README

Contents

SC2GroupProject

The goal of SC2GroupProject is to Perform SMC sampling in order to approximate expectations of interest for diffusions processes.

Installation

You can install the development version of SC2GroupProject from GitHub with:

```r
# install.packages("devtools")
devtools::install_github("joshgivens/SC2GroupProject")
```

Main Problem

Here is the R code required to compute the statistics of interest for our problem

```r
library(SC2GroupProject)

#Set our final timepoints
Final_T=1
#Set number of timesteps we will use
timesteps=20
#Get the size of each timestep
step=Final_T/timesteps

#Create sampling function
sampler <- function(x_0){
  sin_samp(1,x_0,step,pi)
}

# Create Likelihood Function
true_lik <- function(x,x_0){
  sin_lik(x,x_0,step,pi)
}

# Create g function
g <- function(x){
  zeta <- 100
  omega2 <- 1e-30
  diff=sqrt(zeta)-omega2
  if(x>-diff & x<diff){
    return(
      exp(1/(zeta-x^2))
    )
  }
  else{
    return(0)
  }
}
```

Now we have set up we can use this to approximate our expectation of interest. We first do this with 1,000 samples starting at $x_0 = 1$.

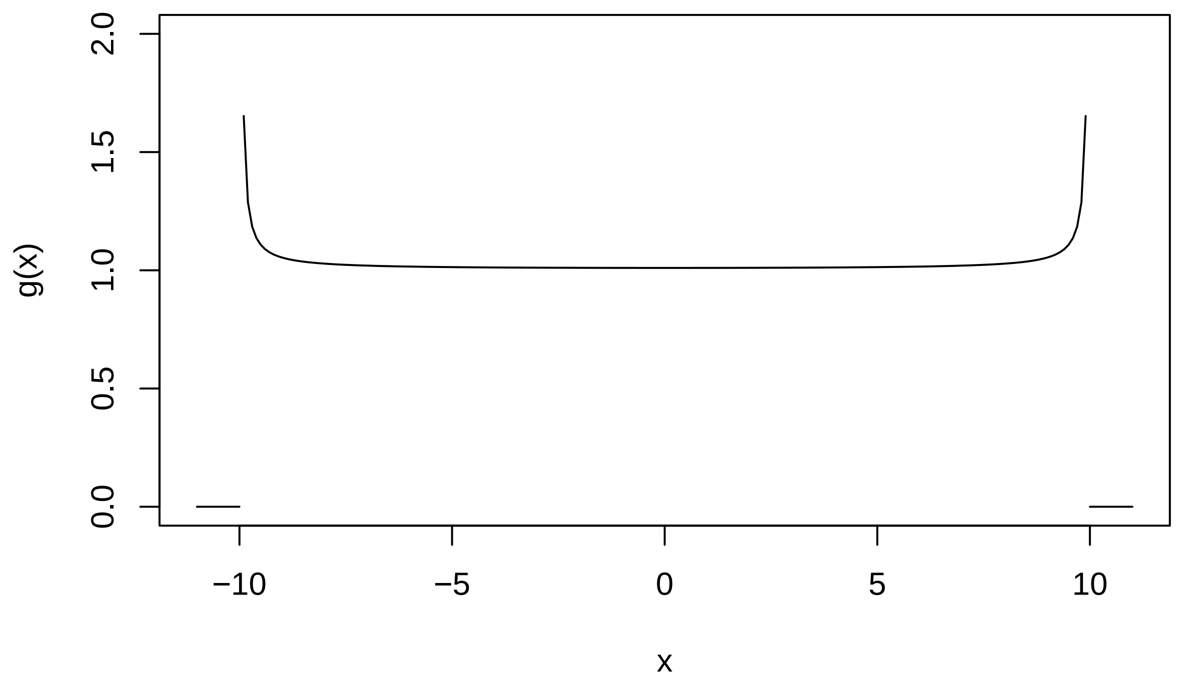
```
#Simulate 1,000 particles
out <- diff_SMC(g=g,p_delta=true_lik,M_T_samplers = rep(list(sampler),timesteps),
         M_T_lik = rep(list(true_lik),timesteps), n = 1000, timesteps = timesteps,
         varphi = 0.9,x_0=1,resample = T)
```

