R documentation

of ‘LAPinfer-package.Rd’ etc.

March 19, 2016

`LAPinfer-package` *Laplace based Approximate Posterior Inference for Differential Equation Models*

Description

The `LAPinfer` package provides `R` functions for the Laplace based approximate posterior inference for the differential equation models.

Details

Package:	LAPinfer
Type:	Package
Version:	1.0
Date:	2016-03-19
License:	.

Author(s)

Kyoungjae Lee <leekjstat@gmail.com>

References

Sarat C. Dass, Jaeyong Lee, Kyoungjae Lee and Jonghun Park. (2016) “Laplace Based Approximate Posterior Inference for Differential Equation Models”, *Statistics and Computing*.

See Also

`Rcpp, RcppArmadillo, inline`

inc.base — *A character containing the basic functions for "LAP" package*

Description

A character containing the basic functions for "LAP" package. All functions are written in `C++` language.

Examples

```
data(inc.base)
## maybe str(inc.base) ; plot(inc.base) ...
```

GetX — *Numerical integration of state variable X*

Description

The function `GetX` returns a numerical integration of state variable X as a $p \times n$ matrix.

Usage

```
GetX(inc.fn, X, t, m, theta)
```

Arguments

`inc.fn`	A character containing the code for `DEf` (Differential equation function `f`), `df_dx` (Differentiation of `f` with respect to x) and `d2f_dx2` (Differentiation of `df_dx` with respect to x). The code for each function should be written in `C++` grammar. See `Details` and `Examples` for `DEf`, `df_dx` and `d2f_dx2`.
`X`	A $n \times p$ matrix of state variables. n is the number of observation, and p is the dimensionality of state variable. The first column of `X` should have the initial value $x_1 = x(t_1)$.
`t`	A n-dimensional vector containing the time points for observation.
`m`	A positive integer. Each interval $[t_{i-1}, t_i]$ is divided into m segments to form a refined time points. The state variable is successively approximated by a numerical integration (fourth order Runge-Kutta) on this time points.
`theta`	A vector containing the parameters of the differential equation.

Details

This function returns a numerical integration of state variable X as a $p \times n$ matrix using fourth order Runge-Kutta method.

A character input argument `inc.fn` should be made by the user. It specifies the differential equation and its derivatives for the state variable. For example, we consider the FitzHugh-Nagumo model in `Examples`:

$$\dot{x}_1(t) = f_1(x, t; \theta) = \theta_3(x_1(t) - \frac{1}{3}x_1^3(t) + x_2(t))$$

$$\dot{x}_2(t) = f_2(x,t;\theta) = -\frac{1}{\theta_3}(x_1(t) - \theta_1 + \theta_2 x_2(t)).$$

`df_dx` for the above differential equation is

$$\frac{\partial f_1(x,t;\theta)}{\partial x_1} = \theta_3(1 - x_1(t)^2),$$

$$\frac{\partial f_2(x,t;\theta)}{\partial x_1} = -\frac{1}{\theta_3},$$

$$\frac{\partial f_1(x,t;\theta)}{\partial x_2} = \theta_3,$$

$$\frac{\partial f_2(x,t;\theta)}{\partial x_2} = -\frac{\theta_2}{\theta_3},$$

and `d2f_dx2` for the above differential equation is

$$\frac{\partial^2 f_1(x,t;\theta)}{\partial x_1^2} = -2\frac{\theta_3}{x_1(t)},$$

$$\frac{\partial^2 f_i(x,t;\theta)}{\partial x_j^2} = 0$$

for all $i, j = 1, 2$ except $i = j = 1$.

The corresponding character input argument is `inc.FN` in `Examples`. `DEf` represents the differential equation at time t. `df_dx` and `d2f_dx2` represent the first and second derivatives of `DEf` *at all time points*. The `DEf` should be a p-dimensional vector. The `df_dx` should be a $p \times p \times (n-1)$ cube object whose (j_1, j_2, i)th element is

$$\frac{\partial f_{j_2}(x,t_i;\theta)}{\partial x_{j_1}}.$$

The `d2f_dx2` should be a $(n-1)$-dimensional field object whose ith element is $p \times p \times p$ cube. The (j_1, j_2, j_3)th element of `d2f_dx2(i)` is

$$\frac{\partial^2 f_{j_3}(x,t_i;\theta)}{\partial x_{j_1}\partial x_{j_2}}.$$

Value

The function `GetX` returns a numerical integration of state variable X as a $n \times p$ matrix.

Author(s)

Kyoungjae Lee

References

Sarat C. Dass, Jaeyong Lee, Kyoungjae Lee and Jonghun Park. (2016) "Laplace Based Approximate Posterior Inference for Differential Equation Models", *Statistics and Computing*.

Examples

```
library(inline)
library(RcppArmadillo)

# FitzHugn-Nagumo model
inc.FN <- '
using namespace Rcpp;
using namespace arma;

vec DEf(vec x, double t, vec theta){
int p = x.n_rows;
vec res(p);

res(0) = theta(2)*(x(0) - pow(x(0),3)/3. + x(1));
res(1) = -1./theta(2)*(x(0) - theta(0) + theta(1)*x(1));
return res;
}

cube df_dx(mat X, vec theta){
int n = X.n_cols;
int p = X.n_rows;
int i;
cube df_dxval(p,p,n-1);

df_dxval = df_dxval.zeros();
for(i=0; i<n-1; i++){
df_dxval.slice(i)(0,0) = theta(2)*(1 - pow(X(0,i),2));
df_dxval.slice(i)(0,1) = -1./theta(2);
df_dxval.slice(i)(1,0) = theta(2);
df_dxval.slice(i)(1,1) = -theta(1)/theta(2);
}
return df_dxval;
}

field<cube> d2f_dx2(mat X, vec theta){
int n = X.n_cols;
int p = X.n_rows;
int i;
cube zeroval(p,p,p);
field<cube> d2f_dx2val(n-1);

zeroval = zeroval.zeros();
for(i=0; i<n-1; i++){
d2f_dx2val(i) = zeroval;
d2f_dx2val(i).slice(0)(0,0) = -2*theta(2)*X(0,i);
}
return d2f_dx2val;
}
'

n = 100; p = 2; m = 100; h = 0.2; x1 = c(-1,1)
timept = seq(from = 0, by = h, length.out = n)
trueX = matrix(0, p,n); trueX[,1] = matrix(x1, 2, 1)
theta = c(0.2, 0.2, 3)
```

```
X = GetX(inc.FN, trueX, timept, m, theta)
```

