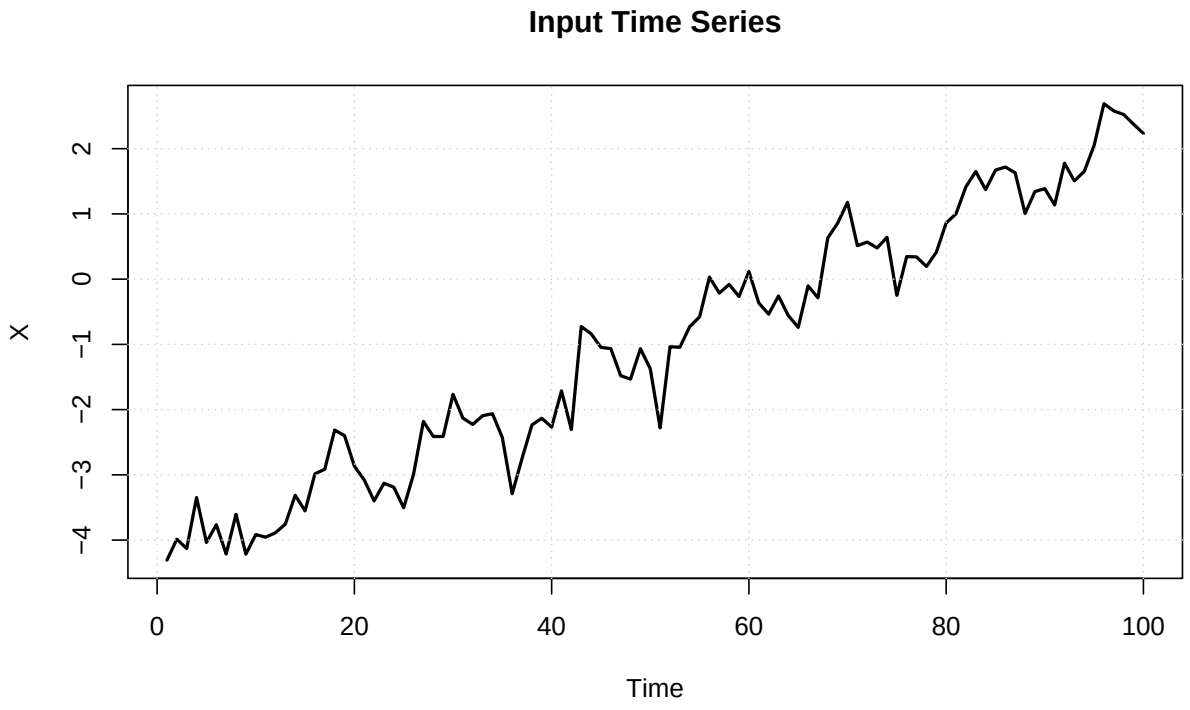
Data Simulation Demonstration

Three Approaches for Training Data Simulation

The Neural Network Imputer relies on these four libraries specifically:

