `regfac.expand.1par`, a vector of values are supplied, and return objects will have the same length as `u`.
y	Fixed slot of the base distribution, corresponding to the response variable in the regression model. For `Poisson` family, it must be a vector of non-negative integers.
fgh	Integer with possible values 0,1,2. If `fgh=0`, the function only calculates and returns the log-likelihood vector and no derivatives. If `fgh=1`, it returns the log-likelihood and its first derivative in a list. If `fgh=2`, it returns the log-likelihood, as well as its first and second derivatives in a list.

Value

If `fgh==0`, the function returns `-u-y*exp(-u)` for `log`. If `fgh==1`, a list is returned with elements `f` and `g`, where the latter is a vector of length `length(u)`, with each element being the first derivative of the above expressions. If `fgh==2`, the list will include an element named `h`, consisting of the second derivatives of `f` with respect to `u`.

Author(s)

Alireza S. Mahani, Mansour T.A. Sharabiani

See Also

`regfac.expand.1par`

Examples

```
## Not run:
library(sns)
library(MfUSampler)

# using the expander framework and base distributions to define
# log-likelihood function for exponential regression
loglike.exponential <- function(beta, X, y, fgh) {
  regfac.expand.1par(beta, X, y, fbase1.exponential.log, fgh)
}

# generate data for exponential regression
N <- 1000
K <- 5
X <- matrix(runif(N*K, min=-0.5, max=+0.5), ncol=K)
beta <- runif(K, min=-0.5, max=+0.5)
```

```
y <- rexp(N, rate = exp(-X%*%beta))

# mcmc sampling of log-likelihood
nsmp <- 100

# Slice Sampler (no derivatives needed)
beta.smp <- array(NA, dim=c(nsmp,K))
beta.tmp <- rep(0,K)
for (n in 1:nsmp) {
  beta.tmp <- MfU.Sample(beta.tmp
    , f=loglike.exponential, X=X, y=y, fgh=0)
  beta.smp[n,] <- beta.tmp
}
beta.slice <- colMeans(beta.smp[(nsmp/2+1):nsmp,])

# Adaptive Rejection Sampler
# (only first derivative needed)
beta.smp <- array(NA, dim=c(nsmp,K))
beta.tmp <- rep(0,K)
for (n in 1:nsmp) {
  beta.tmp <- MfU.Sample(beta.tmp, uni.sampler="ars"
    , f=function(beta, X, y, grad) {
        if (grad)
          loglike.exponential(beta, X, y, fgh=1)$g
        else
          loglike.exponential(beta, X, y, fgh=0)
      }
    , X=X, y=y)
  beta.smp[n,] <- beta.tmp
}
beta.ars <- colMeans(beta.smp[(nsmp/2+1):nsmp,])

# SNS (Stochastic Newton Sampler)
# (both first and second derivative needed)
beta.smp <- array(NA, dim=c(nsmp,K))
beta.tmp <- rep(0,K)
for (n in 1:nsmp) {
  beta.tmp <- sns(beta.tmp, fghEval=loglike.exponential, X=X, y=y, fgh=2)
  beta.smp[n,] <- beta.tmp
}
beta.sns <- colMeans(beta.smp[(nsmp/2+1):nsmp,])

# compare results
cbind(beta, beta.slice, beta.ars, beta.sns)


## End(Not run)
```

---

fbase1.geometric.logit

Single-Parameter Base Log-likelihood Function for Exponential GLM

Description

Vectorized, single-parameter base log-likelihood functions for geometric GLM using logit link function. The base function(s) can be supplied to the expander function `regfac.expand.1par` in order to obtain the full, high-dimensional log-likleihood and its derivatives.

Usage

```
fbase1.geometric.logit(u, y, fgh=2)
```

Arguments

`u`	Varying parameter of the base log-likelihood function. This parameter is intended to be projected onto a high-dimensional space using the familiar regression transformation of `u <- X%*%beta`. In the typical use-case where the caller is `regfac.expand.1par`, a vector of values are supplied, and return objects will have the same length as `u`.
`y`	Fixed slot of the base distribution, corresponding to the response variable in the regression model. For `Geometric` family, it must be a vector of non-negative integers.
`fgh`	Integer with possible values 0,1,2. If `fgh=0`, the function only calculates and returns the log-likelihood vector and no derivatives. If `fgh=1`, it returns the log-likelihood and its first derivative in a list. If `fgh=2`, it returns the log-likelihood, as well as its first and second derivatives in a list.

