SDEApproximation

2024-10-29

dX = u(t, X(t))dt + sig(t, X(t))dB(t) goal: find X(t) numerically Mathematically we know the answer. We can check the error at the end. Define Parameters

```
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
```

```
library(ggplot2)
# Define parameters
T = 1                  # End time
N = 1000               # Number of time steps
dt = T / N             # Time step size
X0 = 1                 # Initial condition
num_paths = 1000       # Number of Monte Carlo simulations
```

Define drift and diffusion functions. We are assuming constant mu and sigma for simplicity+testing purposes

```
mu = function(t, X) { return(0.5) }
sigma = function(t, X) { return(0.2) }
```

Initialize matrices. Each row will correspond to a different brownian motion. We can initialize X0 to be the first column for both the analytical and tested solutions.

```
X = matrix(0, nrow=num_paths, ncol=N+1)
X_analytical = matrix(0, nrow=num_paths, ncol=N+1)
X[,1] = X0
X_analytical[,1] <- X0
```

Initialize E[X] to be a vector of the length of the matrix for further analysis.

```
# Arrays to store expectations
E_X_numerical = numeric(N+1)
E_X_analytical = numeric(N+1)
E_X_numerical[1] = X0
E_X_analytical[1] = X0
```

Same idea

```
# Arrays to store errors
mean_abs_error = numeric(N+1)
rmse = numeric(N+1)
```

Using a simple Euler scheme, for each brownian motion we can calculate the numerical and analytical solution. Increment time by dt and dB is estimated to be the square root of time multiplied by a random variable Z. The analytical solution is derived by Ito calculus, we guess X= f(t, B(t)) and proceed backward to solve simple ODEs.

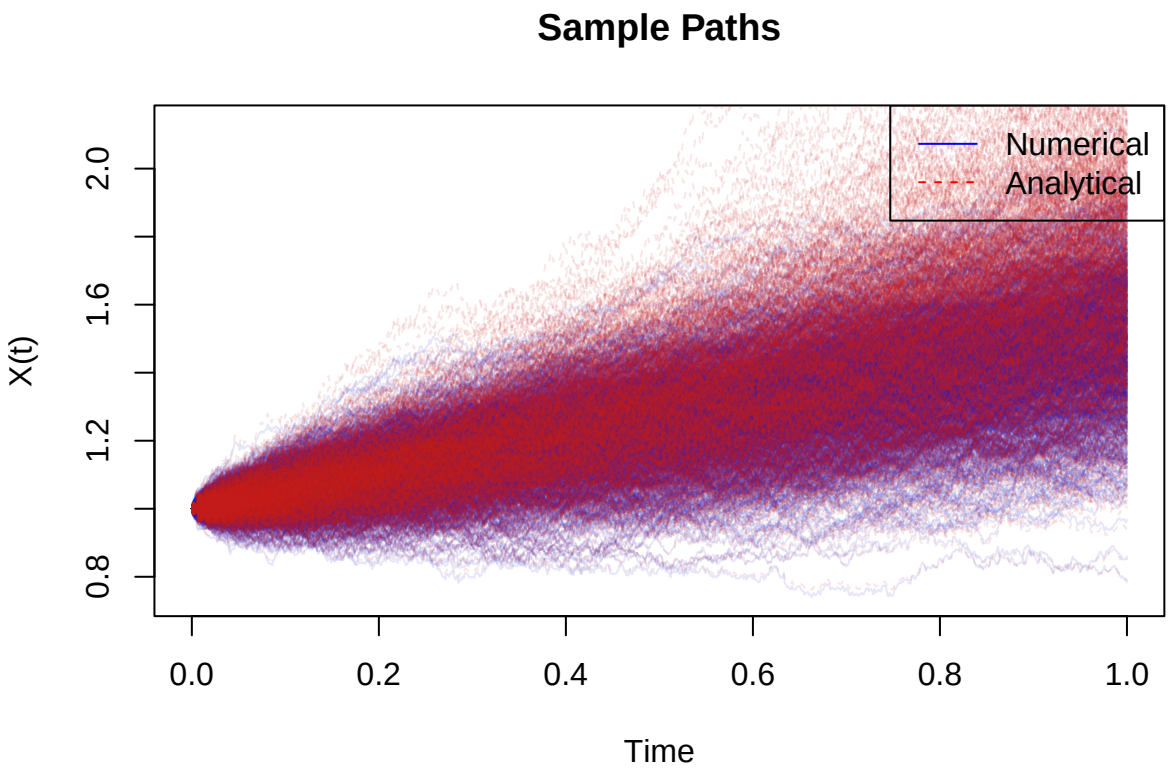
```
# Simulate paths
for (i in 1:num_paths) {
  B_t = 0
  for (j in 1:N) {
    t = j * dt
    dB = sqrt(dt) * rnorm(1)
    B_t = B_t + dB

    # Numerical solution (Euler-Maruyama)
    X[i, j+1] = X[i, j] + mu(t, X[i, j]) * dt + sigma(t, X[i, j]) * dB

    # Analytical solution
    X_analytical[i, j+1] = X0 * exp(sigma(t, X0) * B_t + (mu(t, X0) - 0.5 * sigma(t, X0)^2) * t)
  }
}
```