Now we have performed the sampling we can calculate the normalising constant of this distribution using the `harmonic_norm` and `basic_norm` functions.

```
log(harmonic_norm(out$gmat[,timesteps],out$Wmat,0.9))
#> [1] 0.01001797
log(basic_norm(out$Wmat))
#> [1] 0.009016182
```

Both these give approximations of $\log \mathbb{E}(g(X_T))$ with $T = 1$. For illustration, below is the plot of the function g on $(-11, 11)$

```
x <- seq(from=-10+1e-15,to=10-1e-15,by=0.1)
y=rep(NA,length(x))
for(i in 1:length(x)){
  y[i] <- g(x[i])
}
plot(x,y,type="l",ylim=c(0,2),xlim=c(-11,11),xlab="x",ylab="g(x)",main="")
lines(c(10,11),y=c(0,0))
lines(-c(10,11),y=c(0,0))
```



C++ implementation

As well as an R implementation of this sampler we have included a C++ implementation which is significantly quicker. This implementation is specialised to this exact case and can be found in the function `diff_SMC_sin`.

First we demonstrate that it does indeed estimate the same procedure

```
out_cpp <- diff_SMC_sin(x_0=1,t=step,n=10000,timesteps=timesteps,varphi=0.9,TRUE)
out <- diff_SMC(g=g,p_delta=true_lik,M_T_samplers = rep(list(sampler),timesteps),
         M_T_lik = rep(list(true_lik),timesteps), n = 10000, timesteps = timesteps,
         varphi = 0.9,x_0=1,resample = T)


log(harmonic_norm(out$gmat[,timesteps],out$Wmat,0.9))
#> [1] 0.01001828
log(harmonic_norm(out_cpp$gmat[,timesteps],out$Wmat,0.9))
#> [1] 0.01001795
```

We now show that this process gives significant speed increases

```
library(microbenchmark)
options(microbenchmark.unit="relative")
microbenchmark(C_version = diff_SMC_sin(x_0=1,t=step,n=10000,timesteps=timesteps,varphi=0.9,TRUE),
               R_version = diff_SMC(
                 g=g,p_delta=true_lik,M_T_samplers = rep(list(sampler),timesteps),
                 M_T_lik = rep(list(true_lik),timesteps), n = 10000, timesteps = timesteps,
                 varphi = 0.9,x_0=1,resample = T),
               times = 10)
```

```
#> Unit: relative
#>       expr     min       lq     mean   median       uq      max neval
#>  C_version  1.0000  1.00000  1.00000  1.00000  1.00000  1.00000    10
#>  R_version 28.0048 28.28041 28.67441 28.55913 28.67643 29.62002    10
```