Value

If `fgh==0`, the function returns `-(y*u+(1+y)*log(1+exp(-u)))` for `log`. If `fgh==1`, a list is returned with elements `f` and `g`, where the latter is a vector of length `length(u)`, with each element being the first derivative of the above expressions. If `fgh==2`, the list will include an element named `h`, consisting of the second derivatives of `f` with respect to `u`.

Note

The logit function must be applied to the probability parameter to give `X%*%beta`, which is in turn the inverse of the mean of the geometric distribution. For brevity, we still call the link function 'logit'.

Author(s)

Alireza S. Mahani, Mansour T.A. Sharabiani

See Also

`regfac.expand.1par`

Examples

```
## Not run:
library(sns)
library(MfUSampler)

# using the expander framework and base distributions to define
# log-likelihood function for geometric regression
loglike.geometric <- function(beta, X, y, fgh) {
  regfac.expand.1par(beta, X, y, fbase1.geometric.logit, fgh)
}

# generate data for geometric regression
N <- 1000
K <- 5
X <- matrix(runif(N*K, min=-0.5, max=+0.5), ncol=K)
beta <- runif(K, min=-0.5, max=+0.5)
y <- rgeom(N, prob = 1/(1+exp(-X%*%beta)))

# mcmc sampling of log-likelihood
nsmp <- 100

# Slice Sampler
beta.smp <- array(NA, dim=c(nsmp,K))
beta.tmp <- rep(0,K)
for (n in 1:nsmp) {
  beta.tmp <- MfU.Sample(beta.tmp
    , f=loglike.geometric, X=X, y=y, fgh=0)
  beta.smp[n,] <- beta.tmp
}
beta.slice <- colMeans(beta.smp[(nsmp/2+1):nsmp,])

# Adaptive Rejection Sampler
beta.smp <- array(NA, dim=c(nsmp,K))
beta.tmp <- rep(0,K)
for (n in 1:nsmp) {
  beta.tmp <- MfU.Sample(beta.tmp, uni.sampler="ars"
   , f=function(beta, X, y, grad) {
     if (grad)
       loglike.geometric(beta, X, y, fgh=1)$g
     else
       loglike.geometric(beta, X, y, fgh=0)
   }
   , X=X, y=y)
  beta.smp[n,] <- beta.tmp
}
beta.ars <- colMeans(beta.smp[(nsmp/2+1):nsmp,])

# SNS (Stochastic Newton Sampler)
beta.smp <- array(NA, dim=c(nsmp,K))
beta.tmp <- rep(0,K)
for (n in 1:nsmp) {
  beta.tmp <- sns(beta.tmp, fghEval=loglike.geometric, X=X, y=y, fgh=2, rnd = n>nsmp/4)
```

```
  beta.smp[n,] <- beta.tmp
}
beta.sns <- colMeans(beta.smp[(nsmp/2+1):nsmp,])

# compare sample averages with actual values
cbind(beta, beta.sns, beta.slice, beta.ars)

## End(Not run)
```

`fbase1.poisson.log` *Single-Parameter Base Log-likelihood Function for Poisson GLM*

Description

Vectorized, single-parameter base log-likelihood functions for Poisson GLM using log link function. The base function(s) can be supplied to the expander function `regfac.expand.1par` in order to obtain the full, high-dimensional log-likleihood and its derivatives.

Usage

```
fbase1.poisson.log(u, y, fgh=2)
```

Arguments

`u`	Varying parameter of the base log-likelihood function. This parameter is intended to be projected onto a high-dimensional space using the familiar regression transformation of `u <- X%*%beta`. In the typical use-case where the caller is `regfac.expand.1par`, a vector of values are supplied, and return objects will have the same length as `u`.
`y`	Fixed slot of the base distribution, corresponding to the response variable in the regression model. For `Poisson` family, it must be a vector of non-negative integers.
`fgh`	Integer with possible values 0,1,2. If `fgh=0`, the function only calculates and returns the log-likelihood vector and no derivatives. If `fgh=1`, it returns the log-likelihood and its first derivative in a list. If `fgh=2`, it returns the log-likelihood, as well as its first and second derivatives in a list.

Value

If `fgh==0`, the function returns `y*u-exp(u)-lfactorial(y)` for `log`. If `fgh==1`, a list is returned with elements `f` and `g`, where the latter is a vector of length `length(u)`, with each element being the first derivative of the above expressions. If `fgh==2`, the list will include an element named `h`, consisting of the second derivatives of `f` with respect to `u`.

Note

In all base log-likelihood functions, we have dropped any additive terms that are independent of the distribution parameter, e.g. constant terms or those terms that are dependent on the response variable only. This is done for computational efficiency. Therefore, these functions cannot be used to obtain the absolute values of log-likelihood functions but only in the context of optimization and/or sampling. Users can write thin wrappers around these functions to add the constant terms to the function value. (Derivatives do not need correction. For Poisson family, the `lfactorial(y)` term is dropped.)

