Stochastic Differential Equation and discretization

Sang

11/20/2019

Contents

The main goal of this document is to validate the discretization of linear Stochastic differential Equations (SDEs). Here, I got the equations from Wikipedia link. In 3, 4, I will validate the equations by comparing with 3. discretized solution in a book and 4. numerical solutions from SDE solver.

1. Simple SDE example

See the following stochastic differential equation (Sarkka p. 80, 2019).

$$d\mathbf{x} = \mathbf{A}\mathbf{x}dt + \boldsymbol{\sigma}d\boldsymbol{\omega}$$

where $\boldsymbol{\sigma}d\omega$ follows Brownian motion with a diagonal diffusion matrix ($\boldsymbol{Q} = \text{diag}(\boldsymbol{\sigma}^2)$) (i.e., ω is standard weiner process). $\mathbf{x}$ is state vector, $\mathbf{A}$ is state transition matrix.

In a discrete form (Sarkka, 2019),

$$\mathbf{x}[t_k + 1] = \mathbf{A}_d\mathbf{x}[t_k] + \boldsymbol{\varepsilon}_d$$

Here subscript $_d$ indicates discretized parameter. (i.e., disctretized $\mathbf{A}$ is $\mathbf{A}_d$)

Where

$$\begin{aligned}
&\boldsymbol{\varepsilon}_d \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_d), \text{ and } \mathbf{Q}_d \text{ is discretized from } \mathbf{Q} = \text{diag}(\boldsymbol{\sigma}^2) \\
&\mathbf{A}_d = \exp(\mathbf{A}\Delta t) \\
&\mathbf{Q}_d = \int_0^{\Delta t} \exp(\mathbf{A}(\Delta t - \tau))\,\mathbf{Q}\exp(\mathbf{A}(\Delta t - \tau))^\intercal d\tau
\end{aligned}$$

2. Discretization of system of SDEs (linear)

Matrix exponential can be used to discretize $\mathbf{A}$, but the process (state) noise ($\boldsymbol{\sigma}_d$ and $\mathbf{Q}_d$) are hard to compute due to $\mathbf{Q}_d = \int_0^{\Delta t} \exp(\mathbf{A}(\Delta t - \tau))\,\mathbf{Q}\exp(\mathbf{A}(\Delta t - \tau))^\intercal d\tau$. Therefore, I refer below two references.

According to Wikipedia (link or this reference(Loan, 1978)), the discretized process noise can be calculated from below relationship.

$$\begin{aligned}
\exp\left(\begin{bmatrix} -\mathbf{A} & \mathbf{Q} \\ \mathbf{0} & \mathbf{A}^\intercal \end{bmatrix}\Delta T\right) &= \begin{bmatrix} \dots & \mathbf{A}_d^{-1}\mathbf{Q}_d \\ \mathbf{0} & \mathbf{A}_d^\intercal \end{bmatrix} \\
\mathbf{Q}_d &= \mathbf{A}_d(\mathbf{A}_d^{-1}\mathbf{Q}_d)
\end{aligned}$$

Specifically, first we need to calculate $\exp\left(\begin{bmatrix} -\mathbf{A} & \mathbf{Q} \\ \mathbf{0} & \mathbf{A}^{\mathrm{T}} \end{bmatrix}\Delta T\right)$ and then get $\mathbf{Q}_d$ by multiplying right-hand side elements of $\begin{bmatrix} \dots & \mathbf{A}_d^{-1}\mathbf{Q}_d \\ \mathbf{0} & \mathbf{A}_d^{\mathrm{T}} \end{bmatrix}$ from bottom element.

3. Validate discretization by comparing with a given solution

In an example 3 in (Sarkka p. 82, 2019), a solution of SDE discretization is given. Let's compare the solution with the method given the above approach.

3.1 SDEs

The system is as follows:

$$\begin{pmatrix} dx_1 \\ dx_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} dt + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \sigma d\omega$$

where $\sigma d\omega$ is a standard weiner process with diffusion coefficient $q = \sigma^2$. When it is written in a discrete form, the solution given the book is

3.2 Solution

Here is the solution given in the book (Sarkka p. 82, 2019).

$$\begin{aligned} \mathbf{A}_d &= \exp(\mathbf{A}\Delta t) \\ \mathbf{Q}_d &= \begin{pmatrix} \frac{1}{3}\Delta t^3 & \frac{1}{2}\Delta t^2 \\ \frac{1}{2}\Delta t^2 & \Delta t \end{pmatrix} q \end{aligned}$$