`GetXhat` *Finding the state variable X minimizing the objective function using Newton-Raphson algorithm*

Description

The function `GetXhat` returns a numerical integration of state variable $X(\hat{x}_1(\theta), \theta)$ as a $p \times n$ matrix. Here, $\hat{x}_1(\theta)$ is

$$argmin_{x_1} n g_n(x_1, \theta) + \|x_1 - \mu_{x_1}\|^2/c$$

where $g_n(x_1, \theta) = \sum_{i=1}^n \|y_i - x_i(x_1, \theta)\|^2/n$ and $X(x_1, \theta) = (x_i(x_1, \theta))$ is a numerical integration of state variable with the initial state x_1 and the parameter θ. y_i denotes the i-th observation.

Usage

```
GetXhat(inc.fn, Y, t, m, theta, prior, eps = 0.1^5, maxiter = 50)
```

Arguments

`inc.fn`	A character containing the code for `DEf` (Differential equation function `f`), `df_dx` (Differentiation of `f` with respect to x) and `d2f_dx2` (Differentiation of `df_dx` with respect to x). The code for each function should be written in `C++` grammar. See `Details` and `Examples` for `DEf`, `df_dx` and `d2f_dx2`.
`Y`	A $p \times n$ matrix of observation. n is the number of observation, and p is the dimensionality of observation.
`t`	A n-dimensional vector containing the time points for observation.
`m`	A positive integer. Each interval $[t_{i-1}, t_i]$ is divided into m segments to form a refined time points. The state variable is successively approximated by a numerical integration (fourth order Runge-Kutta) on this time points.
`theta`	A vector containing the parameters of the differential equation.
`prior`	A list containing the following objects: `c` is a positive real number c. `mu_x1` is a p-dimensional vector μ_{x_1}.
`eps`	A positive real number. It determines the stopping rule of the Newton-Raphson algorithm. The default value is 0.1^5.
`maxiter`	A positive integer. It is the maximum iteration number of the Newton-Raphson algorithm. The default value is 50.

Details

This function returns returns a numerical integration of state variable $X(\hat{x}_1(\theta), \theta)$ as a $p \times n$ matrix.

A character input argument `inc.fn` should be made by the user. It specifies the differential equation and its derivatives for the state variable. For example, we consider the FitzHugh-Nagumo model in `Examples`:

$$\dot{x}_1(t) = f_1(x, t; \theta) = \theta_3(x_1(t) - \frac{1}{3}x_1^3(t) + x_2(t))$$

$$\dot{x}_2(t) = f_2(x, t; \theta) = -\frac{1}{\theta_3}(x_1(t) - \theta_1 + \theta_2 x_2(t)).$$

df_dx for the above differential equation is

$$\frac{\partial f_1(x,t;\theta)}{\partial x_1} = \theta_3(1-x_1(t)^2),$$

$$\frac{\partial f_2(x,t;\theta)}{\partial x_1} = -\frac{1}{\theta_3},$$

$$\frac{\partial f_1(x,t;\theta)}{\partial x_2} = \theta_3,$$

$$\frac{\partial f_2(x,t;\theta)}{\partial x_2} = -\frac{\theta_2}{\theta_3},$$

and d2f_dx2 for the above differential equation is

$$\frac{\partial^2 f_1(x,t;\theta)}{\partial x_1^2} = -2\frac{\theta_3}{x_1(t)},$$

$$\frac{\partial^2 f_i(x,t;\theta)}{\partial x_j^2} = 0$$

for all $i, j = 1, 2$ except $i = j = 1$.

The corresponding character input argument is inc.FN in Examples. DEf represents the differential equation at time t. df_dx and d2f_dx2 represent the first and second derivatives of DEf *at all time points*. The DEf should be a p-dimensional vector. The df_dx should be a $p \times p \times (n-1)$ cube object whose (j_1, j_2, i)th element is

$$\frac{\partial f_{j_2}(x, t_i;\theta)}{\partial x_{j_1}}.$$

The d2f_dx2 should be a $(n-1)$-dimensional field object whose ith element is $p \times p \times p$ cube. The (j_1, j_2, j_3)th element of d2f_dx2(i) is

$$\frac{\partial^2 f_{j_3}(x, t_i;\theta)}{\partial x_{j_1}\partial x_{j_2}}.$$

The stopping rule used in this function is

$$\frac{\|\hat{x}_1^{old} - \hat{x}_1^{new}\|_2}{\|\hat{x}_1^{old}\|_2} \le \texttt{eps} \quad \textit{or} \quad \frac{|g_n(\hat{x}_1^{old},\theta) - g_n(\hat{x}_1^{new},\theta)|}{g_n(\hat{x}_1^{old},\theta)} \le \texttt{eps}$$

where $\hat{x}_1^{old}$ and $\hat{x}_1^{old}$ are the estimated initial state from the last step and present step, respectively.