Author(s)

Alireza S. Mahani, Mansour T.A. Sharabiani

See Also

`regfac.expand.1par`

Examples

```
## Not run:
library(sns)
library(MfUSampler)

# using the expander framework and base distributions to define
# log-likelihood function for Poisson regression
loglike.poisson <- function(beta, X, y, fgh) {
  regfac.expand.1par(beta, X, y, fbase1.poisson.log, fgh)
}

# generate data for Poisson regression
N <- 1000
K <- 5
X <- matrix(runif(N*K, min=-0.5, max=+0.5), ncol=K)
beta <- runif(K, min=-0.5, max=+0.5)
y <- rpois(N, lambda = exp(X%*%beta))

# obtaining glm coefficients for comparison
beta.glm <- glm(y~X-1, family="poisson")$coefficients

# mcmc sampling of log-likelihood
nsmp <- 100

# Slice Sampler (no derivatives needed)
beta.smp <- array(NA, dim=c(nsmp,K))
beta.tmp <- rep(0,K)
for (n in 1:nsmp) {
  beta.tmp <- MfU.Sample(beta.tmp
    , f=loglike.poisson, X=X, y=y, fgh=0)
  beta.smp[n,] <- beta.tmp
}
beta.slice <- colMeans(beta.smp[(nsmp/2+1):nsmp,])
```

```
# Adaptive Rejection Sampler
# (only first derivative needed)
beta.smp <- array(NA, dim=c(nsmp,K))
beta.tmp <- rep(0,K)
for (n in 1:nsmp) {
  beta.tmp <- MfU.Sample(beta.tmp, uni.sampler="ars"
    , f=function(beta, X, y, grad) {
        if (grad)
          loglike.poisson(beta, X, y, fgh=1)$g
        else
          loglike.poisson(beta, X, y, fgh=0)
      }
    , X=X, y=y)
  beta.smp[n,] <- beta.tmp
}
beta.ars <- colMeans(beta.smp[(nsmp/2+1):nsmp,])

# SNS (Stochastic Newton Sampler)
# (both first and second derivative needed)
beta.smp <- array(NA, dim=c(nsmp,K))
beta.tmp <- rep(0,K)
for (n in 1:nsmp) {
  beta.tmp <- sns(beta.tmp, fghEval=loglike.poisson, X=X, y=y, fgh=2, rnd = n>nsmp/4)
  beta.smp[n,] <- beta.tmp
}
beta.sns <- colMeans(beta.smp[(nsmp/2+1):nsmp,])

# compare results
cbind(beta.glm, beta.slice, beta.ars, beta.sns)

## End(Not run)
```

`fbase2.gamma.log.log` *Double-Parameter Base Log-likelihood Function for Gamma GLM*

Description

Vectorized, double-parameter base log-likelihood functions for Gamma GLM. The link functions map the mean and dispersion parameter to linear predictors. The base function can be supplied to the expander function `regfac.expand.2par` in order to obtain the full, high-dimensional log-likleihood and its derivatives.

Usage

```
fbase2.gamma.log.log(u, v, y, fgh = 2)
```

Arguments

u	First parameter of the base log-likelihood function (usually the result of applying a link function to distribution mean). This parameter is intended to be projected onto a high-dimensional space using the familiar regression transformation of `u <- X%*%beta`. In the typical use-case where the caller is `regfac.expand.2par`, a vector of values are supplied, and return objects will have the same length as `u` or `v`.
v	Second parameter of the base log-likelihood function (usually the result of applying a link function to distribution dispersion parameter). This parameter is intended to be projected onto a high-dimensional space using the familiar regression transformation of `v <- Z%*%gamma`. In the typical use-case where the caller is `regfac.expand.2par`, a vector of values are supplied, and return objects will have the same length as `u` or `v`.
y	Fixed slot of the base distribution, corresponding to the response variable in the regression model.
fgh	Integer with possible values 0,1,2. If `fgh=0`, the function only calculates and returns the log-likelihood vector and no derivatives. If `fgh=1`, it returns the log-likelihood and its first derivative in a list. If `fgh=2`, it returns the log-likelihood, as well as its first and second derivatives in a list.

