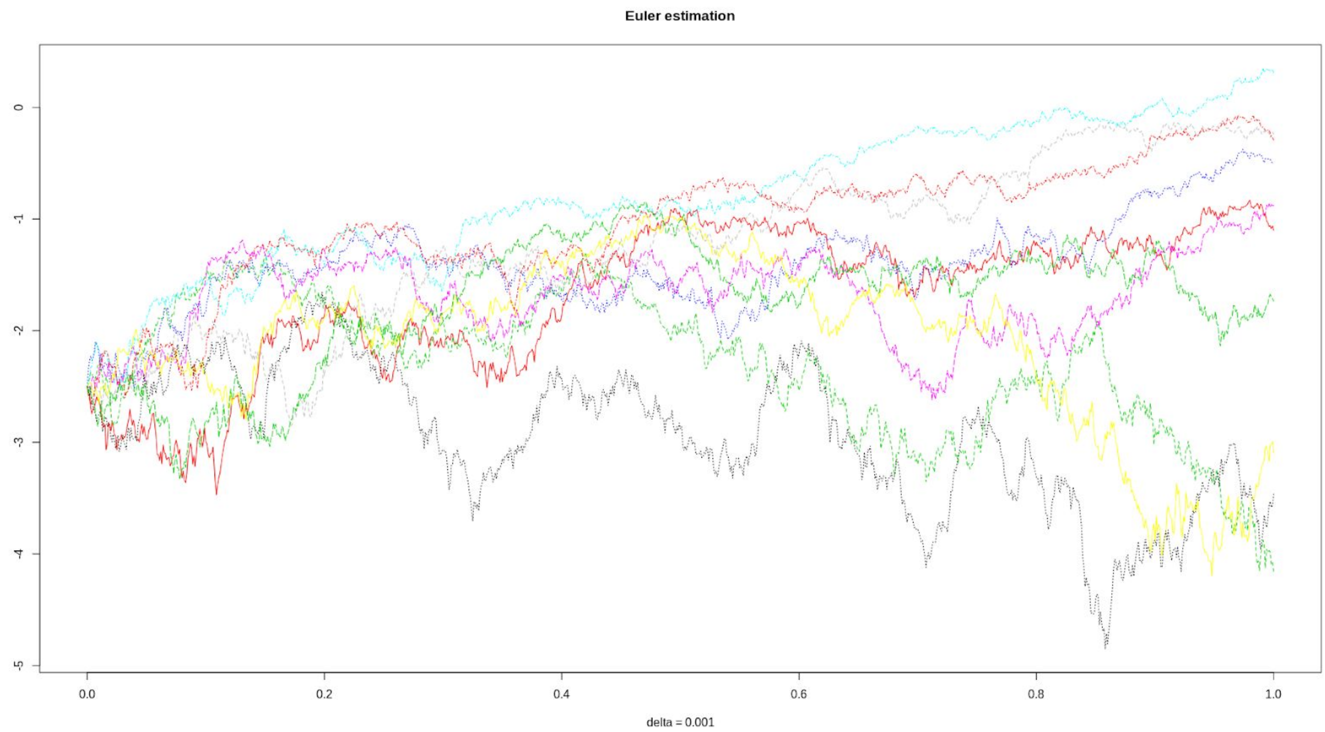
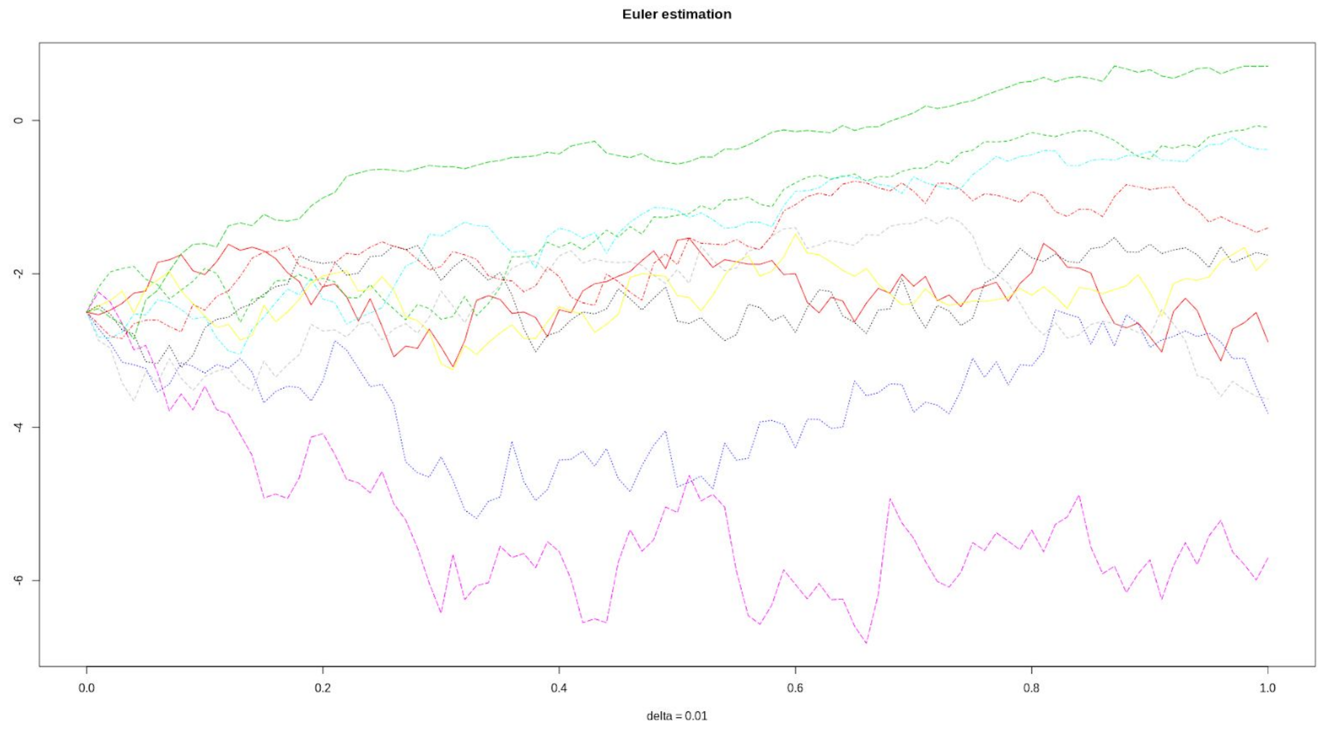
Part 1. Code in R

```
library(MASS)
Wien = function(num, delta){
  rn<-rnorm(n=num, m=0, sd=1)
  c(0, cumsum(rn))*sqrt(delta)}

Euler_est = function(delta, t_max, t_0=0){
  t = seq(t_0, t_max, by = delta)
  n = length(t)
  x = c(x0)
  w = Wien(n, delta)
  for(k in 1:(n-1))
        x<-c(x, x[k] + a(t[k], x[k])*delta + b(t[k],x[k])*(w[k+1]-w[k]))
  return(x)
}
x0 = -2.5
delta = 0.01 #0.001
a = function(t, x) atan(2 + x*t)
b = function(t, x) log(1 + t + x*x)
t = seq(0, 1, by = delta)
n = length(t)
x =  matrix(nrow=10, ncol=n)
for(i in 1:10)
  x[i,] = Euler_est(delta, 1)
matplot(t, t(x[1:10,]), col=c(2:11), type = "l", main = "Euler estimation", xlab = "delta = 0.001", ylab = "")
```