Value

The function GetXhat returns a list including the following objects:

Xhat	a $p \times n$ matrix giving the numerical integration of state variable X with the initial state $\hat{x}_1(\theta)$ and the parameter θ.
dfdx	a $p \times p \times (n-1)$ array giving the values of df_dx.
dxdx	a $p \times p \times n$ array giving the values of dx_dx, which represents the first derivatives of X with respect to x_1. The derivatives are computed by using the sensitivity equation for differential equation.

Author(s)

Kyoungjae Lee

References

Sarat C. Dass, Jaeyong Lee, Kyoungjae Lee and Jonghun Park. (2016) “Laplace Based Approximate Posterior Inference for Differential Equation Models”, *Statistics and Computing*.

Examples

```
library(inline)
library(RcppArmadillo)

# FitzHugn-Nagumo model
inc.FN <- '
using namespace Rcpp;
using namespace arma;

vec DEf(vec x, double t, vec theta){
int p = x.n_rows;
vec res(p);

res(0) = theta(2)*(x(0) - pow(x(0),3)/3. + x(1));
res(1) = -1./theta(2)*(x(0) - theta(0) + theta(1)*x(1));
return res;
}

cube df_dx(mat X, vec theta){
int n = X.n_cols;
int p = X.n_rows;
int i;
cube df_dxval(p,p,n-1);

df_dxval = df_dxval.zeros();
for(i=0; i<n-1; i++){
df_dxval.slice(i)(0,0) = theta(2)*(1 - pow(X(0,i),2));
df_dxval.slice(i)(0,1) = -1./theta(2);
df_dxval.slice(i)(1,0) = theta(2);
df_dxval.slice(i)(1,1) = -theta(1)/theta(2);
}
return df_dxval;
}

field<cube> d2f_dx2(mat X, vec theta){
int n = X.n_cols;
int p = X.n_rows;
int i;
cube zeroval(p,p,p);
field<cube> d2f_dx2val(n-1);

zeroval = zeroval.zeros();
for(i=0; i<n-1; i++){
d2f_dx2val(i) = zeroval;
d2f_dx2val(i).slice(0)(0,0) = -2*theta(2)*X(0,i);
}
return d2f_dx2val;
}
'

n = 100; p = 2; Ym = 100; h = 0.2; x1 = c(-1,1)
```

```
timept = seq(from = 0, by = h, length.out = n)
trueX = matrix(0, p,n); trueX[,1] = matrix(x1, 2, 1)
theta = c(0.2, 0.2, 3)
trueX = GetX(inc.FN, trueX, timept, Ym, theta)

# Get data set
library(mvtnorm)
sigma = 0.5
set.seed(2)
Y = trueX + t(rmvnorm(n, mean=rep(0,p), sigma=diag(p)*sigma^2))

prior = list()
prior$c = 100; prior$mu_x1 = Y[,1]
m = 1

Xhat = GetXhat(inc.FN, Y, timept, m, theta, prior)
```

LAPsetup *Defining a main function for the Laplace based Approximate Posterior inference*

Description

The function `LAPsetup` defines a main function for the Laplace based Approximate Posterior inference using `cxxfunction` in `Rcpp` package.

Usage

```
LAPsetup(inc.fn, method)
```

Arguments

`inc.fn`	A character containing the code for `DEf` (Differential equation function `f`), `df_dx` (Differentiation of `f` with respect to x) and `d2f_dx2` (Differentiation of `df_dx` with respect to x). The code for each function should be written in `C++` grammar. See `Details` and `Examples` for `DEf`, `df_dx` and `d2f_dx2`.
`method`	A character determining the method for posterior sampling for θ. `method`="grid" means the grid sampling, and except that, the other arguments mean the griddy Gibbs sampling.

Details

This function defines a main function for the Laplace based Approximate Posterior inference using `cxxfunction` in `Rcpp` package.

A character input argument `inc.fn` should be made by the user. It specifies the differential equation and its derivatives for the state variable. For example, we consider the FitzHugh-Nagumo model in `Examples`:

$$\dot{x}_1(t) = f_1(x,t;\theta) = \theta_3(x_1(t) - \frac{1}{3}x_1^3(t) + x_2(t))$$

$$\dot{x}_2(t) = f_2(x,t;\theta) = -\frac{1}{\theta_3}(x_1(t) - \theta_1 + \theta_2 x_2(t)).$$

`df_dx` for the above differential equation is

$$\frac{\partial f_1(x,t;\theta)}{\partial x_1} = \theta_3(1 - x_1(t)^2),$$

$$\frac{\partial f_2(x,t;\theta)}{\partial x_1} = -\frac{1}{\theta_3},$$

$$\frac{\partial f_1(x,t;\theta)}{\partial x_2} = \theta_3,$$

$$\frac{\partial f_2(x,t;\theta)}{\partial x_2} = -\frac{\theta_2}{\theta_3},$$

and `d2f_dx2` for the above differential equation is

$$\frac{\partial^2 f_1(x,t;\theta)}{\partial x_1^2} = -2\frac{\theta_3}{x_1(t)},$$

$$\frac{\partial^2 f_i(x,t;\theta)}{\partial x_j^2} = 0$$

for all $i, j = 1, 2$ except $i = j = 1$.

The corresponding character input argument is `inc.FN` in `Examples`. `DEf` represents the differential equation at time t. `df_dx` and `d2f_dx2` represent the first and second derivatives of `DEf` *at all time points*. The `DEf` should be a p-dimensional vector. The `df_dx` should be a $p \times p \times (n-1)$ cube object whose (j_1, j_2, i)th element is

$$\frac{\partial f_{j_2}(x, t_i; \theta)}{\partial x_{j_1}}.$$

The `d2f_dx2` should be a $(n-1)$-dimensional field object whose ith element is $p \times p \times p$ cube. The (j_1, j_2, j_3)th element of `d2f_dx2(i)` is

$$\frac{\partial^2 f_{j_3}(x, t_i; \theta)}{\partial x_{j_1} \partial x_{j_2}.}$$

Value

The function `LAPsetup` returns a function which will be used as an argument for `LAPinfer` function.