Value

If `fgh==0`, the function returns `-exp(-v)*(u + y*exp(-u) + v - log(y)) - log(y) - log(gamma(exp(-v)))`. If `fgh==1`, a list is returned with elements `f` and `g`, where `f` is the same object as in `fgh==0` and `g` is a matrix of dimensions `length(u)`-by-2, with first column being the derivative of the above expression with respect to `u`, and the second column being the derivative of the above expression with respect to `v`. If `fgh==2`, the list will include an element named `h`, which is a matrix of dimensions `length(u)`-by-3, with the first column being the second derivative of `f` with respect to `u`, the second column being the second derivative of `f` with respect to `v`, and the third column is the cross-derivative term, i.e. the derivative of `f` with respect to `u` and `v`.

Author(s)

Alireza S. Mahani, Mansour T.A. Sharabiani

See Also

regfac.expand.2par

Examples

```
## Not run:
# we use this library for univariate slice sampling
# of multivariate distributions
library(MfUSampler)
library(dglm)

# simulating data according to assumed generative model
# we assume log link functions for both mean and dispersion
```

```
# given variance function V(mu) = mu^2, we have:
# log(mu) = X%*%beta
# log(phi) = X%*%gamma
N <- 10000
K <- 5
X <- cbind(1,matrix(runif(N*(K-1), min=-0.5, max=+0.5), ncol=K-1))
beta <- runif(K, min=0.0, max=+1.0)
gamma <- runif(K, min=0.0, max=+1.0)
shape.vec <- 1 / exp(X%*%gamma)
rate.vec <- 1 / exp(X%*%gamma + X%*%beta)
y <- rgamma(N, shape = shape.vec, rate = rate.vec)
# implied dispersion:
dispersion.vec <- 1 / shape.vec

# model estimation using dglm package
reg.dglm <- dglm(y~X-1, dformula = ~X-1, family=Gamma(link="log"), dlink = "log")
beta.dglm <- reg.dglm$coefficients
gamma.dglm <- reg.dglm$dispersion.fit$coefficients

# model estimation using RegressionFactory
# (with univariate slice sampling)
# defining the log-likelihood using the expander framework
# assumng same covariates for both slots, hence we set Z=X
# slice sampler does not need derivatives, hence we set fgh=0
loglike.gamma <- function(coeff, X, y) {
  regfac.expand.2par(coeff, X=X, Z=X, y=y, fbase2=fbase2.gamma.log.log, fgh=0)
}
nsmp <- 100
coeff.smp <- array(NA, dim=c(nsmp, 2*K))
coeff.tmp <- rep(0.1, 2*K)
for (n in 1:nsmp) {
  coeff.tmp <- MfU.Sample(coeff.tmp, f=loglike.gamma, X=X, y=y)
  coeff.smp[n,] <- coeff.tmp
}
beta.slice <- colMeans(coeff.smp[(nsmp/2+1):nsmp, 1:K])
gamma.slice <- colMeans(coeff.smp[(nsmp/2+1):nsmp, K+1:K])

# compare results
cbind(beta.dglm, beta.slice)
cbind(gamma.dglm, gamma.slice)


## End(Not run)
```

`fbase2.gaussian.identity.log`

Double-Parameter Base Log-likelihood Function for Gaussian GLM

Description

Vectorized, double-parameter base log-likelihood functions for Gaussian GLM. The link functions map the mean and variance to linear predictors. The base function can be supplied to the expander function `regfac.expand.2par` in order to obtain the full, high-dimensional log-likleihood and its derivatives.

Usage

```
fbase2.gaussian.identity.log(u, v, y, fgh = 2)
```

Arguments

`u`	First parameter of the base log-likelihood function (usually the mean). This parameter is intended to be projected onto a high-dimensional space using the familiar regression transformation of `u <- X%*%beta`. In the typical use-case where the caller is `regfac.expand.1par`, a vector of values are supplied, and return objects will have the same length as `u`.
`v`	Second parameter of the base log-likelihood function (usually the mean). This parameter is intended to be projected onto a high-dimensional space using the familiar regression transformation of `v <- Z%*%gamma`. In the typical use-case where the caller is `regfac.expand.1par`, a vector of values are supplied, and return objects will have the same length as `u`.
`y`	Fixed slot of the base distribution, corresponding to the response variable in the regression model. For `binomial` family, it must be an integer vector with values between `0` and `n`.
`fgh`	Integer with possible values 0,1,2. If `fgh=0`, the function only calculates and returns the log-likelihood vector and no derivatives. If `fgh=1`, it returns the log-likelihood and its first derivative in a list. If `fgh=2`, it returns the log-likelihood, as well as its first and second derivatives in a list.