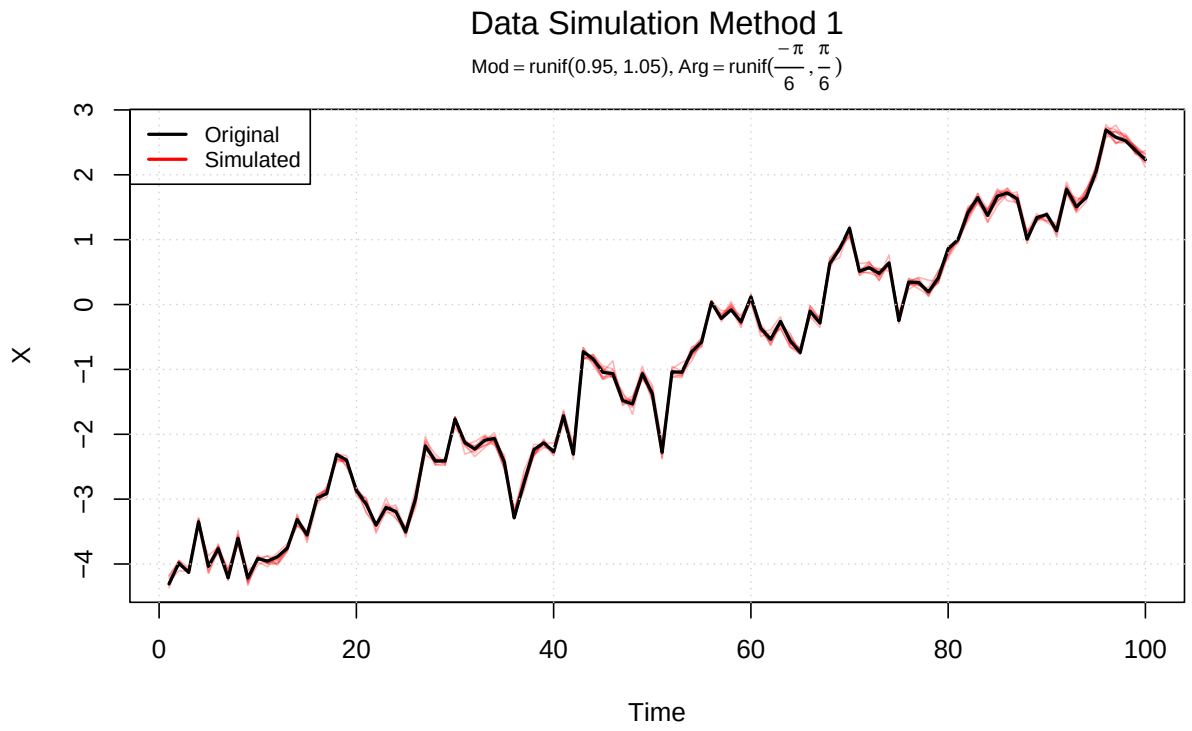
```
library(tsinterp)
library(interpTools)
library(tensorflow)
library(keras)
```



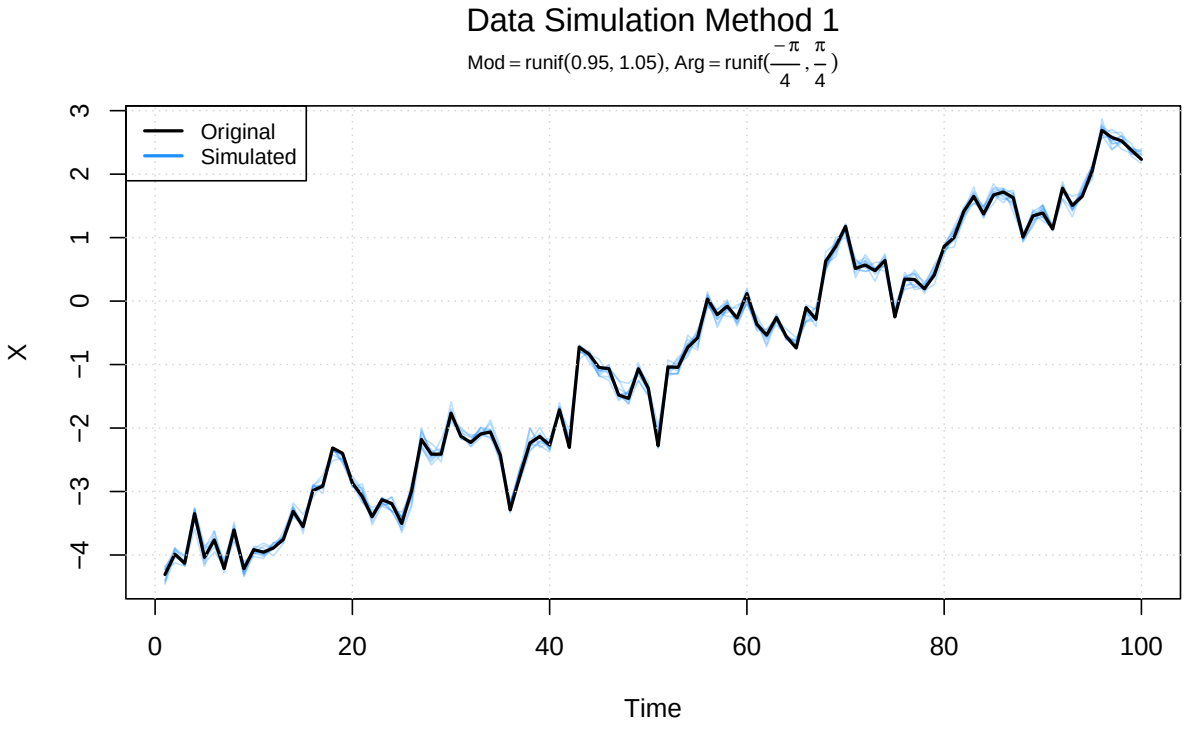
Method 1:

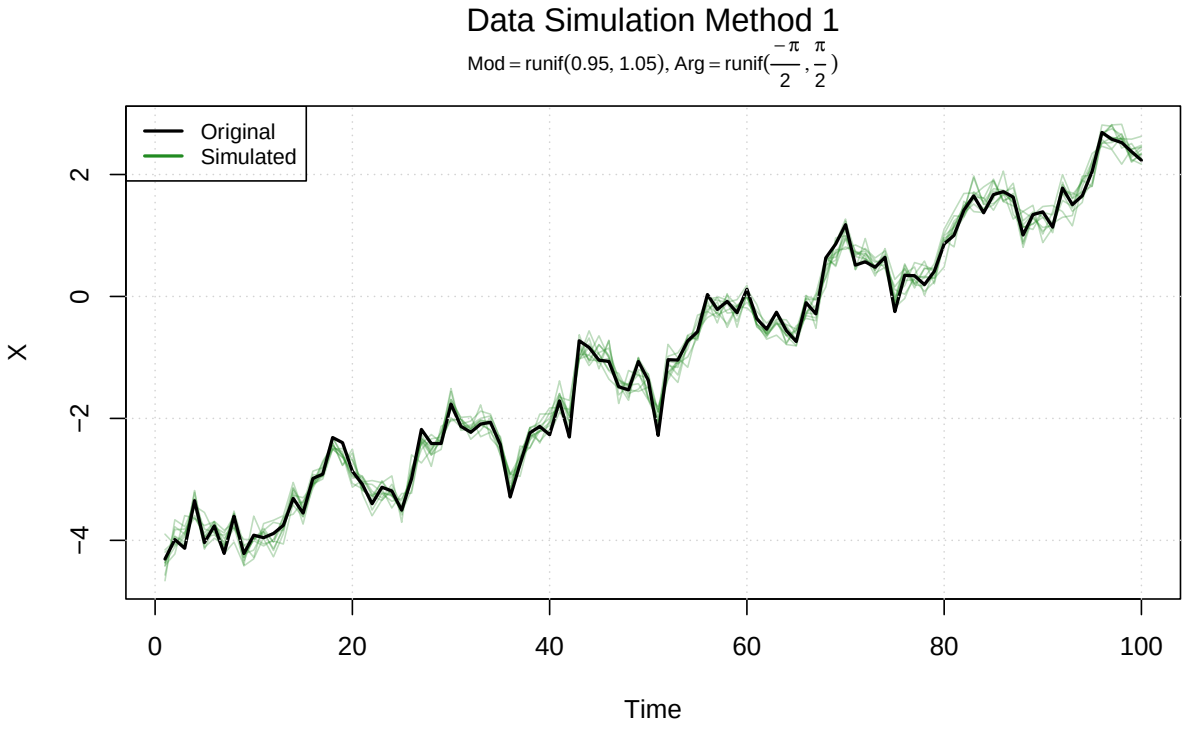
0. Input a complete time series.
1. Estimate the trend and periodic component(s).
2. Compute the residual noise between the estimate series and the input series.
3. Perturb the noise series (using multiplication of complex vector).
4. Add the perturbed noise back to the estimate series.