Approximations for various x_0

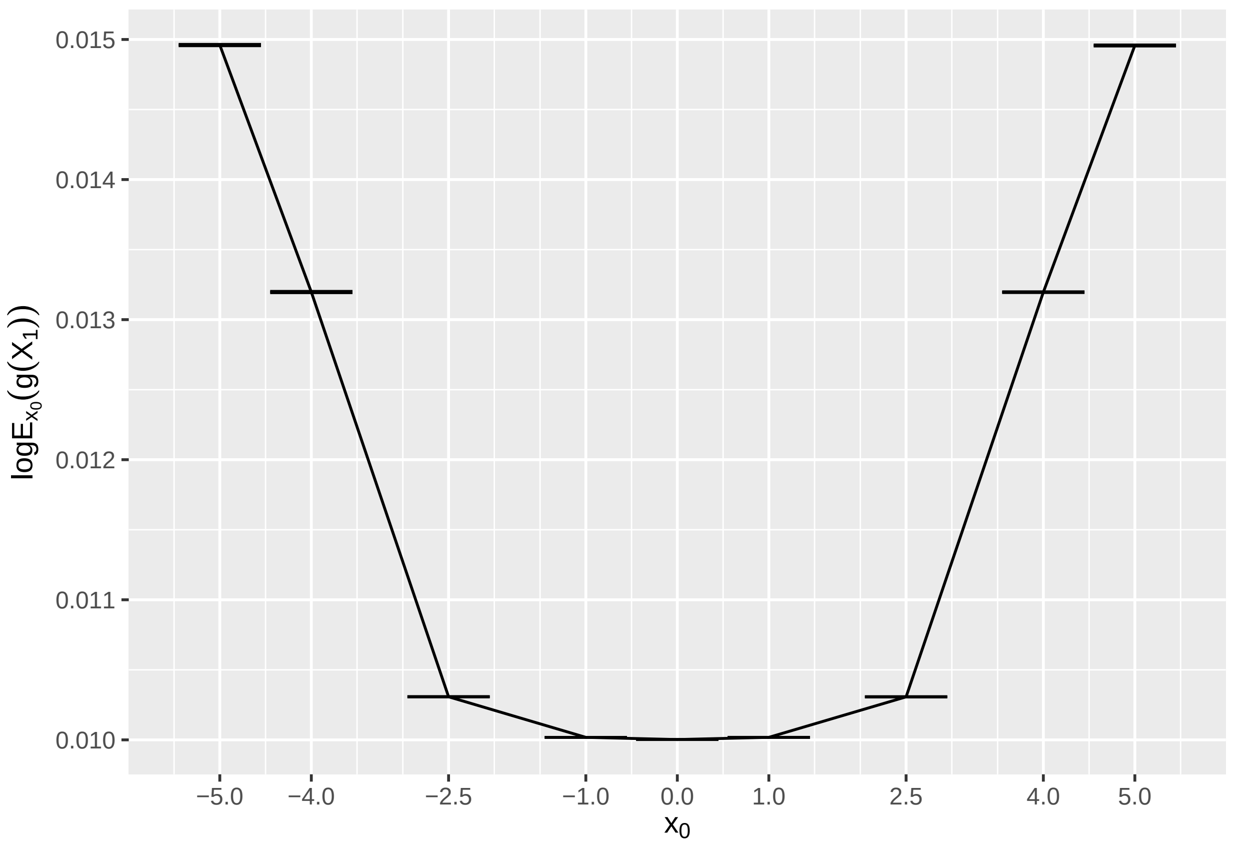
We see that the C++ versions is almost 30 times faster which is impressive. We now use this to perform approximations for various values of x_0,

```
library(dplyr)
#>
#> Attaching package: 'dplyr'
#> The following objects are masked from 'package:stats':
#>
#>     filter, lag
#> The following objects are masked from 'package:base':
#>
#>     intersect, setdiff, setequal, union
x_0s <- c(-5,-4,-2.5,-1,0,1,2.5,4,5)
set.seed(1234)
Preds <- data.frame(x_0=numeric(),sample=numeric(),estimate=numeric())
for (i in 1:length(x_0s)){
  for (j in 1:10){
    out <- diff_SMC_sin(x_0=x_0s[i],t=step,n=10000,timesteps=timesteps,varphi=0.9,TRUE)
    pred <- log(harmonic_norm(out$gmat[,timesteps],out$Wmat,0.9))
    Preds <- Preds %>% add_row(x_0=x_0s[i],sample=j,estimate=pred)
  }
}
```

We now plot the estimations alongside there standard deviations.

```
library(latex2exp)
library(ggplot2)
Preds_sum <- Preds %>% group_by(x_0) %>%
  summarise(mean=mean(estimate),
            sd=sd(estimate),
            upper=mean+sd,lower=mean-sd)

ggplot(Preds_sum,aes(x=x_0,y=mean,ymin=lower,ymax=upper))+geom_line()+
  geom_errorbar()+labs(x=TeX("$x_0$"),y=TeX("$\\log E_{x_0}(g(X_1))$"))+
  scale_x_continuous(breaks=x_0s,minor_breaks = seq(from=-6,to=6,by=0.5))
```