Take the mean of each time point, this is a good representation of each brownian motion. Using this we can easily get our RMSE and absolute difference.

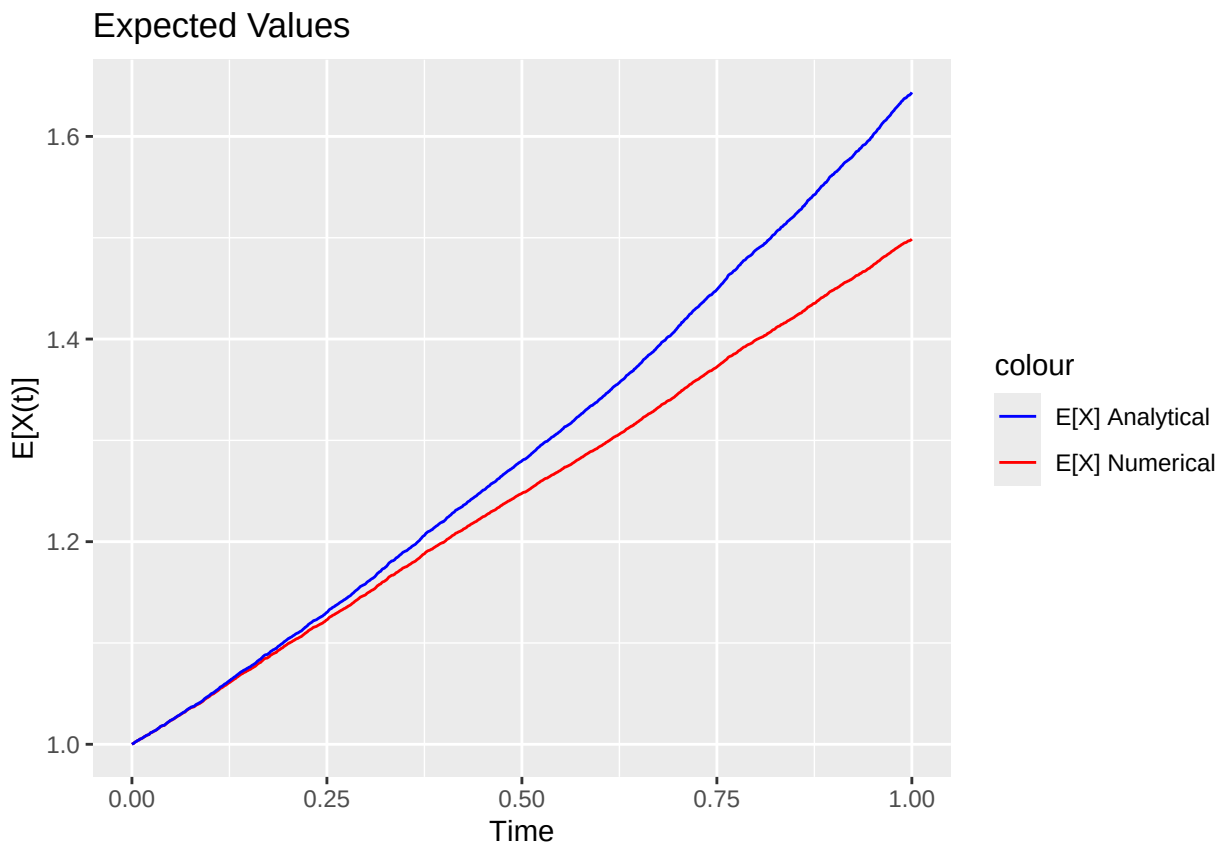
```
# Calculate expectations at each time point
for (j in 1:(N+1)) {
  E_X_numerical[j] = mean(X[, j])
  E_X_analytical[j] = mean(X_analytical[, j])

  # Calculate errors
  mean_abs_error[j] = mean(abs(X[, j] - X_analytical[, j]))
  rmse[j] = sqrt(mean((X[, j] - X_analytical[, j])^2))
}
```

We can visualize the data using ggplot2 and matplot for our matrices. Here matline overlaps the underlying plot so we can see the difference between the two plots.