3.3 Discretization and comparison

Let's have some numerical example. Here, my interest is $\mathbf{Q}_d$ because we already knows that $\mathbf{A}_d$ can be easily calcualted by matrix exponential.

```
q=0.5 #diffusion matrix element (variance)
A=matrix(c(0,0,1,0),ncol=2) #A matrix
Q=matrix(c(0,0,0,q),ncol=2) #diffusion Q matrix.
dt=2 #discrete time sampling

# Ad
Ad=Matrix::expm(A*dt) #matrix exponential
# Temporal calculation
Qd_temp=Matrix::expm(rbind(cbind(-A,Q),cbind(Q*0,t(A)))*dt)
Qd=as.matrix(Ad%*%Qd_temp[1:2,3:4])
# Qd_true is obtained from the equation in the book
Qd_true=as.matrix(matrix(c(1/3*dt^3,1/2*dt^2,1/2*dt^2,dt),ncol=2)*q)
```

Print two solutions and check if they are same.

```
print(Qd) # from calculation.
```

```
##          [,1] [,2]
## [1,] 1.333333    1
## [2,] 1.000000    1
```

```
print(Qd_true) #given in the book.
```

```
##          [,1] [,2]
## [1,] 1.333333    1
## [2,] 1.000000    1
```

```
# Correct.
```

So, yes, the discretization equation is valid.

4. SDE discretization comparison with SDE solvers

4.1 Probelm description

Now, let's have this discretization method in a real stochastic differential examples, and then see how much the discretization make sense.

I will use a system of SDE example. I will solve this problem by using SDE solver library in Juliar language. And then, the solution is compared with discretization scheme given above.

We have following SDE system.

$$\dot{x}_1 = -0.5x_1 - 2x_2 + 0.5\dot{\omega}_1$$
$$\dot{x}_2 = 2x_1 - x_2 + 0.3\dot{\omega}_1$$

In a matrix format,

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \boldsymbol{\sigma}\dot{\omega}$$

where $\mathbf{A} = \begin{bmatrix} -0.5 & -2 \\ 2 & -1 \end{bmatrix}$ and $\boldsymbol{\sigma}\dot{\omega}$ follows a brownian motion with diffusion matrix $\mathbf{Q} = \begin{bmatrix} 0.5^2 & 0 \\ 0 & 0.3^2 \end{bmatrix}$.

4.2 Solve SDE by using Juliar solver

Solving this system of differential equation by using Julia's Differential equation library link. Here, u is used instead of x due to the notation given in the library. f is state transition process (i.e., $\mathbf{A}$), and g is diffusion process ($\boldsymbol{\sigma}\dot{\omega}$). The notation is different, but the equations are same.

```
f<-JuliaCall::julia_eval("
                         function f(du,u,p,t)
                         du[1]=p[1,1]*u[1]+p[1,2]*u[2]
                         du[2]=p[2,1]*u[1]+p[2,2]*u[2]
                         end")

g<-JuliaCall::julia_eval("
                         function g(du,u,p,t)
                         du[1]=0.5
                         du[2]=0.3
                         end")

A=matrix(c(-0.5,2,-2,-1),ncol=2) # parameter matrix p
#noise parameter. I put 0.5, 0.3 in the g function explicitly.
Q=matrix(c(0.5^2,0,0,0.3^2),ncol=2)

NT=12
u0 = c(3,3) #initial point
tspan <- list(0.0,NT) #time from 0 to 12
sol_list=list() # store montecarlo simulation results
niter=2000 # number of stochastic simulation. #
for (i in 1:niter){
  set.seed(i)
  sol_list[[i]]=diffeqr::sde.solve('f','g',u0,tspan,p=A,saveat=0.005)
  # if(i%%50==0) print(i) # print progress
}

# extract results
x1s=bind_cols(lapply(sol_list,function(x) x$u[,1]))
```

```
tgrid=sol_list[[1]]$t
x2s=bind_cols(lapply(sol_list,function(x) x$u[,2]))

set.seed(1234)
sample_idx<-sample(niter,5,replace=FALSE)
plot_sample<-list()
for (i in 1:length(sample_idx)){
  plot_sample[[i]]=
    tibble(t=tgrid,x1=x1s[,sample_idx[i]][[1]],x2=x2s[,sample_idx[i]][[1]],id=i)
}
plot_sample_all=bind_rows(plot_sample)%>%
  group_by(id)%>%arrange(t)%>%
  ungroup()%>%mutate(id=as.character(id))
```