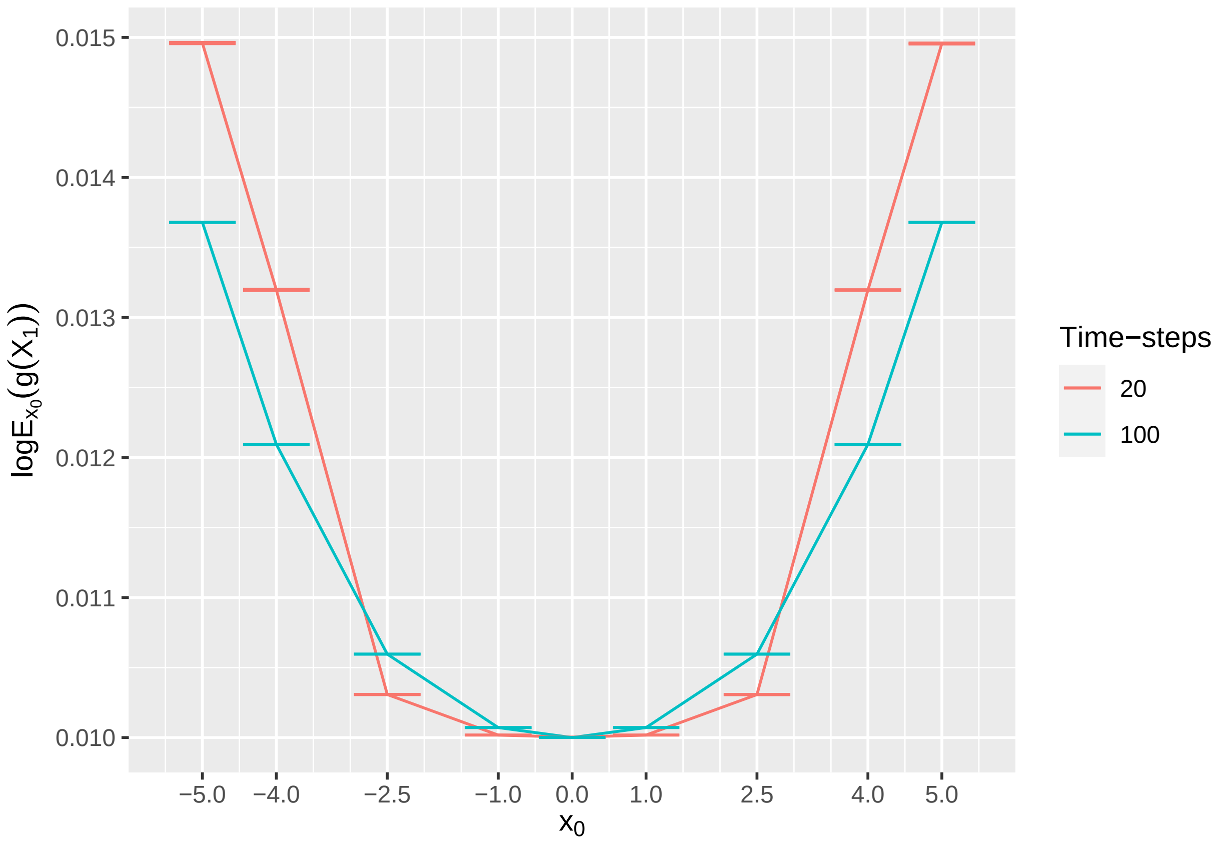
We can additionally try more time-points and see if that varies our prediction. We now try with 100 time-steps giving $\Delta t = 0.01$

```
Preds2 <- Preds %>% mutate(timesteps=20)
for (i in 1:length(x_0s)){
  for (j in 1:10){
    out <- diff_SMC_sin(x_0=x_0s[i],t=1/100,n=10000,timesteps=100,varphi=0.9,TRUE)
    pred <- log(harmonic_norm(out$gmat[,timesteps],out$Wmat,0.9))
    Preds2 <- Preds2 %>% add_row(x_0=x_0s[i],sample=j,estimate=pred,timesteps=100)
  }
}
```

No plot the 2 against one another.

```
Preds_sum2 <- Preds2 %>% group_by(timesteps,x_0) %>%
  summarise(mean=mean(estimate),
            sd=sd(estimate),
            upper=mean+sd,lower=mean-sd)
#> `summarise()` has grouped output by 'timesteps'. You can override using the
#> `.groups` argument.

ggplot(Preds_sum2,aes(x=x_0,y=mean,ymin=lower,ymax=upper,colour=factor(timesteps)))+geom_line()+
  geom_errorbar()+labs(x=TeX("$x_0$"),y=TeX("$\\log E_{x_0}(g(X_1))$"),colour="Time-steps")+
  scale_x_continuous(breaks=x_0s,minor_breaks = seq(from=-6,to=6,by=0.5))
```