Value

If `fgh==0`, the function returns `-0.5*(v + exp(-v)*(u-y)*(u-y))` (ignoring additive terms that are independent of `u,v`). It will therefore be of the same length as `u`. If `fgh==1`, a list is returned with elements `f` and `g`, where `f` is the same object as in `fgh==1` and `g` is a matrix of dimensions `length(u)`-by-2, with first column being the derivative of the above expression with respect to `u`, and the second column being the derivative of the above expression with respect to `v`. If `fgh==2`, the list will include an element named `h`, which is a matrix of dimensions `length(u)`-by-3, with the first column being the second derivative of `f` with respect to `u`, the second column being the second derivative of `f` with respect to `v`, and the third column is the cross-derivative term, i.e. the derivative of `f` with respect to `u` and `v`.

Author(s)

Alireza S. Mahani, Mansour T.A. Sharabiani

See Also

regfac.expand.2par

Examples

```
## Not run:
library(sns)
library(MfUSampler)
library(dglm)

# defining log-likelihood function
# vd==FALSE leads to constant-dispersion model (ordinary linear regression)
# while vd==TRUE produces varying-dispersion model
loglike.linreg <- function(coeff, X, y, fgh, block.diag = F, vd = F) {
  if (vd) regfac.expand.2par(coeff = coeff, X = X, Z = X, y = y
    , fbase2 = fbase2.gaussian.identity.log, fgh = fgh, block.diag = block.diag)
  else regfac.expand.2par(coeff = coeff, X = X, y = y
    , fbase2 = fbase2.gaussian.identity.log, fgh = fgh, block.diag = block.diag)
}

# simulating data according to generative model
N <- 1000 # number of observations
K <- 5 # number of covariates
X <- matrix(runif(N*K, min=-0.5, max=+0.5), ncol=K)
beta <- runif(K, min=-0.5, max=+0.5)
gamma <- runif(K, min=-0.5, max=+0.5)
mean.vec <- X%*%beta
sd.vec <- exp(X%*%gamma)
y <- rnorm(N, mean.vec, sd.vec)

# constant-dispersion model
# estimation using glm
est.glm <- lm(y~X-1)
beta.glm <- est.glm$coefficients
sigma.glm <- summary(est.glm)$sigma
# estimation using RegressionFactory
# (we set rnd=F in sns to allow for better comparison with glm)
nsmp <- 20
coeff.smp <- array(NA, dim=c(nsmp, K+1))
coeff.tmp <- rep(0, K+1)
for (n in 1:nsmp) {
  coeff.tmp <- sns(coeff.tmp, fghEval=loglike.linreg
    , X=X, y=y, fgh=2, block.diag = F, vd = F, rnd = F)
  coeff.smp[n,] <- coeff.tmp
}
beta.regfac.cd <- colMeans(coeff.smp[(nsmp/2+1):nsmp, 1:K])
sigma.regfac.cd <- sqrt(exp(mean(coeff.smp[(nsmp/2+1):nsmp, K+1])))
# comparing glm and RegressionFactory results
# beta's must match exactly between glm and RegressionFactory
cbind(beta, beta.glm, beta.regfac.cd)
# sigma's won't match exactly
cbind(mean(sd.vec), sigma.glm, sigma.regfac.cd)

# varying-dispersion model
# estimation using dglm
est.dglm <- dglm(y~X-1, dformula = ~X-1, family = "gaussian", dlink = "log")
```

```
beta.dglm <- est.dglm$coefficients
gamma.dglm <- est.dglm$dispersion.fit$coefficients
# estimation using RegressionFactory
coeff.smp <- array(NA, dim=c(nsmp, 2*K))
coeff.tmp <- rep(0, 2*K)
for (n in 1:nsmp) {
  coeff.tmp <- sns(coeff.tmp, fghEval=loglike.linreg
    , X=X, y=y, fgh=2, block.diag = F, vd = T, rnd = F)
  coeff.smp[n,] <- coeff.tmp
}
beta.regfac.vd <- colMeans(coeff.smp[(nsmp/2+1):nsmp, 1:K])
gamma.regfac.vd <- colMeans(coeff.smp[(nsmp/2+1):nsmp, K+1:K])
# comparing dglm and RegressionFactory results
# neither beta's nor gamma's will match exactly
cbind(beta, beta.dglm, beta.regfac.vd)
cbind(gamma, gamma.dglm, gamma.regfac.vd)


## End(Not run)
```

`fbase2.inverse.gaussian.log.log` *Double-Parameter Base Log-likelihood Function for Inverse-Gaussian GLM*

Description

Vectorized, double-parameter base log-likelihood functions for Inverse-Gaussian GLM. The link functions map the mean and dispersion parameter to linear predictors. The base function can be supplied to the expander function `regfac.expand.2par` in order to obtain the full, high-dimensional log-likleihood and its derivatives.