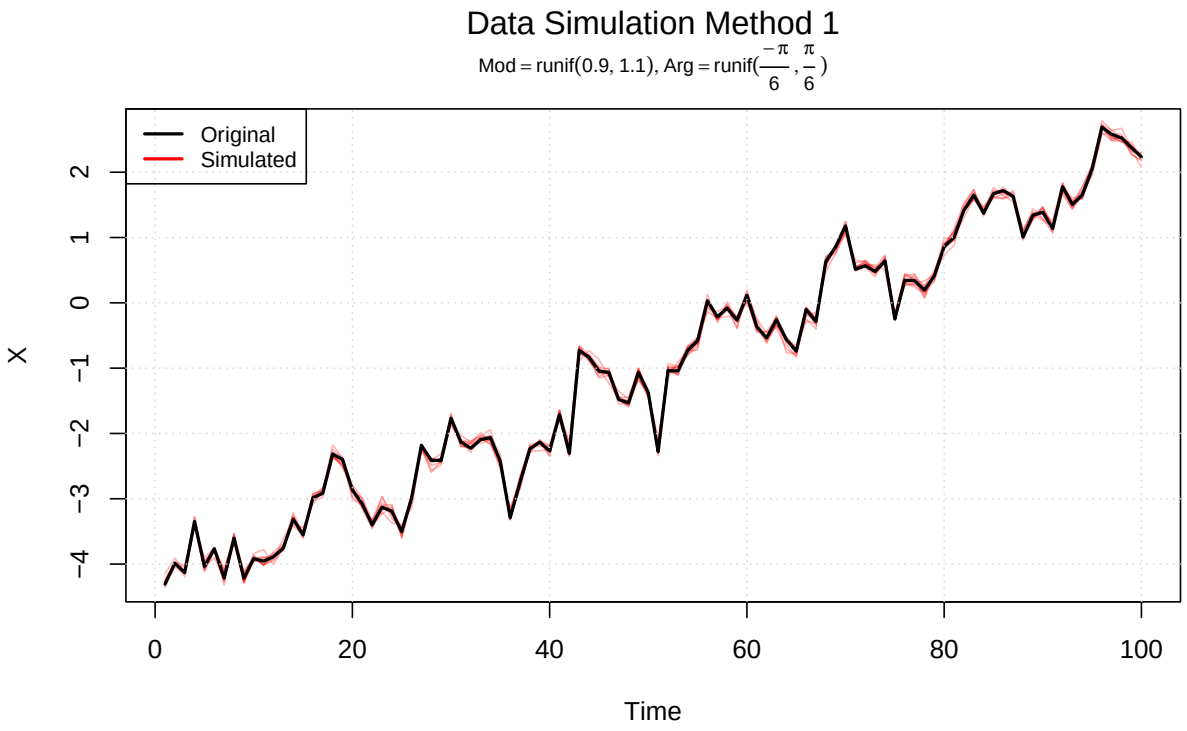
```
xE = estimator(x, method = 'Xt')       ## Estimate trend + periodic and noise components
w = x - xE                             ## Computing the residual noise
w_f = fft(w, inverse = FALSE)          ## Converting noise to frequency domain
inputs = c()                           ## Initializing vector to store values

Mod_a = 0.95; Mod_b = 1.05             ## Defining Modulus and Argument values
Arg_a = -pi/6; Arg_b = pi/6

for (i in 1:10){

  ## Creating small perturbation
  w_p = w_f * complex(modulus = runif(N, Mod_a, Mod_b), argument = runif(N, Arg_a, Arg_b))

  w_t = as.numeric(fft(w_p, inverse = TRUE)) / N  ## Converting back to time domain
  x_p = xE + w_t                                 ## Adding perturbed noise back
  inputs = c(inputs, x_p)                        ## Saving the new series
}
```