Author(s)

Kyoungjae Lee

References

Sarat C. Dass, Jaeyong Lee, Kyoungjae Lee and Jonghun Park. (2016) “Laplace Based Approximate Posterior Inference for Differential Equation Models”, *Statistics and Computing*.

Examples

```
library(inline)
library(RcppArmadillo)

# FitzHugn-Nagumo model
inc.FN <- '
using namespace Rcpp;
using namespace arma;
```

```
vec DEf(vec x, double t, vec theta){
int p = x.n_rows;
vec res(p);

res(0) = theta(2)*(x(0) - pow(x(0),3)/3. + x(1));
res(1) = -1./theta(2)*(x(0) - theta(0) + theta(1)*x(1));
return res;
}

cube df_dx(mat X, vec theta){
int n = X.n_cols;
int p = X.n_rows;
int i;
cube df_dxval(p,p,n-1);

df_dxval = df_dxval.zeros();
for(i=0; i<n-1; i++){
df_dxval.slice(i)(0,0) = theta(2)*(1 - pow(X(0,i),2));
df_dxval.slice(i)(0,1) = -1./theta(2);
df_dxval.slice(i)(1,0) = theta(2);
df_dxval.slice(i)(1,1) = -theta(1)/theta(2);
}
return df_dxval;
}

field<cube> d2f_dx2(mat X, vec theta){
int n = X.n_cols;
int p = X.n_rows;
int i;
cube zeroval(p,p,p);
field<cube> d2f_dx2val(n-1);

zeroval = zeroval.zeros();
for(i=0; i<n-1; i++){
d2f_dx2val(i) = zeroval;
d2f_dx2val(i).slice(0)(0,0) = -2*theta(2)*X(0,i);
}
return d2f_dx2val;
}
'

n = 100; p = 2; Ym = 100; h = 0.2; x1 = c(-1,1)
timept = seq(from = 0, by = h, length.out = n)
trueX = matrix(0, p,n); trueX[,1] = matrix(x1, 2, 1)
theta = c(0.2, 0.2, 3)
trueX = GetX(inc.FN, trueX, timept, Ym, theta)

# Get data set
library(mvtnorm)
sigma = 0.5
set.seed(2)
Y = trueX + t(rmvnorm(n, mean=rep(0,p), sigma=diag(p)*sigma^2))

####################################################################
# LAP setup: defining the function 'FN.grid' using cxxfunction
####################################################################
```

```
FN.grid = LAPsetup(inc.FN, method="grid")
```

`LAPinfer` *Laplace based Approximate Posterior inference*

Description

The function `LAPinfer` runs the Laplace based Approximate Posterior inference for the differential equation models.

Usage

```
LAPinfer(ftn, method, Y, t, m, grid, prior, hessian, checknum = 1000, nsample = NULL,
eps = 0.1^5, maxiter = 50)
```

Arguments

`ftn`	A function used for the Laplace based Approximate Posterior inference. It should be a output of `LAPsetup` function. See `Examples`.
`method`	A character determining the method for posterior sampling for θ. `method`="grid" means the grid sampling, and except that, the other arguments mean the griddy Gibbs sampling.
`Y`	A $p \times n$ matrix of observation. n is the number of observation, and p is the dimensionality of observation.
`t`	A n-dimensional vector containing the time points for observation.
`m`	A positive integer. Each interval $[t_{i-1}, t_i]$ is divided into m segments to form a refined time points. The state variable is successively approximated by a numerical integration (fourth order Runge-Kutta) on this time points.
`grid`	A list containing the following objects: `ctheta` is a q-dimensional vector indicating the center of the grid set. `MM` is a positive integer indicating the number of grid points for each axis. Each axis is divided into $2MM$ intervals of equal length. `h0` is a q-dimensional vector indicating the step size for each axis. If one choose `h0=rep(1,q)`, then the *standardized variables* z are used. Otherwise, if one choose $h_{0j} = k$, then $z_j k$ is used instead of z_j.
`prior`	A list containing the following objects: `a` is a positive real number a. `b` is a positive real number b. `c` is a positive real number c. `mu_x1` is a p-dimensional vector μ_{x_1}.
`hessian`	A list containing the following objects: `use` is a character determining whether to use the reparametrization technique for θ. Choose "No" for not using the reparametrization technique. Choose the other arguments for using the reparametrization technique. `maxiter` is a positive integer, which is the maximum iteration number of the Newton-Raphson algorithm for computing the numerical Hessian. `eps` is a positive real number, which determines the stopping rule of the Newton-Raphson algorithm for computing the numerical Hessian. `step` is a positive real number, which is the step size for the numerical Hessian.
`checknum`	A positive integer. `LAPinfer` reports on the screen when every `checknum` iterations have been carried out.

nsample	A positive integer giving the total number of posterior samples. It has no meaning when “grid” method is chosen.
eps	A positive real number. It determines the stopping rule of the Newton-Raphson algorithm. The default value is 0.1^5.
maxiter	A positive integer. It is the maximum iteration number of the Newton-Raphson algorithm. The default value is 50.

Details

This function runs the Laplace based Approximate Posterior inference for the differential equation models.