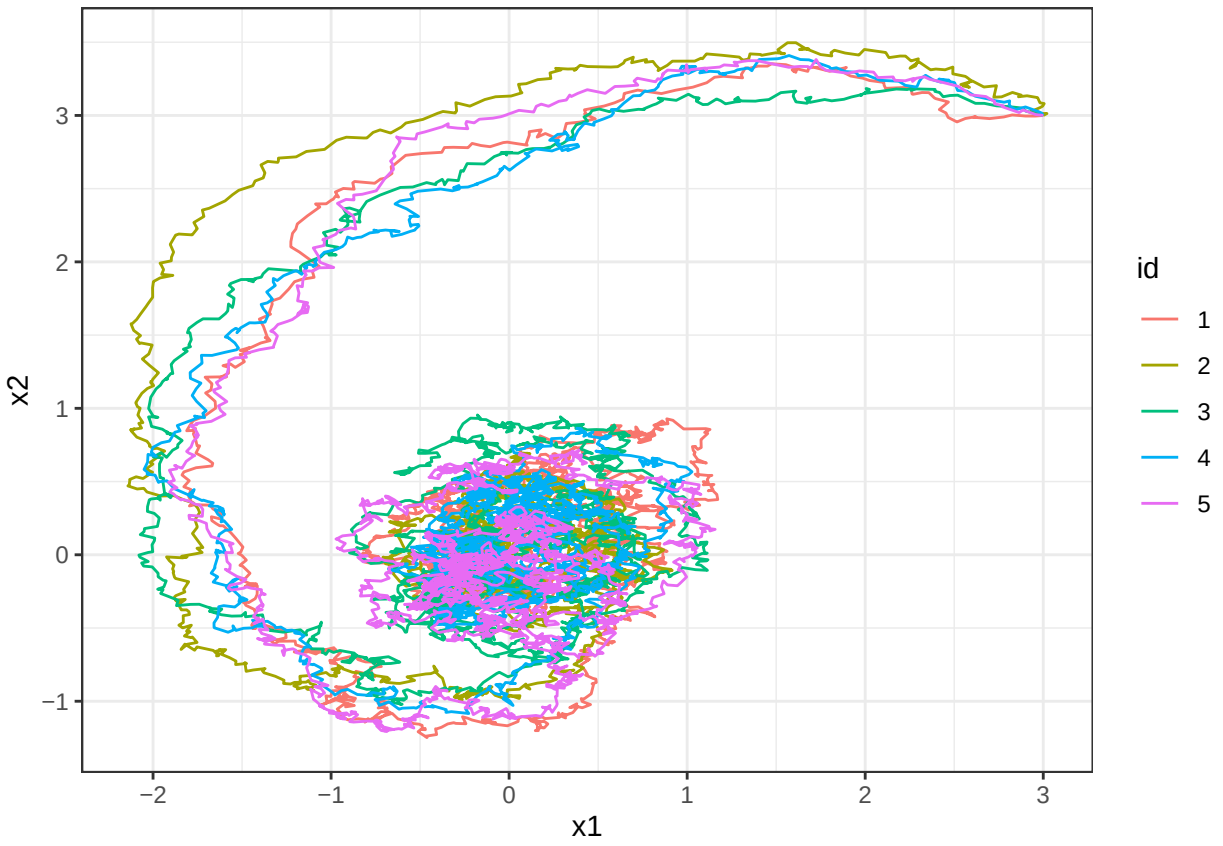
4.3 Visualize SDE solver results

Let's visualize 5 SDE solutions.

```
cat("Visualize 10 sampled solutions")
```

```
## Visualize 10 sampled solutions
```

```
ggplot(plot_sample_all,aes(x=x1,y=x2))+
  geom_path(aes(group=id,color=id))+theme_bw()
```