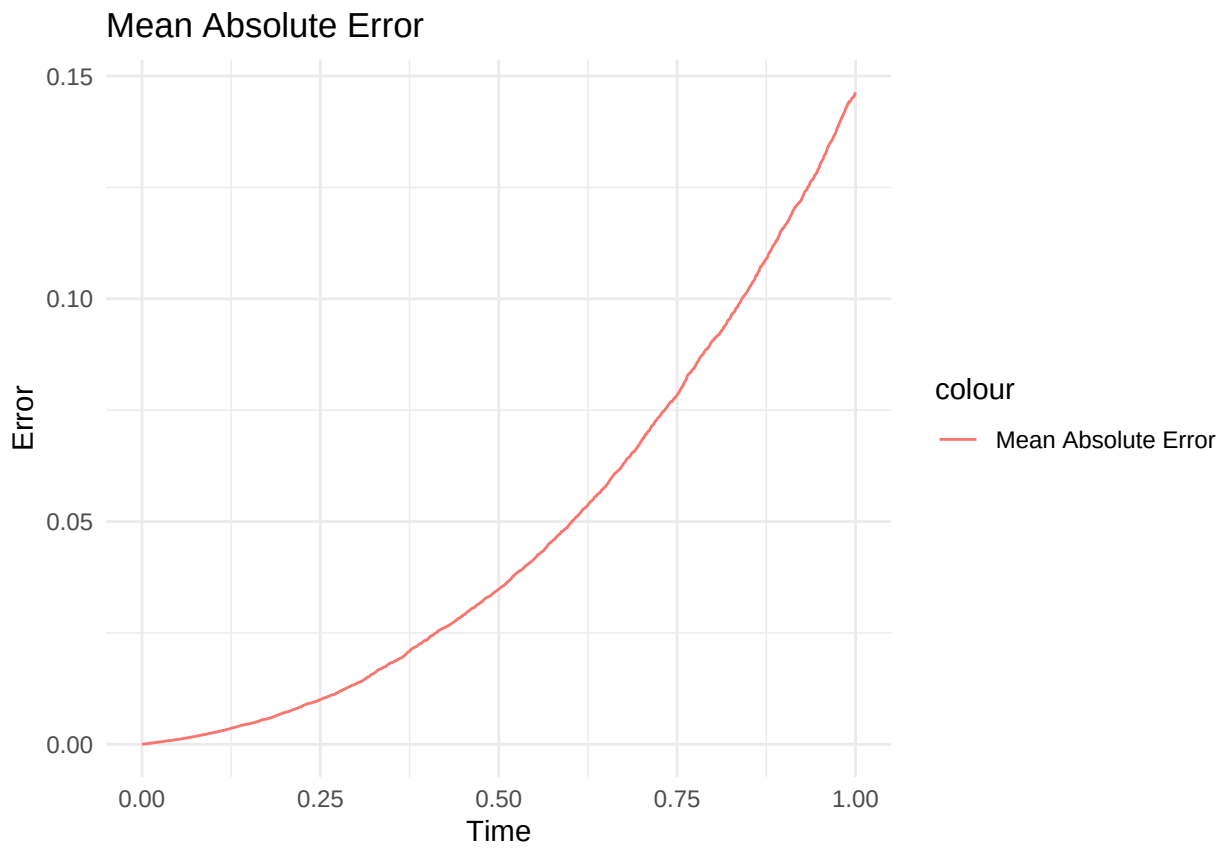
```
time = seq(0, T, by=dt)
matplot(time, t(X), type='l', lty=1, col=rgb(0.1, 0.1, 0.8, 0.1),
        main="Sample Paths", xlab="Time", ylab="X(t)")
matlines(time, t(X_analytical), type='l', lty=2, col=rgb(0.8, 0.1, 0.1, 0.1))
legend("topright", legend=c("Numerical", "Analytical"), lty=c(1,2), col=c("blue", "red"))
```