Usage

```
fbase2.inverse.gaussian.log.log(u, v, y, fgh = 2)
```

Arguments

u	First parameter of the base log-likelihood function (usually the result of applying a link function to distribution mean). This parameter is intended to be projected onto a high-dimensional space using the familiar regression transformation of `u <- X%*%beta`. In the typical use-case where the caller is `regfac.expand.2par`, a vector of values are supplied, and return objects will have the same length as `u` or `v`.
v	Second parameter of the base log-likelihood function (usually the result of applying a link function to distribution dispersion parameter). This parameter is intended to be projected onto a high-dimensional space using the familiar regression transformation of `v <- Z%*%gamma`. In the typical use-case where the caller is `regfac.expand.2par`, a vector of values are supplied, and return objects will have the same length as `u` or `v`.

y	Fixed slot of the base distribution, corresponding to the response variable in the regression model.
fgh	Integer with possible values 0,1,2. If `fgh=0`, the function only calculates and returns the log-likelihood vector and no derivatives. If `fgh=1`, it returns the log-likelihood and its first derivative in a list. If `fgh=2`, it returns the log-likelihood, as well as its first and second derivatives in a list.

Value

If `fgh==0`, the function returns `-v/2 - 0.5*exp(-v-2*u)*(y - exp(u))^2/y`. If `fgh==1`, a list is returned with elements `f` and `g`, where `f` is the same object as in `fgh==0` and `g` is a matrix of dimensions `length(u)`-by-2, with first column being the derivative of the above expression with respect to `u`, and the second column being the derivative of the above expression with respect to `v`. If `fgh==2`, the list will include an element named `h`, which is a matrix of dimensions `length(u)`-by-3, with the first column being the second derivative of `f` with respect to `u`, the second column being the second derivative of `f` with respect to `v`, and the third column is the cross-derivative term, i.e. the derivative of `f` with respect to `u` and `v`.

Author(s)

Alireza S. Mahani, Mansour T.A. Sharabiani

See Also

regfac.expand.2par

Examples

```
## Not run:
# we use this library for univariate slice sampling
# of multivariate distributions
library(MfUSampler)
library(dglm)

# simulating data according to assumed generative model
# we assume log link functions for both mean and dispersion
# (shape parameter is inverse of dispersion)
N <- 10000
K <- 5
X <- cbind(1,matrix(runif(N*(K-1), min=-0.5, max=+0.5), ncol=K-1))
beta <- runif(K, min=-0.5, max=+0.5)
gamma <- runif(K, min=-0.5, max=+0.5)
mean.vec <- exp(X %*% beta)
dispersion.vec <- exp(X %*% gamma)
y <- rinvgauss(N, mean = mean.vec, dispersion = dispersion.vec)

# model estimation using dglm package
reg.dglm <- dglm(y~X-1, dformula = ~X-1, family=inverse.gaussian(link="log"), dlink = "log")
beta.dglm <- reg.dglm$coefficients
gamma.dglm <- reg.dglm$dispersion.fit$coefficients
```

```
# model estimation using RegressionFactory
# (with univariate slice sampling)
# defining the log-likelihood using the expander framework
# assumng same covariates for both slots, hence we set Z=X
# slice sampler does not need derivatives, hence we set fgh=0
loglike.inverse.gaussian <- function(coeff, X, y) {
  regfac.expand.2par(coeff, X=X, Z=X, y=y, fbase2=fbase2.inverse.gaussian.log.log, fgh=0)
}
nsmp <- 100
coeff.smp <- array(NA, dim=c(nsmp, 2*K))
coeff.tmp <- rep(0.1, 2*K)
for (n in 1:nsmp) {
  coeff.tmp <- MfU.Sample(coeff.tmp, f=loglike.inverse.gaussian, X=X, y=y)
  coeff.smp[n,] <- coeff.tmp
}
beta.slice <- colMeans(coeff.smp[(nsmp/2+1):nsmp, 1:K])
gamma.slice <- colMeans(coeff.smp[(nsmp/2+1):nsmp, K+1:K])

# compare results
cbind(beta.dglm, beta.slice)
cbind(gamma.dglm, gamma.slice)


## End(Not run)
```

`regfac.expand.1par` *Expander Function for Single-Parameter Base Distributions*

Description

This function produces the full, high-dimensional gradient and Hessian from the base-distribution derivatives for linear transformations of the arguments of a single-parameter base distribution.