Additional Content

True Samplers

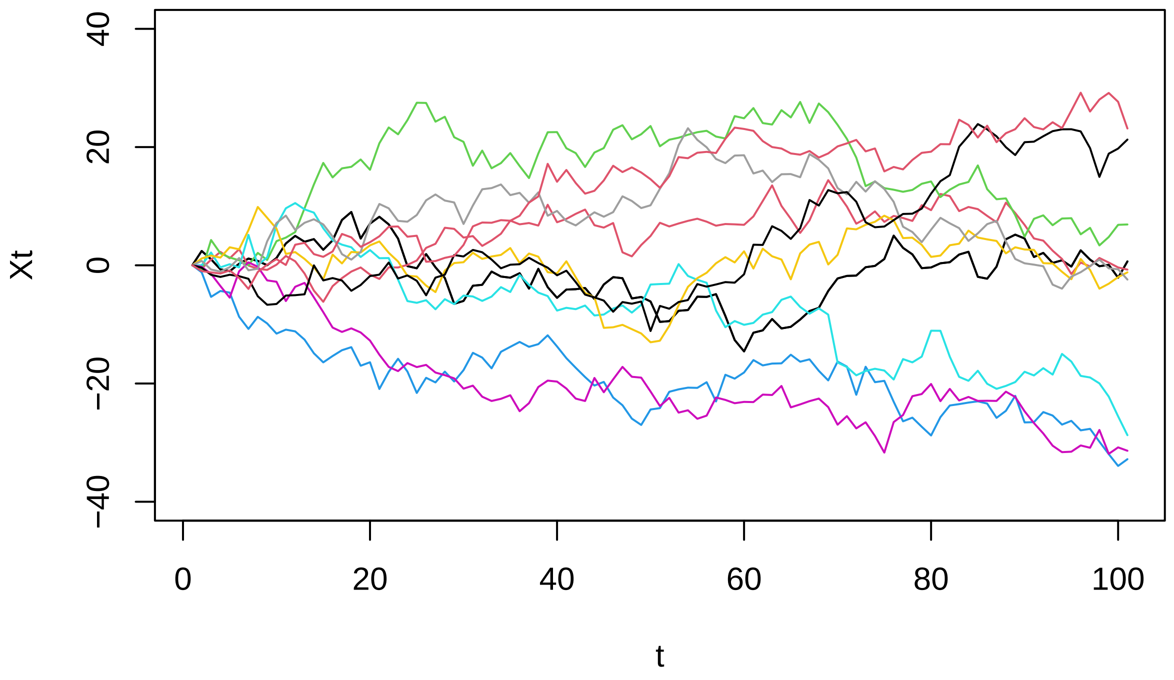
As well as the implementations shown above we have various other functions which can help with SMC. First of all we have implemented a method to do direct discrete sampling as well the the exact transition kernels for both Brownian Motion and Ornstein-Uhlenbeck Process.

Here is an example of various simulated particles from Brownian Motion.

```
true_sampler <- function(x_0){
  brown_samp(1,x_0,t=1,sigma=2)
}

out_true <- discrete_sampler(true_sampler,x_0=0.01,timesteps = 100,n=10)

plot(out_true[1,],type="l", ylim=c(-40,40),xlab="t",ylab="Xt")
for (i in 1:10){
  lines(out_true[i,],col=i)
}
```



Standard SMC

We have also implemented a standard SMC sampler for use with any diffusions process

Below is a demonstration of using a Brownian Motion to generate samples from and Ornstein-Uhlenbeck process with $\alpha = 1$.

```
Final_T=1
timesteps=100
step=Final_T/timesteps

init_sample <- function(n){
  brown_samp(n,1,step,sigma=1)
}

sampler <- function(x_0){
  brown_samp(1,x_0,step,sigma=1)
}

true_sampler <- function(x_0){
  ornuhl_samp(1,x_0,step,alpha=1,sigma=1)
}

prop_lik <- function(x,x_0){
  brown_lik(x,x_0,step,sigma=1)
}
```

```
true_lik <- function(x,x_0){
  ornuhl_lik(x,x_0,step,alpha = 1,sigma=1)
}
```