A character input argument `inc.fn` should be made by the user. It specifies the differential equation and its derivatives for the state variable. For example, we consider the FitzHugh-Nagumo model in `Examples`:

$$\dot{x}_1(t) = f_1(x,t;\theta) = \theta_3(x_1(t) - \frac{1}{3}x_1^3(t) + x_2(t))$$

$$\dot{x}_2(t) = f_2(x,t;\theta) = -\frac{1}{\theta_3}(x_1(t) - \theta_1 + \theta_2 x_2(t)).$$

`df_dx` for the above differential equation is

$$\frac{\partial f_1(x,t;\theta)}{\partial x_1} = \theta_3(1 - x_1(t)^2),$$

$$\frac{\partial f_2(x,t;\theta)}{\partial x_1} = -\frac{1}{\theta_3},$$

$$\frac{\partial f_1(x,t;\theta)}{\partial x_2} = \theta_3,$$

$$\frac{\partial f_2(x,t;\theta)}{\partial x_2} = -\frac{\theta_2}{\theta_3},$$

and `d2f_dx2` for the above differential equation is

$$\frac{\partial^2 f_1(x,t;\theta)}{\partial x_1^2} = -2\frac{\theta_3}{x_1(t)},$$

$$\frac{\partial^2 f_i(x,t;\theta)}{\partial x_j^2} = 0$$

for all $i, j = 1, 2$ except $i = j = 1$.

The corresponding character input argument is `inc.FN` in `Examples`. `DEf` represents the differential equation at time t. `df_dx` and `d2f_dx2` represent the first and second derivatives of `DEf` *at all time points*. The `DEf` should be a p-dimensional vector. The `df_dx` should be a $p \times p \times (n-1)$ cube object whose (j_1, j_2, i)th element is

$$\frac{\partial f_{j_2}(x,t_i;\theta)}{\partial x_{j_1}}.$$

The `d2f_dx2` should be a $(n-1)$-dimensional field object whose ith element is $p \times p \times p$ cube. The (j_1, j_2, j_3)th element of `d2f_dx2(i)` is

$$\frac{\partial^2 f_{j_3}(x,t_i;\theta)}{\partial x_{j_1}\partial x_{j_2}}.$$

Value

The function `LAPinfer` returns a list. The objects in the list depends on the chosen `method`. If the grid sampling was chosen, then the list includes the following objects:

`prob`	a $(2MM + 1)^q$-dimensional vector giving the posterior probabilities for θ on grid matrix.
`bstar`	a $(2MM + 1)^q$-dimensional vector whose j-th element is $u(\theta^j)/2 + b$ where $$u(\theta) = ng_n(\hat{x}_1(\theta), \theta) + \|\hat{x}_1(\theta) - \mu_{x_1}\|^2/c$$ and θ^j is the j-th grid for θ.
`grid mat`	a $q \times (2MM + 1)^q$ matrix whose j-th row is the j-th grid for θ.

If the griddy Gibbs sampling was chosen, then the list includes the following objects:

`sample`	a $q \times nsample$ matrix giving the posterior sample for θ.
`bstar`	a $q \times nsample$ matrix whose j-th element is $u(\theta^j)/2 + b$ where θ^j is the j-th posterior sample for θ.
`endval`	a q-dimensional vector giving the last value for the standardized variable z.

The common objects are

`Tmat`	a $q \times q$ transformation matrix $UD^{1/2}$ where $H^{-1} = UDU^T$ and H is the negative Hessian matrix of $\pi(\theta \mid y_n)$.
`Y`	same as input value.
`method`	same as input value.
`t`	same as input value.
`m`	same as input value.
`grid`	same as input value.
`prior`	same as input value.
`hessian`	same as input value.

Author(s)

Kyoungjae Lee

References

Sarat C. Dass, Jaeyong Lee, Kyoungjae Lee and Jonghun Park. (2016) “Laplace Based Approximate Posterior Inference for Differential Equation Models”, *Statistics and Computing*.

Examples

```
library(inline)
library(RcppArmadillo)

# FitzHugn-Nagumo model
inc.FN <- '
using namespace Rcpp;
using namespace arma;

vec DEf(vec x, double t, vec theta){
```

```
int p = x.n_rows;
vec res(p);

res(0) = theta(2)*(x(0) - pow(x(0),3)/3. + x(1));
res(1) = -1./theta(2)*(x(0) - theta(0) + theta(1)*x(1));
return res;
}

cube df_dx(mat X, vec theta){
int n = X.n_cols;
int p = X.n_rows;
int i;
cube df_dxval(p,p,n-1);

df_dxval = df_dxval.zeros();
for(i=0; i<n-1; i++){
df_dxval.slice(i)(0,0) = theta(2)*(1 - pow(X(0,i),2));
df_dxval.slice(i)(0,1) = -1./theta(2);
df_dxval.slice(i)(1,0) = theta(2);
df_dxval.slice(i)(1,1) = -theta(1)/theta(2);
}
return df_dxval;
}

field<cube> d2f_dx2(mat X, vec theta){
int n = X.n_cols;
int p = X.n_rows;
int i;
cube zeroval(p,p,p);
field<cube> d2f_dx2val(n-1);

zeroval = zeroval.zeros();
for(i=0; i<n-1; i++){
d2f_dx2val(i) = zeroval;
d2f_dx2val(i).slice(0)(0,0) = -2*theta(2)*X(0,i);
}
return d2f_dx2val;
}
'

n = 100; p = 2; Ym = 100; h = 0.2; x1 = c(-1,1)
timept = seq(from = 0, by = h, length.out = n)
trueX = matrix(0, p,n); trueX[,1] = matrix(x1, 2, 1)
theta = c(0.2, 0.2, 3)
trueX = GetX(inc.FN, trueX, timept, Ym, theta)

# Get data set
library(mvtnorm)
sigma = 0.5
set.seed(2)
Y = trueX + t(rmvnorm(n, mean=rep(0,p), sigma=diag(p)*sigma^2))

# LAP setup: defining the function <e2><80><98>FN.grid' using cxxfunction
FN.grid = LAPsetup(inc.FN, method="grid")

# setting the arguments
grid = list()
```

```
grid$ctheta = c(0.2, 0.2, 3); grid$MM = 15; grid$h0 = c(1, 1, 1)
prior = list()
prior$a = 0.1; prior$b = 0.01; prior$c = 100; prior$mu_x1 = Y[,1]
hessian = list()
hessian$use = "Yes"; hessian$maxiter = 500; hessian$eps = 0.1^3; hessian$step = 0.1^4

res = LAPinfer(FN.grid, method="grid", Y, timept, m, grid, prior, hessian, eps=0.1^5, maxiter=30)
```

