Package 'PGAS'

July 25, 2018

Type Package

Title Particle Gibbs with Ancestor Sampling

Version 1.0.0

Author Niharika Gauraha

Maintainer Niharika Gauraha <niharika.gauraha@farmbio.uu.se>

Description Implementation of Particle Gibbs with Ancestor Sampling (see Svensson et al., 2014) and the Particle Marginal Metropolis-Hastings (see Andrieu et. al., 2010) algorithms, see (<http://www.it.uu.se/katalog/freli660/software>) for implementation details in 'MATLAB'.

Depends invgamma, smcUtils

License GPL-3

Encoding UTF-8

LazyData true

R topics documented:

`An Introduction of the package PGAS` *Implementation of Particle Gibbs with Ancestor Sampling and and the Particle Marginal Metropolis-Hastings algorithms*

Description

Implementation of Particle Gibbs with Ancestor Sampling ([1]) and the Particle Marginal Metropolis-Hastings ([2]) algorithms. See (<http://www.it.uu.se/katalog/freli660/software>) for implementation in 'MATLAB'.

Details

This package offeres a inverse gamma prior based solution for the PGAS and PMMH algorithms. In future, more generalized priors are planned to be supported. We refer to [1] for theoretical details of PGAS, [2] for details on PMMH algorithm and [3] for implementation details of PGAS and PMMH in MATLAB.

References

[1] Lindsten, Fredrik, Michael I. Jordan, and Thomas B. Schön. "Particle Gibbs with ancestor sampling." The Journal of Machine Learning Research 15.1 (2014): 2145-2184.

[2] C. Andrieu, A. Doucet and R. Holenstein, "Particle Markov chain Monte Carlo methods" Journal of the Royal Statistical Society: Series B, 2010, 72, 269-342.

[3] Lindsten, Fredrik, http://www.it.uu.se/katalog/freli660/software

---

conditionalParticleFilter

An implementation of conditional particle filter with ancestor sampling

---

Description

An implementation of conditional particle filter with ancestor sampling

Usage

```
conditionalParticleFilter(param, y, x0, X, N = 100, resamplingMethod = "multi")
```

Arguments

`param`	state parameters
`y`	measurements
`x0`	initial state
`X`	conditioned particles
`N`	number of particles
`resamplingMethod`	resampling methods: 'multi' multinomial and 'systematic' systematic resampling methods are currently supported.

Value

`particles`	particles
`logLikelihood`	log Likelihood
`normalisedWeights`	normalised Weights

Author(s)

Niharika Gauraha

See Also

PGAS, PMMH, particleFilter

Examples

```
generateData <- function(param, x0, T)
{
  #Initialize the state parameters
  f <- param$f # state transition function
  g <- param$g # tranfer function
  Q <- param$Q # process noise variance
  R <- param$R # measurement noise variance

  x = rep(0, T)
  y = rep(0, T)
  x[1] = x0  # Initial state

  for(t in 1:T)
  {
    if(t < T)
    {
      x[t+1] = stateTransFunc(x[t],t) + sqrt(Q)*rnorm(1)
    }
    y[t] = transferFunc(x[t]) + sqrt(R)*rnorm(1)
  }
  return(list(x = x, y = y))
}

stateTransFunc = function(xt, t)  0.5*xt + 25*xt/(1+xt^2) + 8*cos(1.2*t)
  transferFunc = function(x) x^2/20

  # Set up some parameters
  T = 100                # Length of data record

  # Generate data
  Q = 0.1  # True process noise variance
  R = 1 # True measurement noise variance
  param <- list(f = stateTransFunc, g = transferFunc, Q = Q, R = R)
  res = generateData(param = param, x0 = 0, T = T)
  x <- res$x
```

```
y <- res$y

cat("First plot true states and observed states ")
#p <-plot_ly(x = c(1:T), y = x,
#            name = 'Real States', type = 'scatter', mode = 'lines+markers')
#add_lines(p, x = c(1:T), y = y,
#          name = 'Observed States', type = 'scatter', mode = 'lines+markers')

cat("Running conditional particle filter ")
param <- list(f = stateTransFunc, g = transferFunc, Q = Q, R = R)
res = conditionalParticleFilter(param = param, y = y, x0 = 0, X = x, N = 100)
J <- which(runif(1) < cumsum(res$w[,T]))[1]
#p <-plot_ly(x = c(1:T), y = x,
#            name = 'Real States', type = 'scatter', mode = 'lines     +markers')
#add_lines(p, x = c(1:T), y = res$particles[J,],
#         name = 'CPF_AS Filtered States', type = 'scatter', mode =    'lines+markers')
```

`particleFilter` *An implementation of particle filter*

Description

An implementation of particle filter

Usage

```
particleFilter(param, y, x0, N = 100, resamplingMethod = "multi")
```

Arguments

`param`	state parameters
`y`	measurements
`x0`	initial state
`N`	number of particles
`resamplingMethod`	resampling methods: 'multi' multinomial and 'systematic' systematic resampling methods are currently supported.