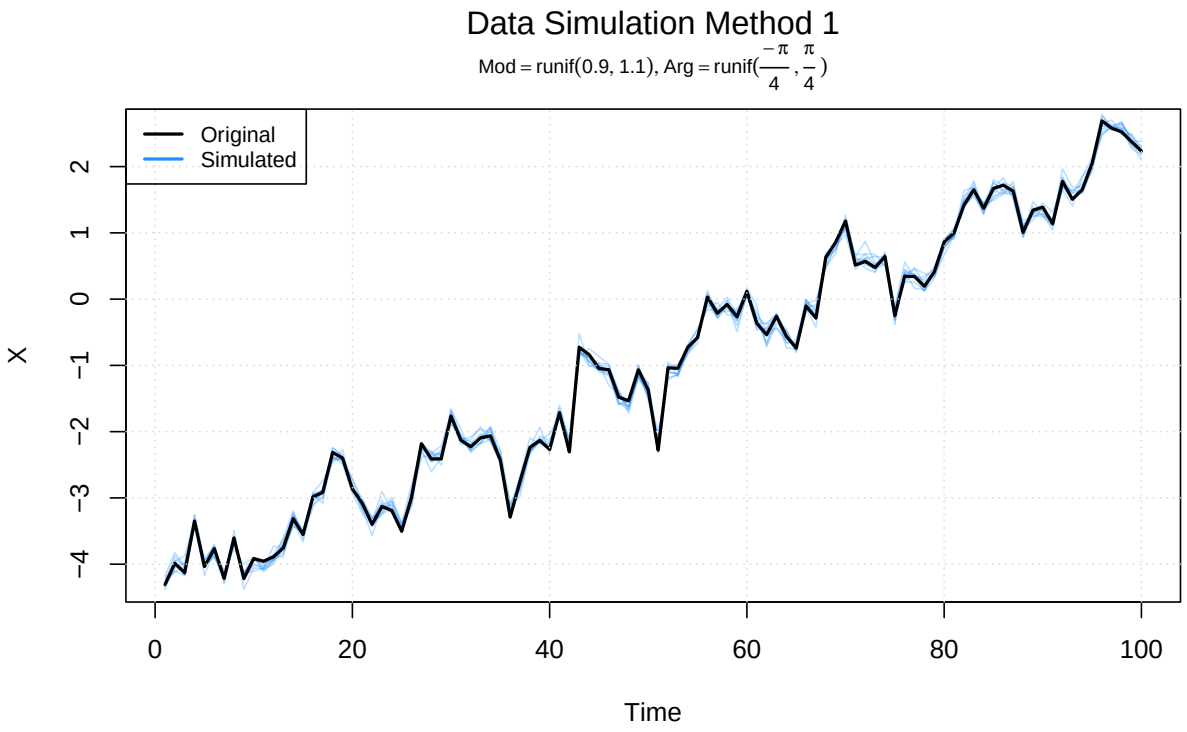


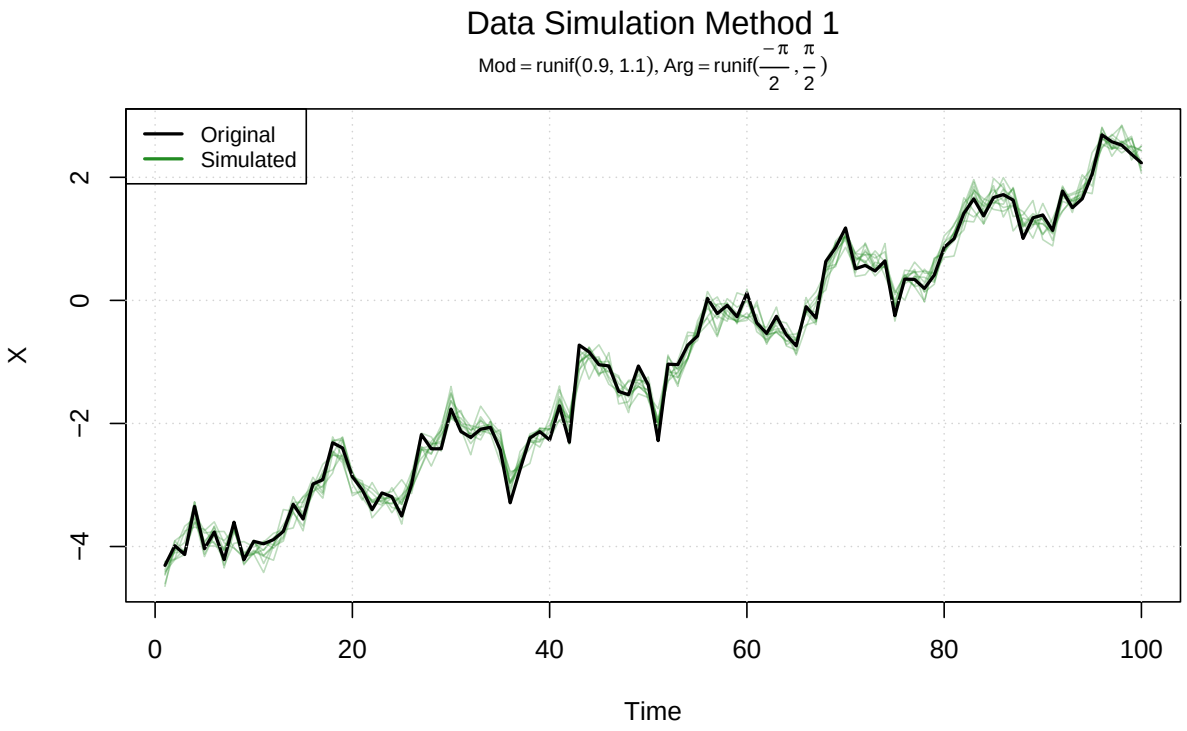
Method 1 w/ different Mod and Arg values:






Data Simulation Method 1
Mod = runif(0.9, 1.1), Arg = runif(−π/6, π/6)
Original
Simulated
X
Time
0
20
40
60
80
100
−4
−3
−2
−1
0
1
2