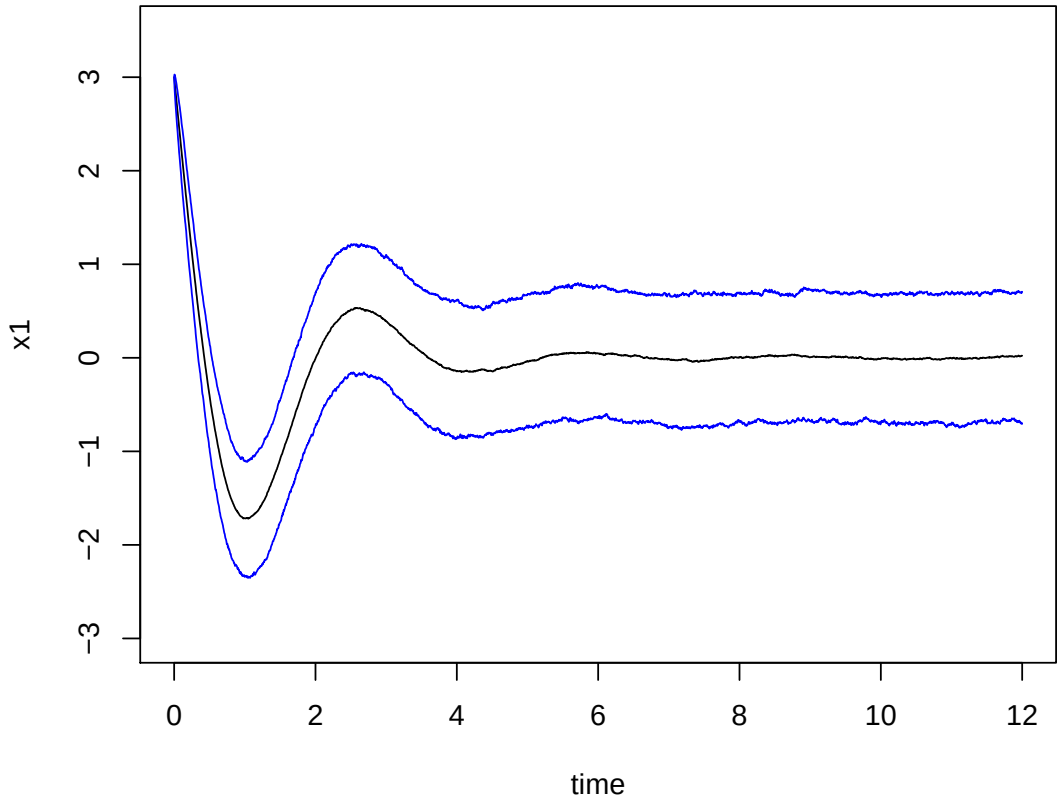


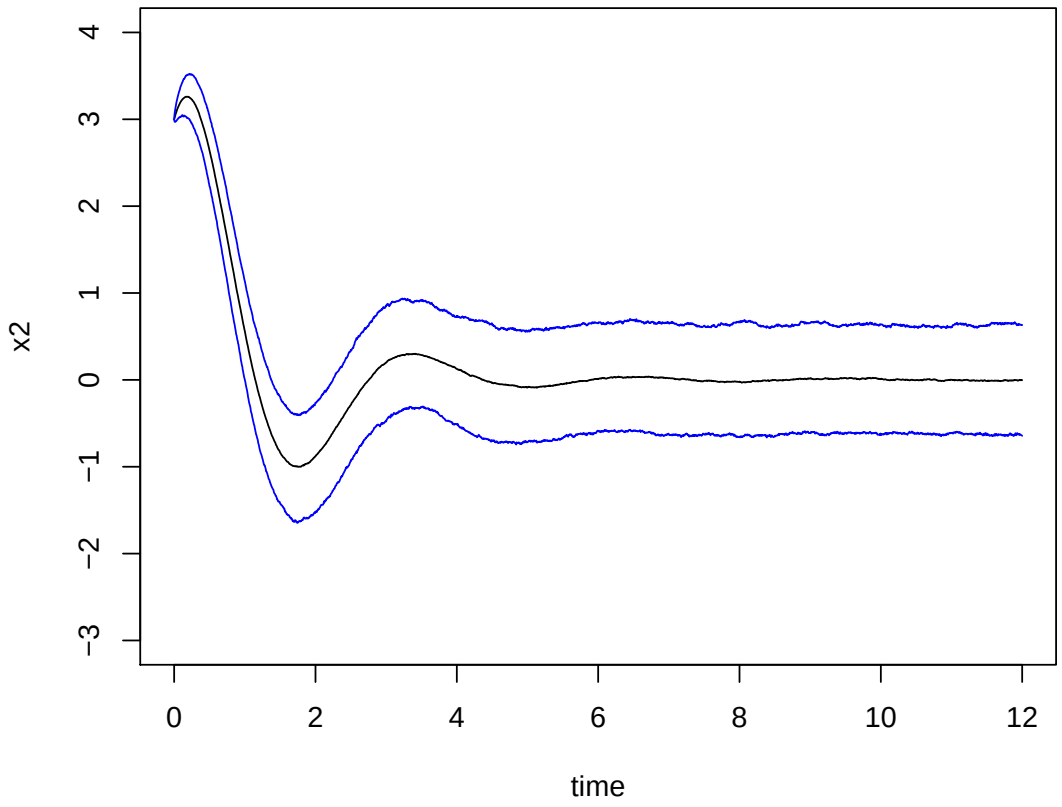
Let's visualize all montecarlo simulations (2000 times). I will visualize each variable with $0.025, 0.5, 0.975$ quentiles.

```
x1q=x1s%>%apply(MARGIN=c(1),function(x) quantile(x,probs=c(0.025,0.5,0.975)))
x2q=x2s%>%apply(MARGIN=c(1),function(x) quantile(x,probs=c(0.025,0.5,0.975)))
par(mfrow=c(2,1),mar = c(4, 4, 2, 2))
cat("Visualize 2.5%~97.5% quantiles for each variable with respect to time")
```

```
## Visualize 2.5%~97.5% quantiles for each variable with respect to time
```

```
plot(tgrid,x1q[2,],type='l',ylim=c(-3,3.5),ylab="x1",xlab="time")
points(tgrid,x1q[1,],type='l',col="blue")
points(tgrid,x1q[3,],type='l',col="blue")

plot(tgrid,x2q[2,],type='l',ylim=c(-3,4),ylab="x2",xlab="time")
points(tgrid,x2q[1,],type='l',col="blue")
points(tgrid,x2q[3,],type='l',col="blue")
```


x1
3
2
1
0
−1
−2
−3
0
2
4
6
8
10
12
time


x2
4
3
2
1
0
−1
−2
−3
0
2
4
6
8
10
12
time

Let's make a comparison of SDE solver and discretized solutions. Here are parameters, and we will find $\mathbf{A}_d$ and $\mathbf{Q}_d$ from above equations.