Value

`particles`	particles
`logLikelihood`	log Likelihood
`normalisedWeights`	normalised Weights

Author(s)

Niharika Gauraha

See Also

PGAS, particleFilter, conditionalParticleFilter

Examples

```
generateData <- function(param, x0, T)
{
  #Initialize the state parameters
  f <- param$f # state transition function
  g <- param$g # tranfer function
  Q <- param$Q # process noise variance
  R <- param$R # measurement noise variance

  x = rep(0, T)
  y = rep(0, T)
  x[1] = x0  # Initial state

  for(t in 1:T)
  {
    if(t < T)
    {
      x[t+1] = stateTransFunc(x[t],t) + sqrt(Q)*rnorm(1)
    }
    y[t] = transferFunc(x[t]) + sqrt(R)*rnorm(1)
  }
  return(list(x = x, y = y))
}

stateTransFunc = function(xt, t)  0.5*xt + 25*xt/(1+xt^2) + 8*cos(1.2*t)
  transferFunc = function(x) x^2/20

  # Set up some parameters
  T = 100                # Length of data record

  # Generate data
  Q = 0.1  # True process noise variance
  R = 1 # True measurement noise variance
  param <- list(f = stateTransFunc, g = transferFunc, Q = Q, R = R)
  res = generateData(param = param, x0 = 0, T = T)
  x <- res$x
  y <- res$y

  #cat("First plot true states and observed states ")
  #p <-plot_ly(x = c(1:T), y = x,
  #            name = 'Real States', type = 'scatter', mode = 'lines+markers')
  #add_lines(p, x = c(1:T), y = y,
  #          name = 'Observed States', type = 'scatter', mode = 'lines+markers')
```

```
# Run the particle filter
cat("Running particle filter ")
param <- list(f = stateTransFunc, g = transferFunc, Q = Q, R = R)
res = particleFilter(param = param, y = y, x0 = 0, N = 100)
#p <-plot_ly(x = c(1:T), y = x,
#            name = 'Real States', type = 'scatter', mode = 'lines+markers')
#J <- which(runif(1) < cumsum(res$w[,T]))[1]
#add_lines(p, x = c(1:T), y = res$particles[J,],
#          name = 'Filtered States', type = 'scatter', mode = 'lines+markers')
```

PGAS *An Implementation of Particle Gibbs with ancestor sampling*

Description

An Implementation of Particle Gibbs with ancestor sampling using inverse gamma priors on noise variables.

Usage

```
PGAS(param, y, x0, prior, M = 1000, N = 100, resamplingMethod = "multi")
```

Arguments

`param`	state parameters
`y`	measurements
`x0`	initial state
`prior`	Hyperparameters for the inverse gamma priors (uninformative)
`M`	number of MCMC runs
`N`	number of particles
`resamplingMethod`	resampling methods: 'multi' multinomial and 'systematic' systematic resampling methods are currently supported.

Value

`q`	Sample path of the process noise Q
`r`	Sample path of the measurement noise R
`x`	Sample path of the states

Author(s)

Niharika Gauraha

References

[1] Lindsten, Fredrik, Michael I. Jordan, and Thomas B. Schön. "Particle Gibbs with ancestor sampling." The Journal of Machine Learning Research 15.1 (2014): 2145-2184.

See Also

PMMH, particleFilter, conditionalParticleFilter

Examples

```
generateData <- function(param, x0, T)
{
  #Initialize the state parameters
  f <- param$f # state transition function
  g <- param$g # tranfer function
  Q <- param$Q # process noise variance
  R <- param$R # measurement noise variance

  x = rep(0, T)
  y = rep(0, T)
  x[1] = x0  # Initial state

  for(t in 1:T)
  {
    if(t < T)
    {
      x[t+1] = stateTransFunc(x[t],t) + sqrt(Q)*rnorm(1)
    }
    y[t] = transferFunc(x[t]) + sqrt(R)*rnorm(1)
  }
  return(list(x = x, y = y))
}

stateTransFunc = function(xt, t)  0.5*xt + 25*xt/(1+xt^2) + 8*cos(1.2*t)
  transferFunc = function(x) x^2/20

  # Set up some parameters
  N1 = 5                # Number of particles used in PGAS
  N2 = 500              # Number of particles used in PMMH
  T = 100               # Length of data record
  numMCMC = 3000        # Number of iterations in the MCMC samplers
  burnin = 300          # Number of interations to burn

  # Generate data
  Q = 0.1  # True process noise variance
  R = 1 # True measurement noise variance
  param <- list(f = stateTransFunc, g = transferFunc, Q = Q, R = R)
  res = generateData(param = param, x0 = 0, T = T)
  x <- res$x
  y <- res$y

  # Hyperparameters for the inverse gamma priors (uninformative)
```

```
prior = c(0.01, 0.01)

cat("First plot true states and observed states ")
#p <-plot_ly(x = c(1:T), y = x,
#            name = 'Real States', type = 'scatter', mode = 'lines+markers')
#add_lines(p, x = c(1:T), y = y,
#          name = 'Observed States', type = 'scatter', mode = 'lines+markers')

cat("Running PGAS : ")
param <- list(f = stateTransFunc, g = transferFunc, Q = 1, R = 0.1)
res = PGAS(param, y, x0 = 0, prior = prior, M = numMCMC, N = N1)
#p <-plot_ly(x = c(1:T), y = x,
#            name = 'Real States', type = 'scatter', mode = 'lines+markers')
#add_lines(p, x = c(1:T), y = res$x[N1,], name = 'PGAS States',
#          type = 'scatter', mode = 'lines+markers')
# plot histrograma of the process noise variance and the measurement variance
hist(res$q[burnin:numMCMC], main = "Distribution of the process noise variance", freq = FALSE)
hist(res$r[burnin:numMCMC], main = "Distribution of the measurement noise variance", freq = FALSE)
```