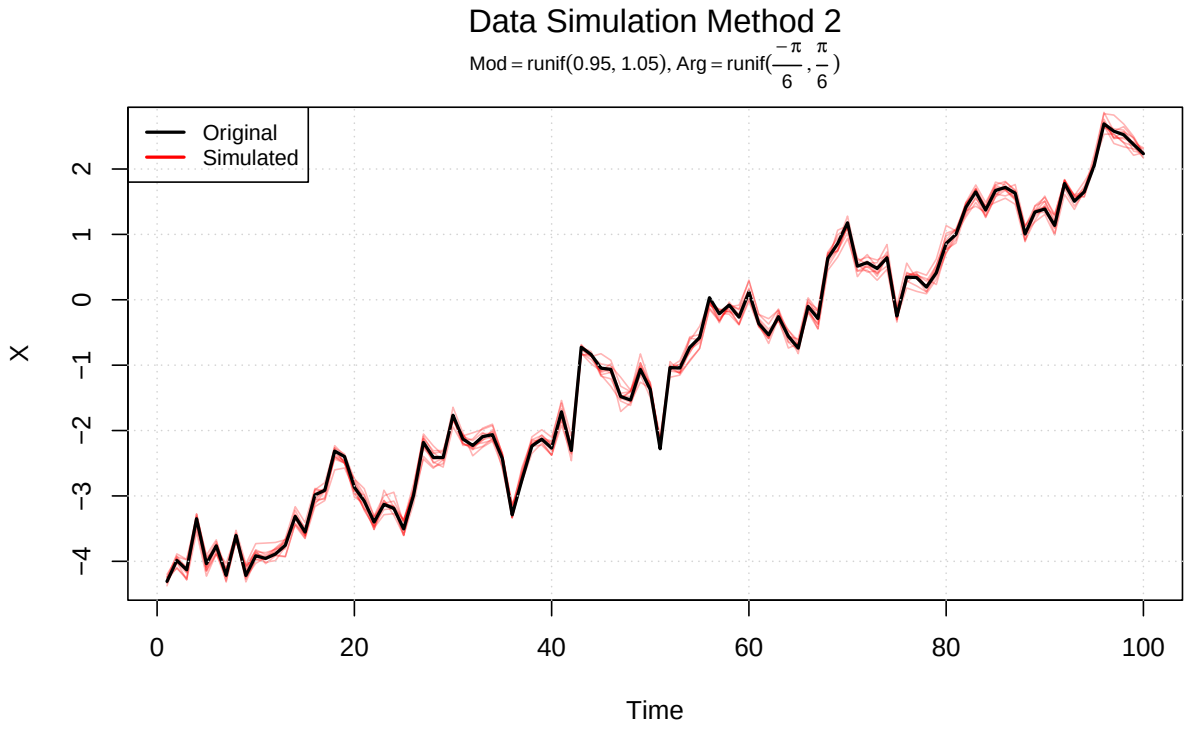

Data Simulation Method 1
Mod = runif(0.9, 1.1), Arg = runif(−π/4, π/4)
Original
Simulated
X
Time
0
20
40
60
80
100
−4
−3
−2
−1
0
1
2