Euler estimation
0
-1
-2
-3
-4
-5
0.0
0.2
0.4
0.6
0.8
1.0
delta = 0.001

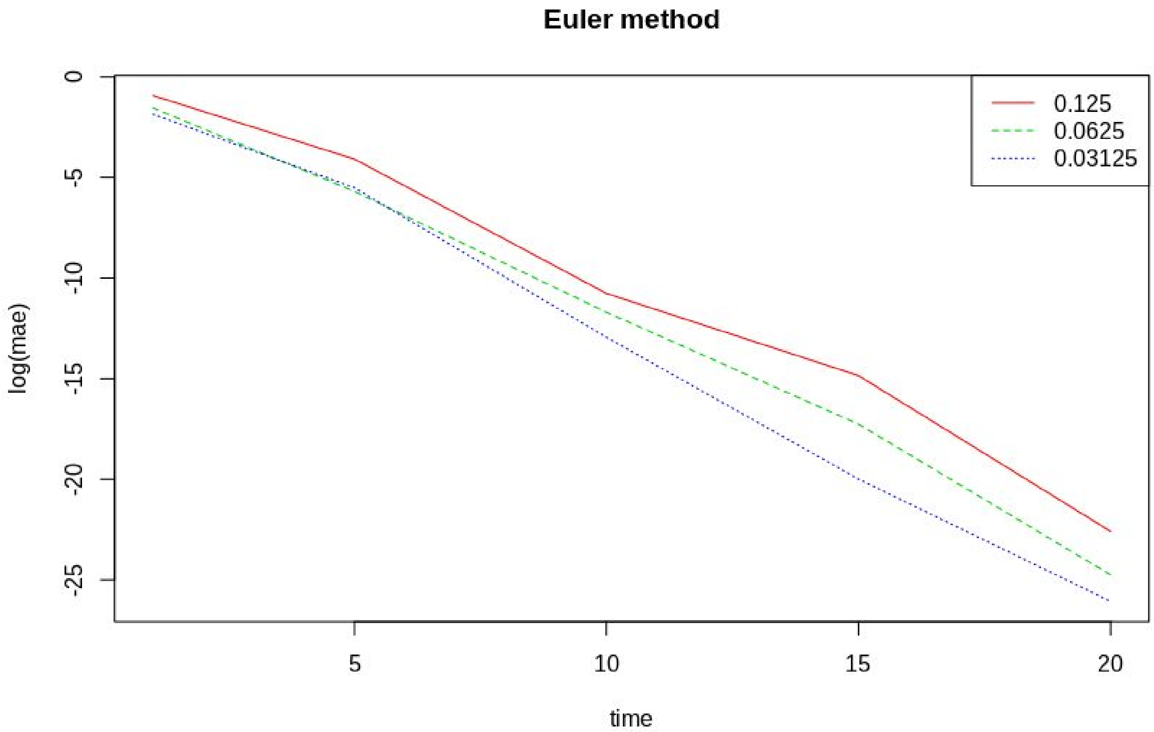

Euler estimation
0
-2
-4
-6
0.0
0.2
0.4
0.6
0.8
1.0
delta = 0.01

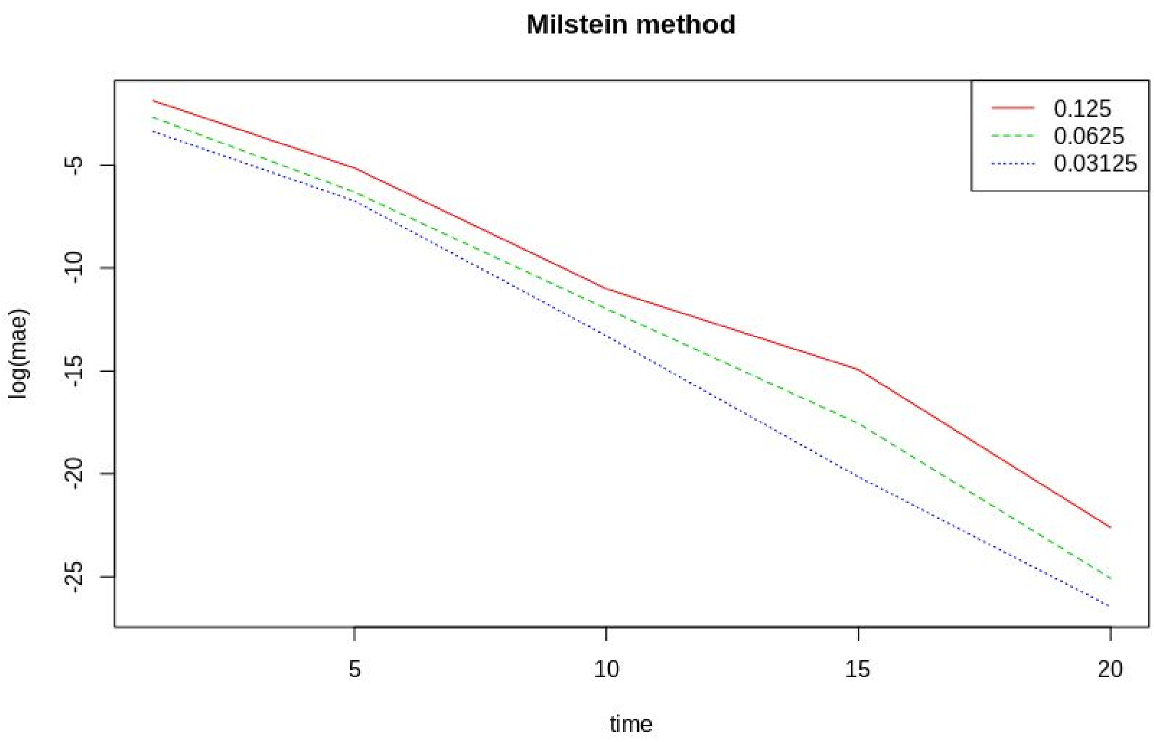
Part 2. Code in R

```
library(MASS)
Wien = function(num, delta){
  rn<-rnorm(n=num - 1, m=0, sd=1)
  c(0, cumsum(rn))*sqrt(delta)}

Euler_estimator = function(delta, w, t){
  n = length(t)
  x = c(x0)
  for(k in 1:(n-1))
           x<-c(x, x[k] + a(t[k], x[k])*delta + b(t[k],x[k])*(w[k+1]-w[k]))
  return(x)
}

Milstein_estimator = function(delta, w, t){
  n = length(t)
  x = c(x0)
  for(k in 1:(n-1))
           x<-c(x, x[k] + a(t[k], x[k])*delta + b(t[k],x[k])*(w[k+1]-w[k]) +
           1/2*b(t[k],x[k])*db_dx(t[k],x[k])*((w[k+1]-w[k])^2-delta))