`post.sampling` *Posterior sampling*

Description

The function `post.sampling` performs the posterior sampling for θ and σ^2. If the grid sampling was chosen for `res`, it conducts the grid sampling for θ on `res$gridmat` with probabilities `res$prob` and gets the posterior samples of σ^2 from $\pi(\sigma^2 \mid \theta, y_n)$. If the griddy Gibbs sampling was chosen for `res`, it selects a part of the posterior samples of θ using `post$nburn` and `post$nthin`, and gets the posterior samples of σ^2.

Usage

```
post.sampling(res, post, prior)
```

Arguments

`res`	A `LAP` object. It should be an output from `LAPinfer` function.
`post`	A list containing the following objects: `nburn` is the number of burn-in, and `nthin` is the thinning interval.
`prior`	A list containing the following objects: `a` is a positive real number a. `b` is a positive real number b. `c` is a positive real number c. `mu_x1` is a p-dimensional vector μ_{x_1}.

Details

This function performs the posterior sampling for θ and σ^2.

Value

The function `post.sampling` returns a list including the following objects:

`thetasamples`	a $q \times npost$ matrix containing the posterior samples of θ. $npost$ is the number of posterior samples.
`sigma2samples`	a $npost$-dimensional vector containing the posterior samples of σ^2.
`res`	same as input value.

Author(s)

Kyoungjae Lee

References

Sarat C. Dass, Jaeyong Lee, Kyoungjae Lee and Jonghun Park. (2016) “Laplace Based Approximate Posterior Inference for Differential Equation Models”, *Statistics and Computing*.

Examples

```
library(inline)
library(RcppArmadillo)

# FitzHugn-Nagumo model
inc.FN <- '
using namespace Rcpp;
using namespace arma;

vec DEf(vec x, double t, vec theta){
int p = x.n_rows;
vec res(p);

res(0) = theta(2)*(x(0) - pow(x(0),3)/3. + x(1));
res(1) = -1./theta(2)*(x(0) - theta(0) + theta(1)*x(1));
return res;
}

cube df_dx(mat X, vec theta){
int n = X.n_cols;
int p = X.n_rows;
int i;
cube df_dxval(p,p,n-1);

df_dxval = df_dxval.zeros();
for(i=0; i<n-1; i++){
df_dxval.slice(i)(0,0) = theta(2)*(1 - pow(X(0,i),2));
df_dxval.slice(i)(0,1) = -1./theta(2);
df_dxval.slice(i)(1,0) = theta(2);
df_dxval.slice(i)(1,1) = -theta(1)/theta(2);
}
return df_dxval;
}

field<cube> d2f_dx2(mat X, vec theta){
int n = X.n_cols;
int p = X.n_rows;
int i;
cube zeroval(p,p,p);
field<cube> d2f_dx2val(n-1);

zeroval = zeroval.zeros();
for(i=0; i<n-1; i++){
d2f_dx2val(i) = zeroval;
d2f_dx2val(i).slice(0)(0,0) = -2*theta(2)*X(0,i);
}
return d2f_dx2val;
}
'

n = 100; p = 2; Ym = 100; h = 0.2; x1 = c(-1,1)
```

```
timept = seq(from = 0, by = h, length.out = n)
trueX = matrix(0, p,n); trueX[,1] = matrix(x1, 2, 1)
theta = c(0.2, 0.2, 3)
trueX = GetX(inc.FN, trueX, timept, Ym, theta)

# Get data set
library(mvtnorm)
sigma = 0.5
set.seed(2)
Y = trueX + t(rmvnorm(n, mean=rep(0,p), sigma=diag(p)*sigma^2))

# LAP setup: defining the function 'FN.grid' using cxxfunction
FN.grid = LAPsetup(inc.FN, method="grid")

# setting the arguments
grid = list()
grid$ctheta = c(0.2, 0.2, 3); grid$MM = 15; grid$h0 = c(1, 1, 1)
prior = list()
prior$a = 0.1; prior$b = 0.01; prior$c = 100; prior$mu_x1 = Y[,1]
hessian = list()
hessian$use = "Yes"; hessian$maxiter = 500; hessian$eps = 0.1^3; hessian$step = 0.1^4

res = LAPinfer(FN.grid, method="grid", Y, timept, m, grid, prior, hessian, eps=0.1^5, maxiter=30)

post = list()
post$npost = 10000

###########################################################
# posterior sampling
###########################################################
samp = post.sampling(res, post, prior)
```

LAPpred *Prediction through the Laplace based Approximate Posterior inference*

Description

The function LAPpred returns a list of the predicted values for observations on present time points and future pred.size time points.

Usage

```
LAPpred(inc.fn, res, samp, pred.size, upper = 0.95, lower = 0.05, eps = 0.1^5, maxiter = 50)
```

Arguments

inc.fn	A character containing the code for DEf (Differential equation function f), df_dx (Differentiation of f with respect to x) and d2f_dx2 (Differentiation of df_dx with respect to x). The code for each function should be written in C++ grammar. See Details and Examples for DEf, df_dx and d2f_dx2.
res	A LAP object. It should be an output from LAPinfer function.
samp	A list of posterior samples. It should be an output from post.sampling function.