$$\mathbf{A} = \begin{bmatrix} -0.5 & -2 \\ 2 & -1 \end{bmatrix} \text{ and } \mathbf{Q} = \begin{bmatrix} 0.5^2 & 0 \\ 0 & 0.3^2 \end{bmatrix}$$

Discrete SDE is done by

$$\mathbf{x}[t_k + 1] = \mathbf{A}_d \mathbf{x}[t_k] + \boldsymbol{\varepsilon}_d \text{ and } \boldsymbol{\varepsilon}_d \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_d)$$

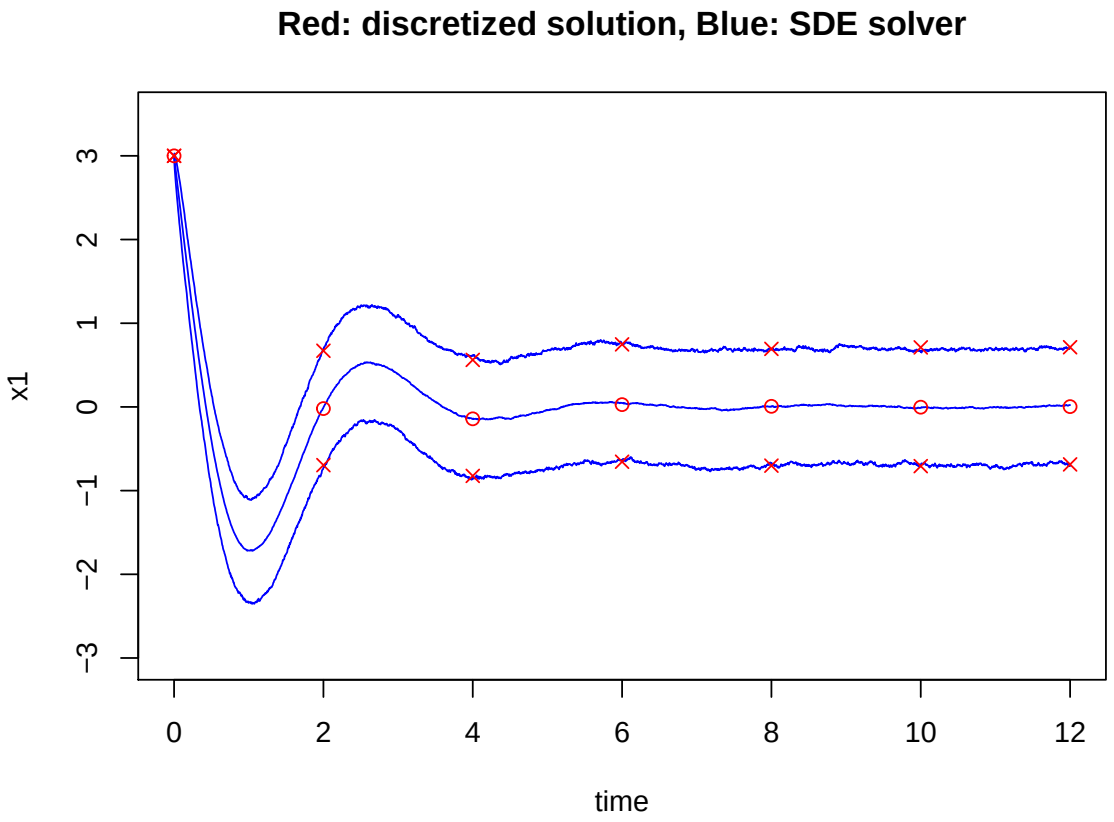
4.4 Discretized solution

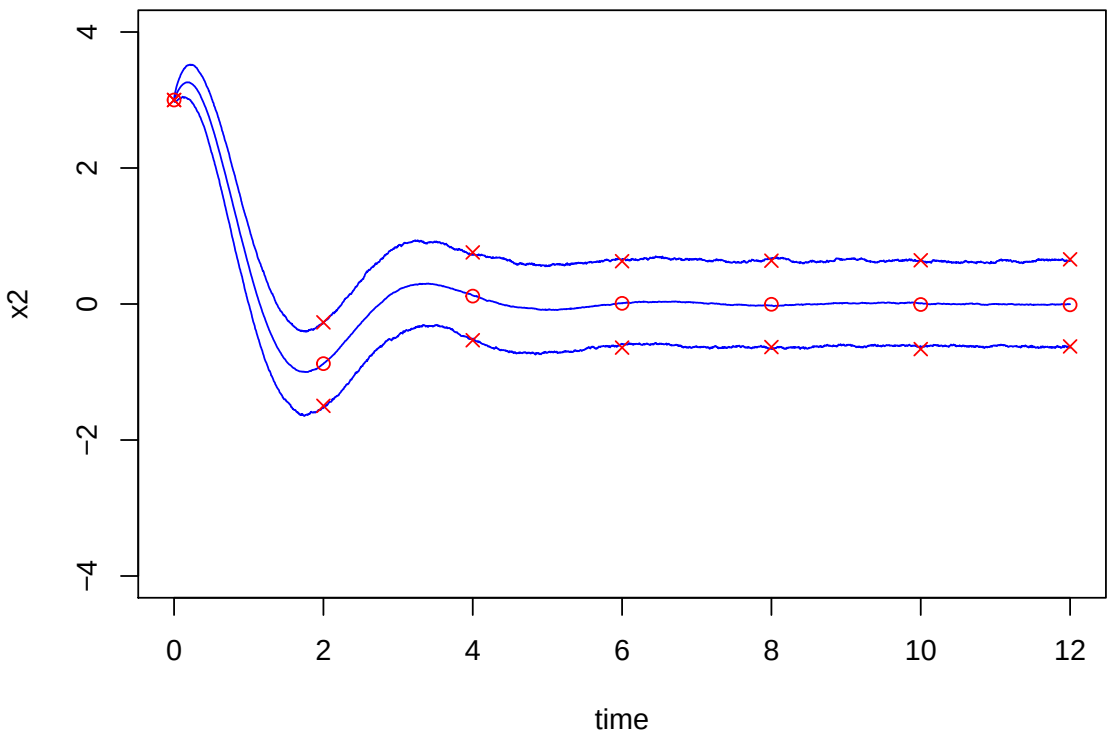
```
# A matrix
A=matrix(c(-0.5,2,-2,-1),ncol=2)
# Diffusion matrix of process (state) noise
Q=matrix(c(0.5^2,0,0,0.3^2),ncol=2)
NT=12
# discretized time interval
dt=2.

nx=dim(A)[1]

# discretized A matrix
Ad=(Matrix::expm(A*dt))%>%as.matrix()

# distretized Q matrix
Qd_temp=Matrix::expm(rbind(cbind(-A,Q),cbind(Q*0,t(A)))*dt) #temporary
Qd=(Ad%*%Qd_temp[1:nx,((nx+1):(nx*2))])%>%as.matrix()

# initial starting point
x0=c(3,3)
# number of particles to do montecarlo simulation
NP=5000
# store data
xs=array(0,c(length(x0),NP,(NT/dt)+1))
# initial point
xs[,,1]=matrix(rep(x0,NP),ncol=NP)

for(i in 2:dim(xs)[3]){
  xs[,,i]=Ad%*%xs[,,i-1]+t(mvtnorm::rmvnorm(NP,mean=c(0,0),sigma=Qd))
}


x1dq=xs[1,,]%>%apply(MARGIN=c(2),function(x) quantile(x,probs=c(0.025,0.5,0.975)))