  return(x)
}
calculate_mae = function(x_real, x_check){
  colMeans(abs(x_real - x_check))
}
Taylor_estimator = function(delta, w, t){
  n = length(t)
  x = c(x0)
  for(k in 1:(n-1)){
           delta_w = w[k+1]-w[k]
           delta_v = rnorm(1, 0, 1) * delta
           delta_z = 1/2*delta*(delta_w + delta_v/sqrt(3))
           x = c(x, x[k] + a(t[k], x[k])*delta + b(t[k],x[k])*delta_w +
           1/2 * b(t[k],x[k]) * db_dx(t[k],x[k]) * (delta_w^2-delta) +
           1/2 * a(t[k],x[k]) * da_dx(t[k],x[k]) * delta^2 +
           a(t[k],x[k])*db_dx(t[k],x[k]) * (delta_w*delta - delta_z) +
           1/2 * b(t[k],x[k]) * db_dx(t[k],x[k])^2 * (1/3*(delta_w)^2-delta)*delta_w)
  }
  return(x)
}

x0 = 2
t1 = c(1, 10, 20)
t_0 = 0
t_max = max(t1)
deltas = c(2**(-3), 2**(-4), 2**(-5))
X_real = function(t, w) x0*exp(-1.845*t -1.3*w)
```

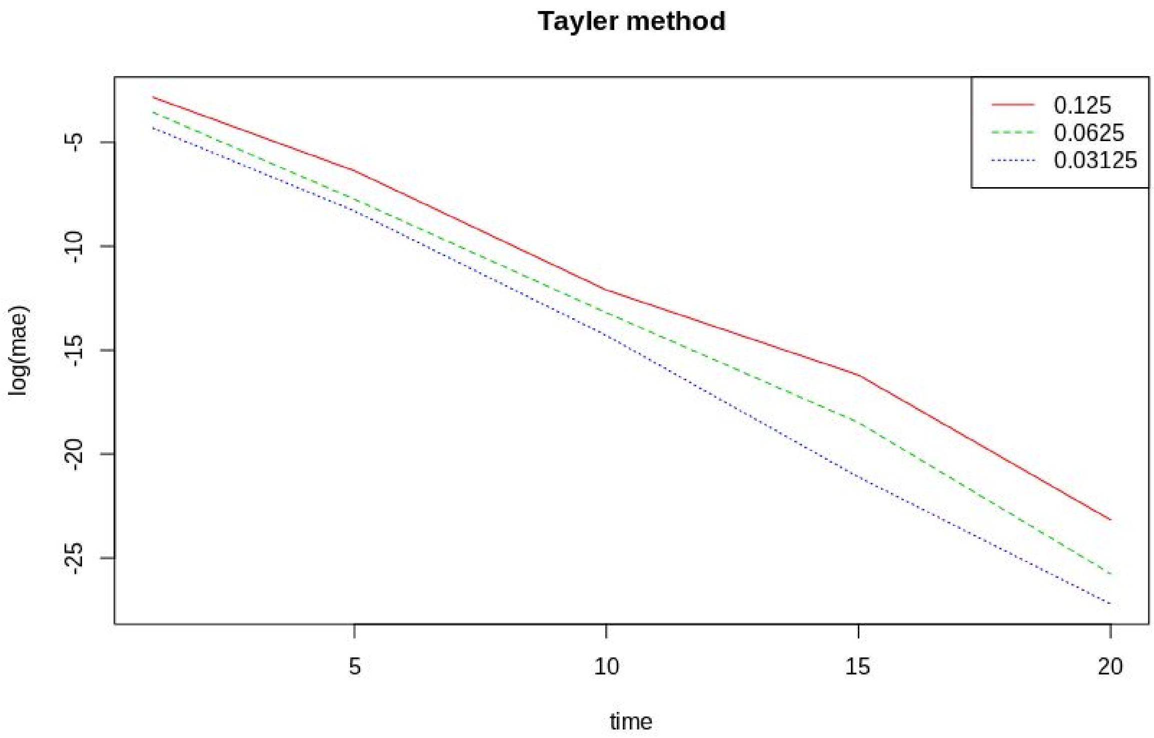
```
a = function(t, x) (-x)
b = function(t, x) (-1.3*x)
da_dx = function(t, x) (-1)
db_dx = function(t, x) (-1.3)

mae_t1_euler = matrix(nrow = length(deltas), ncol = length(t1))
mae_t1_milstein = matrix(nrow = length(deltas), ncol = length(t1))
mae_t1_taylor = matrix(nrow = length(deltas), ncol = length(t1))

for (d in 1:length(deltas))
{
 delta = deltas[d]
 N = 1000
 n = t_max / delta + 1

 x_real = matrix(nrow=N, ncol=n)
 x_euler =  matrix(nrow=N, ncol=n)
 x_milstein =  matrix(nrow=N, ncol=n)
 x_taylor =  matrix(nrow=N, ncol=n)

 for(i in 1:N){
      t = seq(t_0, t_max, by = delta)
      w = Wien(n, delta)
      x_real[i,] = X_real(t, w)
      x_euler[i,] = Euler_estimator(delta, w, t)