`pred.size`	A positive integer giving the number of future prediction time points.
`upper`	A positive real number between 0 and 1 for upper bound of predicted values. The default value is 0.95, which gives the 0.95 quantile of the predicted values.
`lower`	A positive real number between 0 and 1 for lower bound of predicted values. The default value is 0.05, which gives the 0.05 quantile of the predicted values.
`eps`	A positive real number. It determines the stopping rule of the Newton-Raphson algorithm. The default value is 0.1^5.
`maxiter`	A positive integer. It is the maximum iteration number of the Newton-Raphson algorithm. The default value is 50.

Details

This function returns a list of the predicted values for observations on present time points and future `pred.size` time points.

A character input argument `inc.fn` should be made by the user. It specifies the differential equation and its derivatives for the state variable. For example, we consider the FitzHugh-Nagumo model in `Examples`:

$$\dot{x}_1(t) = f_1(x,t;\theta) = \theta_3(x_1(t) - \frac{1}{3}x_1^3(t) + x_2(t))$$
$$\dot{x}_2(t) = f_2(x,t;\theta) = -\frac{1}{\theta_3}(x_1(t) - \theta_1 + \theta_2 x_2(t)).$$

`df_dx` for the above differential equation is

$$\frac{\partial f_1(x,t;\theta)}{\partial x_1} = \theta_3(1 - x_1(t)^2),$$
$$\frac{\partial f_2(x,t;\theta)}{\partial x_1} = -\frac{1}{\theta_3},$$
$$\frac{\partial f_1(x,t;\theta)}{\partial x_2} = \theta_3,$$
$$\frac{\partial f_2(x,t;\theta)}{\partial x_2} = -\frac{\theta_2}{\theta_3},$$

and `d2f_dx2` for the above differential equation is

$$\frac{\partial^2 f_1(x,t;\theta)}{\partial x_1^2} = -2\frac{\theta_3}{x_1(t)},$$
$$\frac{\partial^2 f_i(x,t;\theta)}{\partial x_j^2} = 0$$

for all $i, j = 1, 2$ except $i = j = 1$.

The corresponding character input argument is `inc.FN` in `Examples`. `DEf` represents the differential equation at time t. `df_dx` and `d2f_dx2` represent the first and second derivatives of `DEf` *at all time points*. The `DEf` should be a p-dimensional vector. The `df_dx` should be a $p \times p \times (n-1)$ cube object whose (j_1, j_2, i)th element is

$$\frac{\partial f_{j_2}(x,t_i;\theta)}{\partial x_{j_1}}.$$

The `d2f_dx2` should be a $(n-1)$-dimensional field object whose ith element is $p \times p \times p$ cube. The (j_1, j_2, j_3)th element of `d2f_dx2(i)` is

$$\frac{\partial^2 f_{j_3}(x,t_i;\theta)}{\partial x_{j_1}\partial x_{j_2}}.$$

Value

The function `LAPpred` returns a list. For any $j = 1, \ldots, p$, j-th object in the list is a $4 \times (n + pred.size)$ matrix whose first, second, third and fourth columns are 0.05, 0.5, 0.95 quantile of the predicted values and the observations for j-th state with `NA` for future time points.

Author(s)

Kyoungjae Lee

References

Sarat C. Dass, Jaeyong Lee, Kyoungjae Lee and Jonghun Park. (2016) "Laplace Based Approximate Posterior Inference for Differential Equation Models", *Statistics and Computing*.

Examples

```
library(inline)
library(RcppArmadillo)

# FitzHugn-Nagumo model
inc.FN <- '
using namespace Rcpp;
using namespace arma;

vec DEf(vec x, double t, vec theta){
int p = x.n_rows;
vec res(p);

res(0) = theta(2)*(x(0) - pow(x(0),3)/3. + x(1));
res(1) = -1./theta(2)*(x(0) - theta(0) + theta(1)*x(1));
return res;
}

cube df_dx(mat X, vec theta){
int n = X.n_cols;
int p = X.n_rows;
int i;
cube df_dxval(p,p,n-1);

df_dxval = df_dxval.zeros();
for(i=0; i<n-1; i++){
df_dxval.slice(i)(0,0) = theta(2)*(1 - pow(X(0,i),2));
df_dxval.slice(i)(0,1) = -1./theta(2);
df_dxval.slice(i)(1,0) = theta(2);
df_dxval.slice(i)(1,1) = -theta(1)/theta(2);
}
return df_dxval;
}

field<cube> d2f_dx2(mat X, vec theta){
int n = X.n_cols;
int p = X.n_rows;
int i;
cube zeroval(p,p,p);
field<cube> d2f_dx2val(n-1);
zeroval = zeroval.zeros();
for(i=0; i<n-1; i++){
d2f_dx2val(i) = zeroval;
d2f_dx2val(i).slice(0)(0,0) = -2*theta(2)*X(0,i);
}
return d2f_dx2val;
}
'

n = 100; p = 2; Ym = 100; h = 0.2; x1 = c(-1,1)
timept = seq(from = 0, by = h, length.out = n)
trueX = matrix(0, p,n); trueX[,1] = matrix(x1, 2, 1)
theta = c(0.2, 0.2, 3)
trueX = GetX(inc.FN, trueX, timept, Ym, theta)

# Get data set
library(mvtnorm)
sigma = 0.5
set.seed(2)
Y = trueX + t(rmvnorm(n, mean=rep(0,p), sigma=diag(p)*sigma^2))

# LAP setup: defining the function 'FN.grid' using cxxfunction
FN.grid = LAPsetup(inc.FN, method="grid")

# setting the arguments
grid = list()
grid$ctheta = c(0.2, 0.2, 3); grid$MM = 15; grid$h0 = c(1, 1, 1)
prior = list()
prior$a = 0.1; prior$b = 0.01; prior$c = 100; prior$mu_x1 = Y[,1]
hessian = list()
hessian$use = "Yes"; hessian$maxiter = 500; hessian$eps = 0.1^3; hessian$step = 0.1^4

res = LAPinfer(FN.grid, method="grid", Y, timept, m, grid, prior, hessian, eps=0.1^5, maxiter=30)

post = list()
post$npost = 10000
samp = post.sampling(res, post, prior)

##########################################################
# prediction
##########################################################
pred.size = 10
predmat = LAPpred(inc.FN, res, samp, pred.size)

pred.t = (1:(n+pred.size))*h
pred.trueX = matrix(0, p, n+pred.size)
pred.trueX[,1] = x1
pred.trueX = GetX(inc.FN, pred.trueX, pred.t, Ym, theta)

par(mfrow=c(1,2))
for(j in 1:p){
matplot(pred.t, cbind(predmat[[j]], pred.trueX[j,]), type=c("l","l","l","p","l"), pch="*",
xlab="Time (min)", ylab="Temperature", main="",lwd = 1, lty = c(2,2,2,1,1), bty = "l", col = c(4,3,4,1,2))
}
```