Usage

```
regfac.expand.1par(beta, X, y, fbase1, fgh=2, ...)
```

Arguments

`beta`	Vector of coefficients in the regression model.
`X`	Matrix of covariates in the regression model. Note that `ncol(X)` must be equal to `length(beta)`.
`y`	Vector of response variable in the regression model. Note that `length(y)` must be equal to `nrow(X)`.
`fbase1`	Base distribution function `fbase1(u, y, ...)` for the regression model. It must return a list with elements `f,g,h` corresponding to the function and its first and second derivatives relative to its first argument, `u`.

fgh	Integer with possible values 0,1,2. If fgh=0, the function only calculates and returns the log-likelihood function. If fgh=1, it returns the log-likelihood and its gradient vector. If fgh=2, it returns the log-likelihood, the gradient vector and the Hessian matrix.
...	Other parameters to be passed to fbase1.

Value

A list with elements f,g,h corresponding to the function, gradient vector, and Hessian matrix of the function fbase(X%*%beta,y), i.e. the base function fbase(u,y) projected onto the high-dimensional space of beta through the linear transformation of its first argument (u <- X%*%beta).

Author(s)

Alireza S. Mahani, Mansour T.A. Sharabiani

References

Mahani, Alireza S. and Sharabiani, Mansour T.A. (2013) *Metropolis-Hastings Sampling Using Multivariate Gaussian Tangents* https://arxiv.org/pdf/1308.0657v1.pdf

See Also

regfac.expand.2par

Examples

```
## Not run:
library(sns)
# simulating logistic regression data
N <- 1000 # number of observations
K <- 10 # number of variables
X <- matrix(runif(N*K, min=-0.5, max=+0.5), ncol=K)
beta <- runif(K, min=-0.5, max=+0.5)
Xbeta <- X%*%beta
y <- 1*(runif(N)<1/(1+exp(-Xbeta)))
beta.est <- rep(0,K)
# run sns in non-stochastic mode, i.e. Newton-Raphson optimization
for (i in 1:10) {
  beta.est <- sns(beta.est, regfac.expand.1par, rnd=F, X=X, y=y
    , fbase1=fbase1.binomial.logit)
}
# use glm to estimate beta and compare
beta.est.glm <- glm(y~X-1, family="binomial")$coefficients
cbind(beta.est, beta.est.glm)

## End(Not run)
```

regfac.expand.2par *Expander Function for Two-Parameter Base Distributions*

Description

This function produces the full, high-dimensional gradient and Hessian from the base-distribution derivatives for linear transformations of the arguments of a two-parameter base distribution.

Usage

```
regfac.expand.2par(coeff, X, Z=matrix(1.0, nrow=nrow(X), ncol=1)
  , y, fbase2, fgh=2, block.diag=FALSE, ...)
```

Arguments

coeff	Vector of coefficients in the regression model. The first ncol(X) elements correspond to the first parameter of the base distribution fbase2(u, v, y, ...), and the next ncol(Z) elements corresponds to the second parameter of the base distribution fbase2(u, v, y, ...).
X	Matrix of covariates corresponding to the first parameter of the base distribution fbase2(u, v, y, ...).
Z	Matrix of covariates corresponding to the second parameter of the base distribution fbase2(u, v, y, ...). Default is a single column of 1's, corresponding to an intercept-only model for the second parameter, i.e. assuming the second parameter is constant across all observations. Note that nrow(Z) must be equal to nrow(X).
y	Vector of response variables. Note that length(y) must be equal to nrow(X).
fbase2	Base distribution function fbase2(u, v, y, ...) for the regression model. It must return a list with elements f,g,h corresponding to the function and its first and second derivatives relative to its first two argument, u,v. The gradient must be a matrix of dimensions nrow(X)-by-2, where the first column is the gradient of the log-likelihood function with respect to its first parameter (fbase2_u), evaluated at each of the nrow(X) observations, and the second column is the gradient of the log-likelihood function with repsect to its second parameter (fbase2_v), also evaluated at each observation point. Similarly, the Hessian must be a matrix of dimensions nrow(X)-by-3, with elements being equal to fbase2_uu, fbase2_vv and fbase2_uv evaluated at each observation point (taking advantage of the Hessian being symmetric).
fgh	Integer with possible values 0,1,2. If fgh=0, the function only calculates and returns the log-likelihood function. If fgh=1, it returns the log-likelihood and its gradient vector. If fgh=2, it returns the log-likelihood, the gradient vector and the Hessian matrix.
block.diag	If TRUE, Hessian matrix is block-diagonalized by setting cross-terms between beta and gamma to zero. This can be useful if the full - i.e. non-block-diagonalized - Hessian is not negative definite, but block-diagonalization leads to definiteness.

	If TRUE, third element of the Hessian of fbase is not needed and thus it can be vector of length 2 instead of 3.
...	Other arguments to be passed to fbase2.