x2dq=xs[2,,]%>%apply(MARGIN=c(2),function(x) quantile(x,probs=c(0.025,0.5,0.975)))
td_grid=seq(0,NT,by = dt)
```

4.5 Comparison with solver vs. discretization

Let's visualize SDE solver solutions and discretized solutions together.

As can be seen below two graphs, it is clear that the discretized scheme gives correct answer. Top and bottom lines indicates 0.975, 0.025 quantiles, respectively. The mid line shows median quantile.


Red: discretized solution, Blue: SDE solver
x1
time
0
2
4
6
8
10
12
−3
−2
−1
0
1
2
3


Red: discretized solution, Blue: SDE solver
x2
time
0
2
4
6
8
10
12
−4
−2
0
2
4

4.6 Discretization with normalization over states, parameters, and time.

Let's have a normalization. We will normalize the parameters in a system by a certain large number (so that all parameters are in $[0, 1]$).

$$x_1^* = \frac{x_1}{L_x},\ x_2^* = \frac{x_2}{L_x},\ \mathbf{A}^* = \frac{\mathbf{A}}{L_A},\ \tau = \frac{t}{\Delta t}$$

And thus, the system of equations can be re-written

$$\frac{dx_1}{dt} = A_{11}x_1 + A_{12}x_2 + \sigma_1\frac{\omega_1}{dt}$$
$$\frac{dx_2}{dt} = A_{21}x_1 + A_{22}x_2 + \sigma_2\frac{\omega_2}{dt}$$

as below..