`hist.LAP` *Histogram for LAP objects*

Description

The `hist` method for `LAP` objects.

Usage

```
## S3 method for class LAP
hist.LAP(postsamples, size = NULL)
```

Arguments

`postsamples`	a `LAP` object.
`size`	a 2-dimensional vector giving the number of subplots on each axis.

Value

None.

Author(s)

Kyoungjae Lee

Examples

```
## The function is currently defined as
function (postsamples, size = NULL)
{
    q = nrow(postsamples$thetasamples)
    if (is.null(size))
        size = c(min(1 + floor(q/3), 3), 3)
    par(mfrow = size)
    for (i in 1:q) {
        ind = paste0("theta", i)
        minval = min(postsamples$thetasamples[i, ])
        maxval = max(postsamples$thetasamples[i, ])
        ll = length(postsamples$thetasamples[i, ])
        if (postsamples$res$method == "grid") {
            range = range(postsamples$res$gridmat[i, ])
        }
        else {
            range = c(minval, maxval)
        }
        hist(postsamples$thetasamples[i, ], breaks = seq(minval,
            maxval, by = (maxval - minval)/(ll - 1)), main = "",
            xlab = c(ind), xlim = range)
    }
    hist(samp$sigma2samples, main = "", xlab = "sigma2")
  }
```

`plot.LAP` *Plot for LAP objects*

Description

The `plot` method for `LAP` objects.

Usage

```
## S3 method for class LAP
plot.LAP(postsamples, coin = FALSE, size = NULL)
```

Arguments

`postsamples`	a `LAP` object.
`coin`	A logical value. If `TRUE`, the *standardized variables*, z, are plotted instead of θ. See the subsection 3.4 of Dass et al. (2016) for the details on the standardized variables z.
`size`	a 2-dimensional vector giving the number of subplots on each axis.

Author(s)

Kyoungjae Lee

Examples

```
## The function is currently defined as
function (postsamples, coin = FALSE, size = NULL)
{
    q = nrow(postsamples$thetasamples)
    if (is.null(size))
        size = c(min(1 + floor(q/3), 3), 3)
    if (coin) {
        inv_trans = diag(1/postsamples$res$grid$h0) %*% solve(postsamples$res$Tmat)
        coins = inv_trans %*% (postsamples$thetasamples - postsamples$res$grid$ctheta)
        truecoin = inv_trans %*% (theta - postsamples$res$grid$ctheta)
    }
    par(mfrow = size)
    for (i in 1:q) {
        if (coin) {
            ind = paste0("coin for theta", i)
            range = c(-4, 4)
            ts.plot(coins[i, ], main = "", xlab = c(ind), ylim = range,
                ylab = "")
        }
        else {
            ind = paste0("theta", i)
            ts.plot(postsamples$thetasamples[i, ], main = "",
                xlab = c(ind), ylab = "")
        }
    }
    ts.plot(postsamples$sigma2samples, main = "", xlab = "sigma2",
        ylab = "")
  }
```

summary.LAP *Summary for LAP objects*

Description

The summary method for LAP objects.

Usage

```
## S3 method for class LAP
summary.LAP(postsamples)
```

Arguments

postsamples	a LAP object.

Author(s)

Kyoungjae Lee

Examples

```
## The function is currently defined as
function (postsamples)
{
    q = nrow(postsamples$thetasamples)
    res = matrix(0, q + 1, 4)
    colnames(res) <- c("Mean", "Median", "5% quantile", "95% quantile")
    rownames(res) = rep(0, q + 1)
    for (i in 1:q) {
        res[i, 1] = mean(postsamples$thetasamples[i, ])
        res[i, 2] = median(postsamples$thetasamples[i, ])
        res[i, c(3:4)] = quantile(postsamples$thetasamples[i,
            ], c(0.05, 0.95))
        rownames(res)[i] = paste("theta", i, collapsed = "")
    }
    res[q + 1, 1] = mean(postsamples$sigma2samples)
    res[q + 1, 2] = median(postsamples$sigma2samples)
    res[q + 1, c(3:4)] = quantile(postsamples$sigma2samples,
        c(0.05, 0.95))
    rownames(res)[q + 1] = paste("sigma2")
    return(res)
  }
```

Index

∗Topic **datasets**
    inc.base, 2
∗Topic **package**
    LAPinfer-package, 1

GetX, 2
GetXhat, 5

hist.LAP, 21

inc.base, 2

LAPinfer, 11
LAPinfer *(*LAPinfer-package*)*, 1
LAPinfer-package, 1
LAPpred, 17
LAPsetup, 8

plot.LAP, 22
post.sampling, 15

summary.LAP, 23