Package ‘Rdtq’

November 22, 2016

Type Package

Title Density Tracking by Quadrature

Version 0.1

Date 2016-11-21

Author Harish S. Bhat, R. W. M. A. Madushani, Shagun Rawat

Maintainer Harish S. Bhat <hbhat@ucmerced.edu>

Description Implementation of density tracking by quadrature (DTQ) algorithms for stochastic differential equations (SDEs). DTQ algorithms numerically compute the density function of the solution of an SDE with user-specified drift and diffusion functions. The calculation does not require generation of sample paths, but instead proceeds in a deterministic fashion by repeatedly applying quadrature to the Chapman-Kolmogorov equation associated with a discrete-time approximation of the SDE. The DTQ algorithm is provably convergent. For several practical problems of interest, we have found the DTQ algorithm to be fast, accurate, and easy to use.

Depends R (>= 3.2.0)

License GPL (>= 2)

Imports Rcpp (>= 0.12.4), Matrix (>= 1.2)

Suggests ggplot2 (>= 2.1), scales (>= 0.4.0)

LinkingTo Rcpp

LazyData true

NeedsCompilation yes

Repository CRAN

Date/Publication 2016-11-22 09:06:51

R topics documented:

`rdtq` *Density Tracking by Quadrature*

Description

`rdtq` implements density tracking by quadrature (DTQ) algorithms to compute the probability density function of a stochastic differential equation with user-specified drift and diffusion functions.

Usage

```
rdtq(h, k = NULL, bigm, a = NULL, b = NULL, init, fT, drift = NULL,
  diffusion = NULL, thresh = 0, method = "sparse")
```

Arguments

`h`	Time step size, a positive numeric scalar.
`k`	Spatial grid spacing, a positive numeric scalar. (Must be specified if `a` and `b` are not specified.)
`bigm`	If `k` is specified, then bigm is a positive integer such that `-bigm*k` and `bigm*k` are, respectively, the minimum and maximum grid points. If `a` and `b` are specified, then `bigm` is the total number of grid points. Note that the fractional part of `bigm` is ignored, and that `floor(bigm)` must be at least 2.
`a`	Left boundary, a numeric scalar. (Must be specified if `k` is not specified.)
`b`	Right boundary, a numeric scalar. (Must be specified if `k` is not specified.)
`init`	A numeric scalar indicating either a fixed initial condition of the form $X(0)$=`init`, or a numeric vector giving the PDF at time $t = 0$. In the latter case, the vector must have the same size as the spatial grid.
`fT`	The final time, a positive numeric scalar. The computation assumes an initial time of t=0 and then computes the PDF at time t=`fT`.
`drift`	When the user chooses the `method="cpp"` algorithm, this should be a pointer to a drift function that is implemented in C++ using `Rcpp`. In our C++ code, we define the type `funcPtr` using the following code: `typedef double (*funcPtr)(const double& x);` We expect the drift function to be a C++ function, implemented using Rcpp, of type `XPtr<funcPtr>`. See the first example below. When the user chooses the `method="sparse"` algorithm, this should be an R function that takes as input a numeric vector of values. The function should return as output a numeric vector containing the drift function evaluated at each element of the input vector. See the second example below.
`diffusion`	When the user chooses the `method="cpp"` algorithm, this should be a pointer to a diffusion function that is implemented in C++ using `Rcpp`. All of the details are analogous to that of the `drift` function described above. When the user chooses the `method="sparse"` algorithm, this should be an R function that takes as input a numeric vector of values. The function should

return as output a numeric vector containing the diffusion function evaluated at each element of the input vector. See the second example below.

`thresh`	This is an optional numeric scalar parameter that is only used for the `method="cpp"` algorithm. When the DTQ summand drops below codethresh, the algorithm stops summing, even if it has not summed over all grid points. The default value of this parameter is zero, indicating that the full DTQ sum is evaluated. Setting this parameter to a small positive value such as 2.2×10^{-16} can result in a substantial speed up for computations on large spatial grids, especially when h is also small.
`method`	A string that indicates which DTQ algorithm to use. There are two choices:

`"cpp"` This DTQ method is implemented in C++. No matrices are formed; the method is highly conservative in its usage of memory. For sufficiently small `h` and `k`, it is necessary to use this method. This method also allows for approximate evaluation of the DTQ algorithm by setting a positive threshold parameter.

`"sparse"` This DTQ method is implemented in R using sparse matrices from the Matrix package. The method uses more memory than the `"cpp"` method, but may be faster for larger values of `h` and `k`. This is the default method.