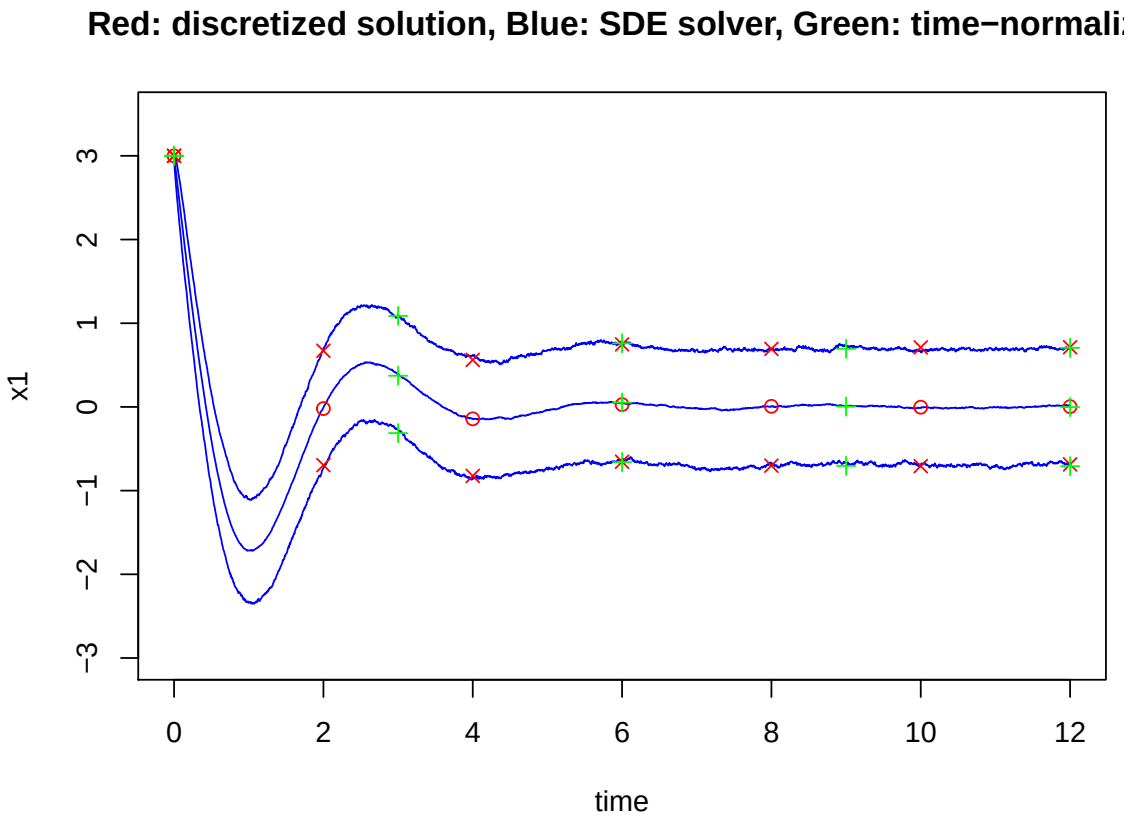
$$\frac{dx_1}{d\tau} = \Delta t L_A(A_{11}^*x_1^* + A_{12}^*x_2^*) + \frac{L_\sigma}{L_x}\sigma_1\frac{\omega_1^*}{d\tau}$$
$$\frac{dx_2}{d\tau} = \Delta t L_A(A_{21}^*x_1^* + A_{22}^*x_2^*) + \frac{L_\sigma}{L_x}\sigma_2\frac{\omega_2^*}{d\tau}$$