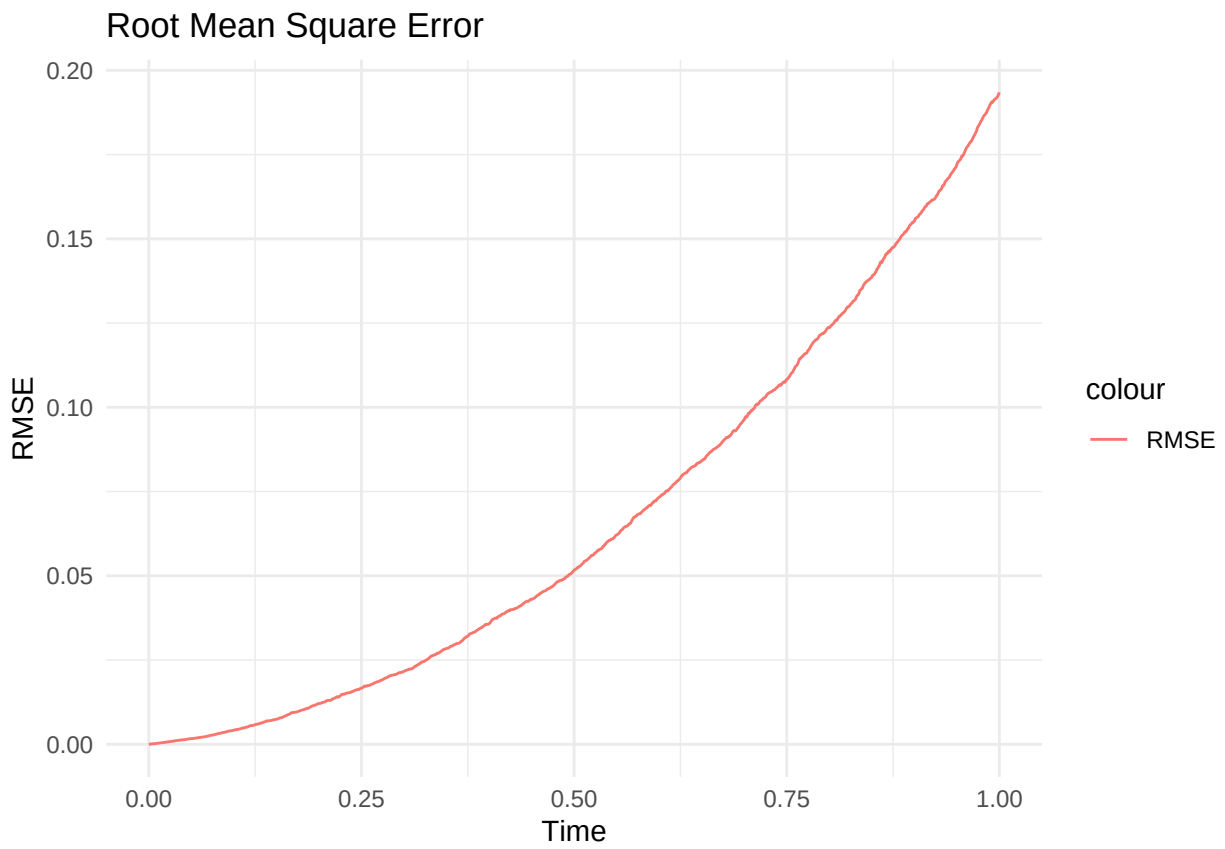

```
p2 = ggplot() +
  geom_line(aes(x=time, y=E_X_numerical, color="E[X] Numerical")) +
  geom_line(aes(x=time, y=E_X_analytical, color="E[X] Analytical")) +
  labs(title="Expected Values", x="Time", y="E[X(t)]") +
  scale_color_manual(values=c("blue", "red"))
p2
```




```
p3 = ggplot() +
  geom_line(aes(x=time, y=mean_abs_error, color="Mean Absolute Error")) +
  labs(title="Mean Absolute Error", x="Time", y="Error") +
  theme_minimal()
p3
```




```
p4 = ggplot() +
  geom_line(aes(x=time, y=rmse, color="RMSE")) +
  labs(title="Root Mean Square Error", x="Time", y="RMSE") +
  theme_minimal()
p4
```




```
cat("Average Mean Absolute Error:", mean(mean_abs_error), "\n")
```

```
## Average Mean Absolute Error: 0.04764448
```

```
cat("Average RMSE:", mean(rmse), "\n")
```

```
## Average RMSE: 0.06673769
```