      x_milstein[i,] = Milstein_estimator(delta, w, t)
      x_taylor[i,] = Taylor_estimator(delta, w, t)
 }

 t1_index = t1 / delta
 mae_t1_euler[d,] = calculate_mae(x_real[, t1_index], x_euler[, t1_index])
 mae_t1_milstein[d,] = calculate_mae(x_real[, t1_index], x_milstein[, t1_index])
 mae_t1_taylor[d,] = calculate_mae(x_real[, t1_index], x_taylor[, t1_index])
}

matplot(t1, log(t(mae_t1_euler)), type = 'l', col = c(2:4), main = "Euler method", xlab = "time", ylab =
"log(mae)")
legend("topright", legend = deltas, lty = c(1:3), col = c(2:4))

matplot(t1, log(t(mae_t1_milstein)), type = 'l', col = c(2:4), main = "Milstein method", xlab = "time", ylab
= "log(mae)")
legend("topright", legend = deltas, lty = c(1:3), col = c(2:4))

matplot(t1, log(t(mae_t1_taylor)), type = 'l', col = c(2:4), main = "Tayler method", xlab = "time", ylab =
"log(mae)")
legend("topright", legend = deltas, lty = c(1:3), col = c(2:4))

matrix_compare_methods = matrix(nrow = length(deltas), ncol = 3)
matrix_compare_methods[,1] = mae_t1_euler[,2]
matrix_compare_methods[,2] = mae_t1_milstein[,2]
matrix_compare_methods[,3] = mae_t1_taylor[,2]
```

```
matplot(deltas, log(matrix_compare_methods), type = 'l', col = c(2:5), main = "Compare methods for fixed time = 5", xlab = "delta", ylab = "")
legend("topright", legend = c("Euler", "Milstein", "Taylor 1.5"), lty = c(1:3), col = c(2:4))

matrix_compare_methods = matrix(nrow = 3, ncol = length(t1))
matrix_compare_methods[1,] = mae_t1_euler[2,]
matrix_compare_methods[2,] = mae_t1_milstein[2,]