Details

Consider the stochastic differential equation (SDE)

$$dX(t) = f(X(t))dt + g(X(t))dW(t)$$

where $W(t)$ is standard Brownian motion, f is the drift function, and g is the diffusion function. Let $p(x,t)$ denote the time-dependent probability density function (PDF) of $X(t)$; then `rdtq` computes $p(x,T)$ for a fixed time T.

Note that the PDF is computed on a spatial grid that can be specified in one of two ways:

1. Specify a real, positive value k and a positive integer M = `bigm`. In this case, the PDF will be computed on the grid $x_j = jk$ where $j = -M, -M+1, ..., M-1, M$. In total, there will be $2M+1$ grid points.
2. Specify a real, positive integer M and a computational domain $[a,b]$. In this case, there will be exactly M equispaced grid points. The grid spacing will be $k = (b-a)/(M-1)$.

Value

The output consists of a list with two elements:

`xvec` a numeric vector that contains the spatial grid

`pdf` a numeric vector that contains the PDF evaluated at the grid points

See Also

H. S. Bhat and R. W. M. A. Madushani, "Density Tracking by Quadrature for Stochastic Differential Equations," arXiv:1610.09572 [stat.CO], http://bit.ly/2fbNsp5

Examples

```
# Example 1:
# Define the drift function f(x) = -x and diffusion function g(x) = 1
# using C++ code:
require(Rcpp)
sourceCpp(code = '#include <Rcpp.h>
using namespace Rcpp;
double drift(double& x)
{
  return(-x);
}
double diff(double& x)
{
  return(1.0);
}
typedef double (*funcPtr)(double& x);
// [[Rcpp::export]]
XPtr<funcPtr> driftXPtr()
{
  return(XPtr<funcPtr>(new funcPtr(&drift)));
}
// [[Rcpp::export]]
XPtr<funcPtr> diffXPtr()
{
  return(XPtr<funcPtr>(new funcPtr(&diff)));
}')
# Solve for the PDF (at final time fT=1) of the SDE with drift f,
# diffusion g, and deterministic initial condition X(0) = 0.
# First we solve using the grid specified by k and bigm.
# Then we solve using the grid specified by a, b, and bigm.
# We then check that we get the same PDF either way.
k = 0.01
M = 250
test1 = rdtq(h=0.1,k,bigm=M,init=0,fT=1,
             drift=driftXPtr(),diffusion=diffXPtr(),method="cpp")
test2 = rdtq(h=0.1,a=-2.5,b=2.5,bigm=501,init=0,fT=1,
             drift=driftXPtr(),diffusion=diffXPtr(),method="cpp")
print(k*sum(abs(test1$pdf-test2$pdf)))

# Example 2:
# We again use the drift function f(x) = -x and diffusion function g(x) = 1.
# This time, we use the method="sparse" version of DTQ.
# This requires us to define the drift and diffusion functions in R:
mydrift = function(x) { -x }
mydiff = function(x) { rep(1,length(x)) }
test = rdtq(h=0.1,k=0.01,bigm=250,init=0,fT=1,
            drift=mydrift,diffusion=mydiff,method="sparse")
plot(test$xvec,test$pdf,type='l')
```

---

studydtqconv *Study DTQ Convergence*

Description

`studydtqconv` facilitates the generation of convergence plots, i.e., plots where one studies the error (in various norms) as a function of the time step h; the error is computed as the difference between the exact PDF and the approximate PDF computed via the DTQ method.

Usage

```
studydtqconv(method, drift, diffusion, exact, hseq, kseq, Mseq, init, fT,
  thresh = 0)
```

Arguments

`method`	This must be a string, either "cpp" or "sparse", that indicates which algorithm to use. See the parameter of the same name in the `rdtq` function.
`drift`	if `method="cpp"`, then this should be a pointer to a drift function implemented in C++ using `Rcpp`. If `method="sparse"`, then this should be the name of an R function. For further details, see the description of the `drift` parameter for the `rdtq` function.
`diffusion`	if `method="cpp"`, then this should be a pointer to a diffusion function implemented in C++ using `Rcpp`. If `method="sparse"`, then this should be the name of an R function. For further details, see the description of the `diffusion` parameter for the `rdtq` function.
`exact`	an R function that accepts two arguments, `xvec` and `T`, and returns the exact probability density function $p(x,T)$ for each x in `xvec`.
`hseq`	a numeric vector of values of h, the time step, to use for the computation of the DTQ solution. Note that `hseq` and `kseq` must have the same lengths.
`kseq`	a numeric vector of values of k, the grid spacing, to use for the computation of the DTQ solution. Note that `hseq` and `kseq` must have the same lengths.
`Mseq`	a numeric vector of integer values of M. For each corresponding value of k, the spatial grid will cover the domain $[-y_M, y_M]$ where $y_M = Mk$. This corresponds to the parameter `bigm` in the `rdtq` function. Note that `kseq` and `Mseq` must have the same lengths.
`init`	a scalar initial condition.
`fT`	a positive numeric scalar giving the final time at which to compare the exact and DTQ solutions.
`thresh`	This optional positive scalar is only used when `method="cpp"`. See the parameter of the same name in the `rdtq` function.