Now we generate the process.

```
# Generate y using exponential with shape parameter x^2
x_0 <- 1
hidden_x <- rep(NA,timesteps)
hidden_x[1] <- true_sampler(x_0)
# Generate hidden x sample
for(i in 2:100){
  hidden_x[i] <- true_sampler(hidden_x[i-1])
}

y=rexp(timesteps,rate=hidden_x^2)

g_lik <- function(y,x){
  dexp(y,rate=x^2)
}
```

```
out <- Basic_SMC(init_sample=init_sample,
                 proposal_sample=sampler,
                 proposal_lik = prop_lik,
                 true_lik=true_lik,
                 g_lik=g_lik,
                 y=y,
                 n=1000,
                 timesteps=timesteps)
```

Non-reweighted approach

There is also an option in both `diff_SMC` and `diff_SMC_sin` to not do re-sampling and simply compound the weights. This is generally note recommended as it can lead to degeneracy in the weights and other issues.

We can see this here as not using re-sampling leads to higher variance in the end weights?

```
#Set our final timepoints
Final_T=1
#Set number of timesteps we will use
timesteps=100
#Get the size of each timestep
step=Final_T/timesteps

#Create sampling function
sampler <- function(x_0){
  sin_samp(1,x_0,step,pi)
}

# Create Likelihood Function
true_lik <- function(x,x_0){
  sin_lik(x,x_0,step,pi)
}

g <- function(x){x^2}
```

```r
#Simulate 1,000 particles
out_resamp <- diff_SMC(g=g,p_delta=true_lik,M_T_samplers = rep(list(sampler),timesteps),
         M_T_lik = rep(list(true_lik),timesteps), n = 100, timesteps = timesteps,
         varphi = 0.9,x_0=1,resample = TRUE)
#Simulate 1,000 particles
out_nonresamp <- diff_SMC(g=g,p_delta=true_lik,M_T_samplers = rep(list(sampler),timesteps),
         M_T_lik = rep(list(true_lik),timesteps), n = 100, timesteps = timesteps,
         varphi = 0.9,x_0=1,resample = FALSE)

# Now normalise the weights to add up to 1
norm_W_resamp <- out_resamp$Wmat[,timesteps]/sum(out_resamp$Wmat[,timesteps])
norm_W_nonresamp <-  out_nonresamp$Wmat[,timesteps]/sum(out_nonresamp$Wmat[,timesteps])

#Compare standard deviations
sd(norm_W_resamp)
#> [1] 0.001952011
sd(norm_W_nonresamp)
#> [1] 0.003335453
```

As we can see the non-resampled weightings give double the standard deviation of the resampled weightings.

Timings

What our algorithm has over the original paper is speed. We demonstrate this now be exactly mimicking their set-up. This is 5 starting points, 10 times each with 1,000 particles and 20 time-points. Overall this took them 76.5 seconds.

```r
x_0s <- c(-5,-2.5,0,2.5,5)
preds <- rep(NA,10*length(x_0s))
index <- 1
start_time <- Sys.time()
for (i in 1:length(x_0s)){
  for (j in 1:10){
    out <- diff_SMC_sin(x_0=x_0s[i],t=1/20,n=1000,timesteps=20,varphi=0.9,TRUE)
    preds[i] <- log(harmonic_norm(out$gmat[,20],out$Wmat,0.9))
    index=index+1
  }
}
end_time <- Sys.time()

print(end_time-start_time)
#> Time difference of 0.3969629 secs
```

That's over 150 time speed up.

As the time is so short we do 100 iterations at each point and then divide the time by 10.

```r
preds <- rep(NA,100*length(x_0s))
index <- 1
start_time <- Sys.time()
for (i in 1:length(x_0s)){
  for (j in 1:100){
    out <- diff_SMC_sin(x_0=x_0s[i],t=1/20,n=1000,timesteps=20,varphi=0.9,TRUE)
    preds[i] <- log(harmonic_norm(out$gmat[,20],out$Wmat,0.9))
    index=index+1
  }
}
end_time <- Sys.time()

print((end_time-start_time)/10)
#> Time difference of 0.3847555 secs
```

Even shorter!