matrix_compare_methods[3,] = mae_t1_taylor[2,]

matplot(t1, log(t(matrix_compare_methods)), type = 'l', col = c(2:5), main = "Compare methods for fixed delta = 2^(-4)", xlab = "time", ylab = "log(mae)")
legend("topright", legend = c("Euler", "Milstein", "Taylor 1.5"), lty = c(1:3), col = c(2:4))
```

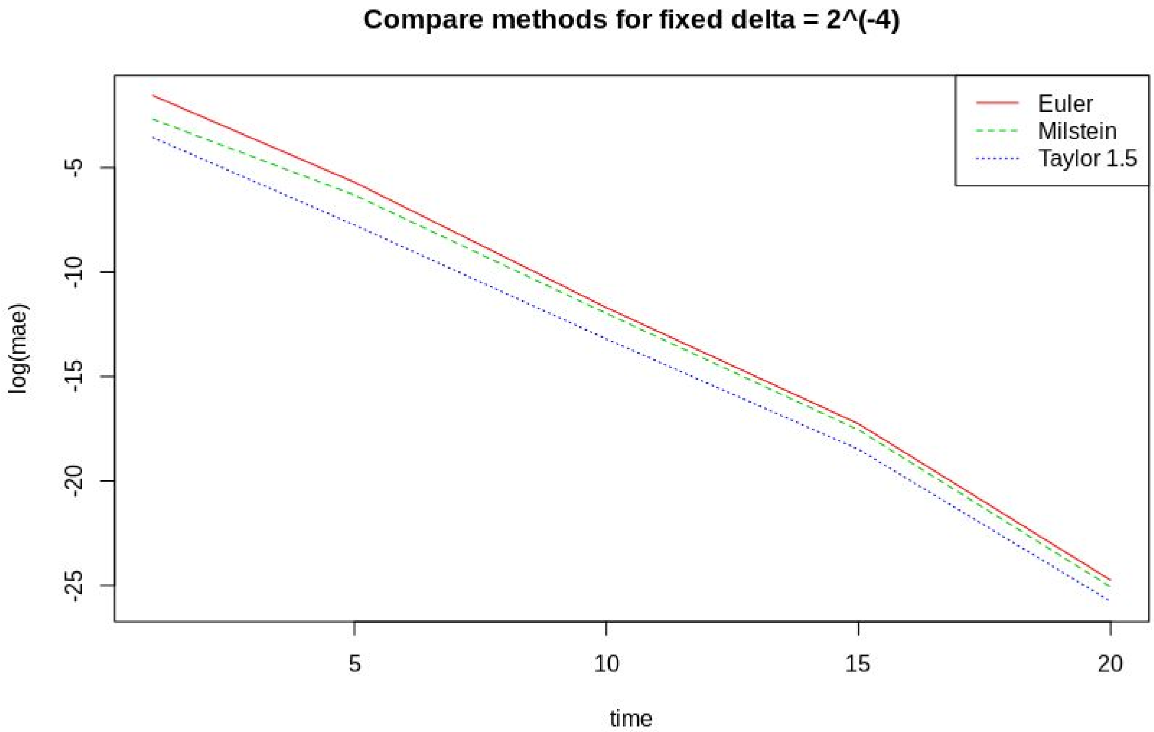


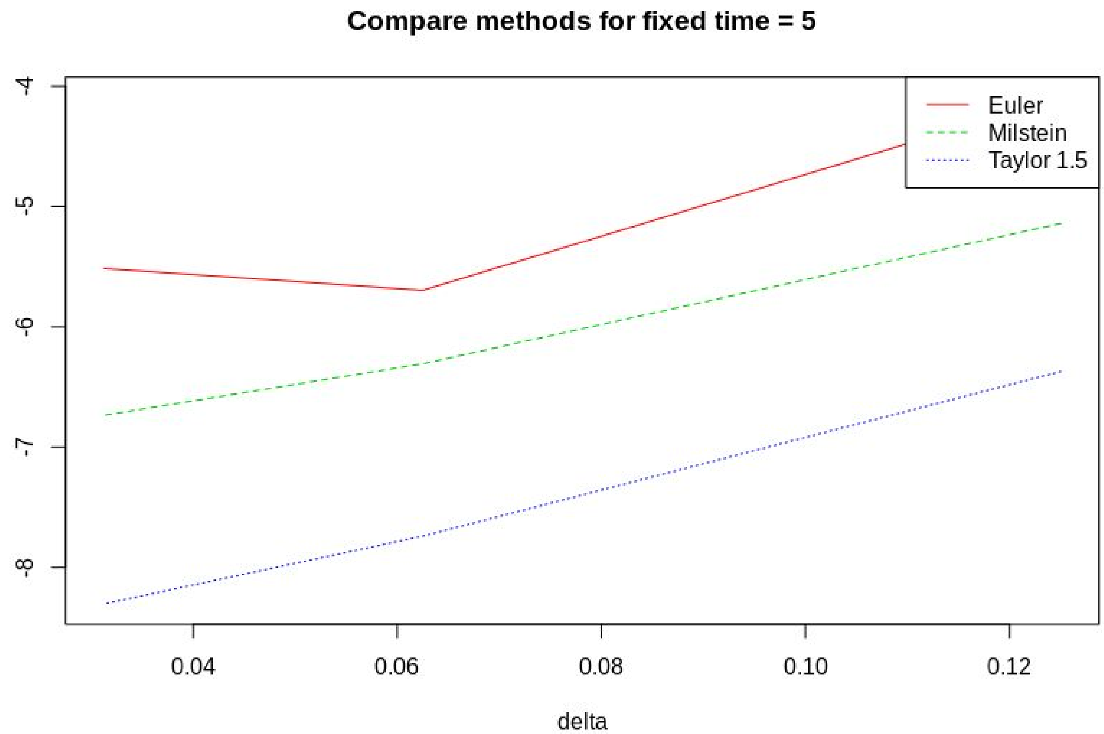


Milstein method
0.125
0.0625
0.03125
log(mae)
-5
-10
-15
-20
-25
5
10
15
20
time



Tayler method
0.125
0.0625
0.03125
log(mae)
-5
-10
-15
-20
-25
5
10
15
20
time


Compare methods for fixed delta = 2^(-4)
Euler
Milstein
Taylor 1.5
log(mae)
-5
-10
-15
-20
-25
5
10
15
20
time


Compare methods for fixed time = 5
Euler
Milstein
Taylor 1.5
-4
-5
-6
-7
-8
0.04
0.06
0.08
0.10
0.12
delta