Value

A list with elements f, g, h corresponding to the function, gradient vector, and Hessian matrix of the function fbase2(X%*%beta, Z%*%gamma, y, ...), where beta=coeff[1:ncol(X)] and gamma=coeff[ncol(X)+1:ncol(Z (Derivatives are evaluated relative to coeff.) In other words, the base function fbase2(u, v, y, ...) is projected onto the high-dimensional space of c(beta, gamma) through the linear transformations of its first argument (u <- X%*%beta) and its second argument (v <- Z%*%gamma).

Author(s)

Alireza S. Mahani, Mansour T.A. Sharabiani

References

Mahani, Alireza S. and Sharabiani, Mansour T.A. (2013) *Metropolis-Hastings Sampling Using Multivariate Gaussian Tangents* https://arxiv.org/pdf/1308.0657v1.pdf

See Also

regfac.expand.1par

Examples

```
## Not run:
library(dglm)
library(sns)

# defining log-likelihood function
loglike.linreg <- function(coeff, X, y) {
  regfac.expand.2par(coeff = coeff, X = X, Z = X, y = y
    , fbase2 = fbase2.gaussian.identity.log, fgh = 2, block.diag = T)
}

# simulating data according to generative model
N <- 1000 # number of observations
K <- 5 # number of covariates
X <- matrix(runif(N*K, min=-0.5, max=+0.5), ncol=K)
beta <- runif(K, min=-0.5, max=+0.5)
gamma <- runif(K, min=-0.5, max=+0.5)
mean.vec <- X%*%beta
sd.vec <- exp(X%*%gamma)
y <- rnorm(N, mean.vec, sd.vec)

# estimation using dglm
est.dglm <- dglm(y~X-1, dformula = ~X-1, family = "gaussian", dlink = "log")
beta.dglm <- est.dglm$coefficients
gamma.dglm <- est.dglm$dispersion.fit$coefficients
```

```
# estimation using RegressionFactory
coeff.tmp <- rep(0, 2*K)
for (n in 1:10) {
  coeff.tmp <- sns(coeff.tmp, fghEval=loglike.linreg
    , X=X, y=y, rnd = F)
}
beta.regfac.vd <- coeff.tmp[1:K]
gamma.regfac.vd <- coeff.tmp[K+1:K]

# comparing dglm and RegressionFactory results
# neither beta's nor gamma's will match exactly
cbind(beta.dglm, beta.regfac.vd)
cbind(gamma.dglm, gamma.regfac.vd)

## End(Not run)
```

`regfac.merge` *Utility Function for Adding Two Functions and Their Derivatives*

Description

Combining two log-density functions by adding the corresponding elements of their lists (function, gradient, Hessian). This can be useful, e.g. in combining the likelihood and the prior (in log domain) to form the posterior according to Bayes rule.

Usage

```
regfac.merge(fgh1, fgh2, fgh = 2)
```

Arguments

`fgh1`	First log-density list, containing elements `f`, `g` and `h`, corresponding to log-density function, its gradient vector, and its Hessian matrix.
`fgh2`	Second log-density list, containing elements `f`, `g` and `h`, corresponding to log-density function, its gradient vector, and its Hessian matrix.
`fgh`	Integer flag with possible values `0`, `1`, `2`, indicating the maximum order of derivative to be returned.

Value

If `fgh==0`, `fgh1+fgh2` is returned. Otherwise, a list is returned with elements `f`, `g`, and `h`, each of which is the sum of corresponding elements of `fgh1` and `fgh2` lists.

Author(s)

Alireza S. Mahani, Mansour T.A. Sharabiani

Examples

```
# constructing the log-posterior for Bayesian logistic regression
# log-likelihood
loglike.logistic <- function(beta, X, y, fgh) {
  regfac.expand.1par(beta, X, y, fbase1.binomial.logit, fgh, n=1)
}
# log-prior
logprior.logistic <- function(beta, mu.beta, sd.beta, fgh) {
  f <- sum(dnorm(beta, mu.beta, sd.beta, log=TRUE))
  if (fgh==0) return (f)
  g <- -(beta-mu.beta)/sd.beta^2
  if (fgh==1) return (list(f=f, g=g))
  #h <- diag(rep(-1/sd.beta^2,length(beta)))
  h <- diag(-1/sd.beta^2)
  return (list(f=f, g=g, h=h))
}
# adding log-likelihood and log-prior according to Bayes rule
logpost.logistic <- function(beta, X, y, mu.beta, sd.beta, fgh) {
  ret.loglike <- loglike.logistic(beta, X, y, fgh)
  ret.logprior <- logprior.logistic(beta, mu.beta, sd.beta, fgh)
  regfac.merge(ret.loglike,ret.logprior, fgh=fgh)
}
```

Index