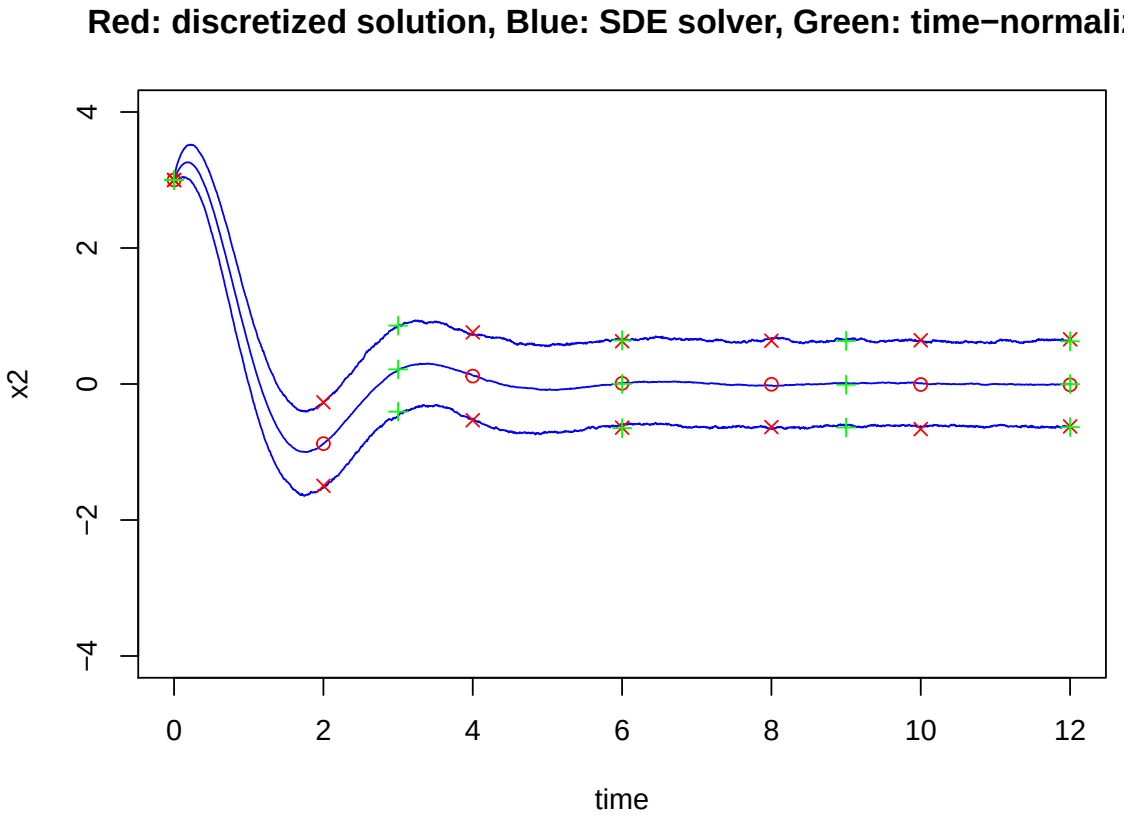
All the things fine. However, when we calculate $\mathbf{Q}_d$, we need to be careful. After we do all discretization, we have $\varepsilon_1 \sim N(0, \sigma_{d,1}^2)$. However, in this case, it should be $\varepsilon_1 \sim \sqrt{\Delta t}N(0, \sigma_{d,1}^2)$. Because ω in time t is the standard wiener process. The wiener process satisfies this relationship $(\omega(t) - \omega(s) \sim \mathcal{N}(0, t-s))$. For the standard wiener process, $\omega((n+1)t) - \omega((n)t) \sim \mathcal{N}(0, 1^2)$. In a τ scale, $\omega((n+1)\tau) - \omega((n)\tau) \sim \mathcal{N}(0, \frac{1}{\Delta t})$. Since $\sqrt{\Delta t}\mathcal{N}(0, \frac{1}{\Delta t}) = \mathcal{N}(0, 1)$, we need to multiply $\sqrt{\Delta t}$ to $N(0, \sigma_{d,1}^2)$ when we normalize original time scale.

```
L_x=5
L_A=3
L_sd=0.5
NT=12

# A matrix
A=matrix(c(-0.5,2,-2,-1),ncol=2)
# Diffusion matrix of process (state) noise
Q=matrix(c(0.5^2,0,0,0.3^2),ncol=2)

As=A/L_A # Astar
Qs=Q/(L_sd^2) #Qstar
nx=dim(As)[1]
# discretized time interval
dt=3. # Delta t

dtau=1 # tau scale.. tau=1,2,3,...
# Qd[1,1]/Qsd[1,1]
# discretized A matrix
Asd=(Matrix::expm(dt*L_A*As*dtau))%>%as.matrix()
# (L_sd^2/(L_x^2)*q) this is new Q
# (dt*L_A*a) this is new A
Qsd_temp=Matrix::expm(rbind(cbind(-(dt*L_A*As),(L_sd^2/(L_x^2)*Qs)),cbind(Qs*0,t((dt*L_A*As))))*dtau) #
# distretized Q matrix
Qsd=(Asd%*%Qsd_temp[1:nx,(nx+1):(2*nx)])%>%as.matrix()

# initial starting point
xs0=c(3,3)/L_x
```

```
# number of particles to do montecarlo simulation
NP=5000
# store data
xs_array=array(0,c(length(xs0),NP,(NT/dt/dtau)+1))
# initial point

xs_array[,,1]=matrix(rep(xs0,NP),ncol=NP)

for(i in 2:dim(xs_array)[3]){
  xs_array[,,i]=Asd%*%xs_array[,,i-1]+t(mvtnorm::rmvnorm(NP,mean=c(0,0),sigma=Qsd))*sqrt(dt)
}




x1dq_star=(xs_array[1,,]*L_x)%>%apply(MARGIN=c(2),function(x) quantile(x,probs=c(0.025,0.5,0.975)))
x2dq_star=(xs_array[2,,]*L_x)%>%apply(MARGIN=c(2),function(x) quantile(x,probs=c(0.025,0.5,0.975)))
tsd_grid=seq(0,NT,by = dt)
```

As can be seen below figures, the discretization is working when we normalize states, parameters, and time.


Red: discretized solution, Blue: SDE solver, Green: time−normali
x1
3
2
1
0
−1
−2
−3
0
2
4
6
8
10
12
time


Red: discretized solution, Blue: SDE solver, Green: time−normali
x2
4
2
0
−2
−4
0
2
4
6
8
10
12
time

5. Reference

- (Sarkka, 2019): Särkkä, S., & Solin, A. (2019). Applied stochastic differential equations (Vol. 10). Cambridge University Press.
- (Loan, 1978): Loan, C. Van. (1978). Computing integrals involving the matrix exponential. IEEE Transactions on Automatic Control, 23(3), 395–404. https://doi.org/10.1109/TAC.1978.1101743