Value

The function returns the errors between the DTQ and exact solutions indexed by the corresponding value of hseq. The errors are returned in the L^1 norm, L^∞ norm, and the Kolmogorov-Smirnov norm. The errors are returned in the form of a data frame.

See Also

H. S. Bhat and R. W. M. A. Madushani, "Density Tracking by Quadrature for Stochastic Differential Equations," arXiv:1610.09572 [stat.CO], http://bit.ly/2fbNsp5

Examples

```
# In this example, we will study the convergence of the DTQ method
# for the SDE with drift f(x) = x/2 + (1 + x^2)^(1/2) and
# diffusion g(x) = (1 + x^2)^(1/2).

library(Rdtq)
library(Rcpp)

# implement the drift and diffusion functions using C++
sourceCpp(code = '#include <Rcpp.h>
          using namespace Rcpp;
          double drift(double& x) { return(0.5*x + sqrt(x*x + 1.0)); }
          double diff(double& x) { return(sqrt(x*x + 1.0)); }
          typedef double (*funcPtr)(double& x);
          // [[Rcpp::export]]
          XPtr<funcPtr> driftXPtr() { return(XPtr<funcPtr>(new funcPtr(&drift))); }
          // [[Rcpp::export]]
          XPtr<funcPtr> diffXPtr() { return(XPtr<funcPtr>(new funcPtr(&diff))); }')

# implement the drift and diffusion functions using R
mydrift = function(y)
{
  return(0.5*y + sqrt(y^2 + 1))
}
mydiff = function(y)
{
  return(sqrt(y^2 + 1))
}

# implement the exact solution at time t, i.e.,
# the analytical formula for the pdf p(x,t)
exactsol = function(xvec,t)
{
  transx = asinh(xvec) - t
  prefac = (1 + xvec^2)^(-1/2)
  z = prefac*dnorm(x=transx)
  return(z)
}

# define the sequence of parameters that will be used to study convergence
hseq = c(0.5,0.2,0.1,0.05)
kseq = hseq^(0.55)
Mseq = ceiling(5*(-log(hseq))/kseq)

# we will use the method="sparse" code for the three largest values in hseq,
# and then switch to the method="cpp" code for the three smallest values
firstpart = c(1:2)
```

```
errpart1 = studydtqconv(method="sparse",drift=mydrift,diffusion=mydiff,exact=exactsol,
                        hseq[firstpart],kseq[firstpart],Mseq[firstpart],
                        init=0,fT=1)
errpart2 = studydtqconv(method="cpp",drift=driftXPtr(),diffusion=diffXPtr(),exact=exactsol,
                        hseq[-firstpart],kseq[-firstpart],Mseq[-firstpart],
                        init=0,fT=1,thresh=1e-16)

# now we will put everything together into one data frame
mydat = rbind(errpart1,errpart2)

# we plot the convergence diagram, on a log-log scale, using ggplot2
library(ggplot2)
library(scales)
myplot = ggplot(data=mydat, aes(x=x,y=y,group=norm,color=norm))
myplot = myplot + theme_bw() + theme(plot.background = element_rect(fill='white'))
myxticks = sort(10^(round(log(hseq)/log(10)*10)/10))
rawyticks = round(log(mydat$y)/log(10)*10)/10
rawyticks = round(seq(from=min(rawyticks),to=max(rawyticks),length.out=length(myxticks))*1)/1
myyticks = unique(10^rawyticks)
myplot = myplot + scale_x_log10(breaks = hseq)
myplot = myplot + theme(axis.text.x = element_text(angle=90,hjust=1))
myplot = myplot + scale_y_log10(breaks = myyticks,
                                labels = trans_format("log10", math_format(10^.x)))
myplot = myplot + labs(x="h (temporal step size)", y="error")
myplot = myplot + geom_line() + geom_point()

# save the plot to a pdf (portable document format) file
ggsave(filename="example.pdf", plot=myplot, width=5, height=4)
```

Index