PMMH *An Implementation of the Particle Marginal Metropolis-Hastings algorithm.*

Description

An Implementation of the Particle Marginal Metropolis-Hastings algorithm using inverse gamma priors on noise variables, and Gaussian random walk proposals.

Usage

```
PMMH(param, y, x0, prior, prop, M = 1000, N = 100, resamplingMethod = "multi")
```

Arguments

param	state parameters
y	measurements
x0	initial state
prior	Hyperparameters for the inverse gamma priors (uninformative)
prop	Proposal for PMMH (Gaussian random walk)
M	number of MCMC runs
N	number of particles
resamplingMethod	resampling methods: 'multi' multinomial and 'systematic' systematic resampling methods are currently supported.

Value

q	Sample path of the process noise Q
r	Sample path of the measurement noise R
x	Sample path of the states

Author(s)

Niharika Gauraha

References

[1] C. Andrieu, A. Doucet and R. Holenstein, "Particle Markov chain Monte Carlo methods" Journal of the Royal Statistical Society: Series B, 2010, 72, 269-342.

See Also

PGAS, particleFilter, conditionalParticleFilter

Examples

```
generateData <- function(param, x0, T)
{
  #Initialize the state parameters
  f <- param$f # state transition function
  g <- param$g # tranfer function
  Q <- param$Q # process noise variance
  R <- param$R # measurement noise variance

  x = rep(0, T)
  y = rep(0, T)
  x[1] = x0  # Initial state

  for(t in 1:T)
  {
    if(t < T)
    {
      x[t+1] = stateTransFunc(x[t],t) + sqrt(Q)*rnorm(1)
    }
    y[t] = transferFunc(x[t]) + sqrt(R)*rnorm(1)
  }
  return(list(x = x, y = y))
}

stateTransFunc = function(xt, t)  0.5*xt + 25*xt/(1+xt^2) + 8*cos(1.2*t)
  transferFunc = function(x) x^2/20

  # Set up some parameters
  N1 = 5                # Number of particles used in PGAS
  N2 = 500              # Number of particles used in PMMH
  T = 100               # Length of data record
  numMCMC = 3000        # Number of iterations in the MCMC samplers
```

```
burnin = 300          # Number of interations to burn

# Generate data
Q = 0.1  # True process noise variance
R = 1 # True measurement noise variance
param <- list(f = stateTransFunc, g = transferFunc, Q = Q, R = R)
res = generateData(param = param, x0 = 0, T = T)
x <- res$x
y <- res$y

# Hyperparameters for the inverse gamma priors (uninformative)
prior = c(0.01, 0.01)

cat("First plot true states and observed states ")
#p <-plot_ly(x = c(1:T), y = x,
#            name = 'Real States', type = 'scatter', mode = 'lines+markers')
#add_lines(p, x = c(1:T), y = y,
#          name = 'Observed States', type = 'scatter', mode = 'lines+markers')

cat("Running PMMH : ")
# Proposal for PMMH (Gaussian random walk)
prop = c(.1, .1)
param <- list(f = stateTransFunc, g = transferFunc, Q = .1, R = 1)
res = PMMH(param, y, x0 = 0, prior, prop, N = N2, M = numMCMC)
# p <-plot_ly(x = c(1:T), y = x,
#            name = 'Real States', type = 'scatter', mode = 'lines  +markers')
# add_lines(p, x = c(1:T), y = res$x[N2,],
          name = 'PMMH States', type = 'scatter', mode = 'lines+markers')
#plot histrograms of the process noise variance and the measurement variance
hist(res$q[burnin:numMCMC], main = "Distribution of the process noise variance", freq = FALSE)
hist(res$r[burnin:numMCMC], main = "Distribution of the measurement noise variance", freq = FALSE)
```

Index