Data Simulation Method 1
Mod = runif(0.9, 1.1), Arg = runif(−π/2, π/2)
Original
Simulated
X
2
0
−2
−4
0
20
40
60
80
100
Time

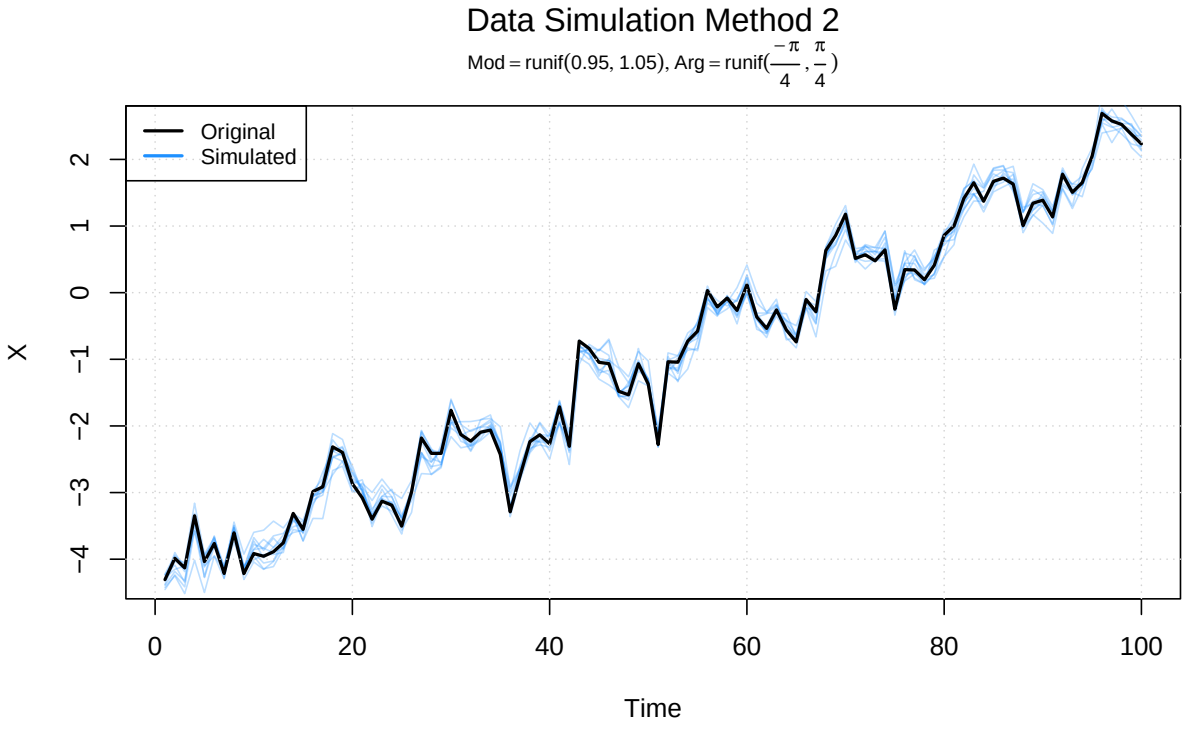
Method 2:

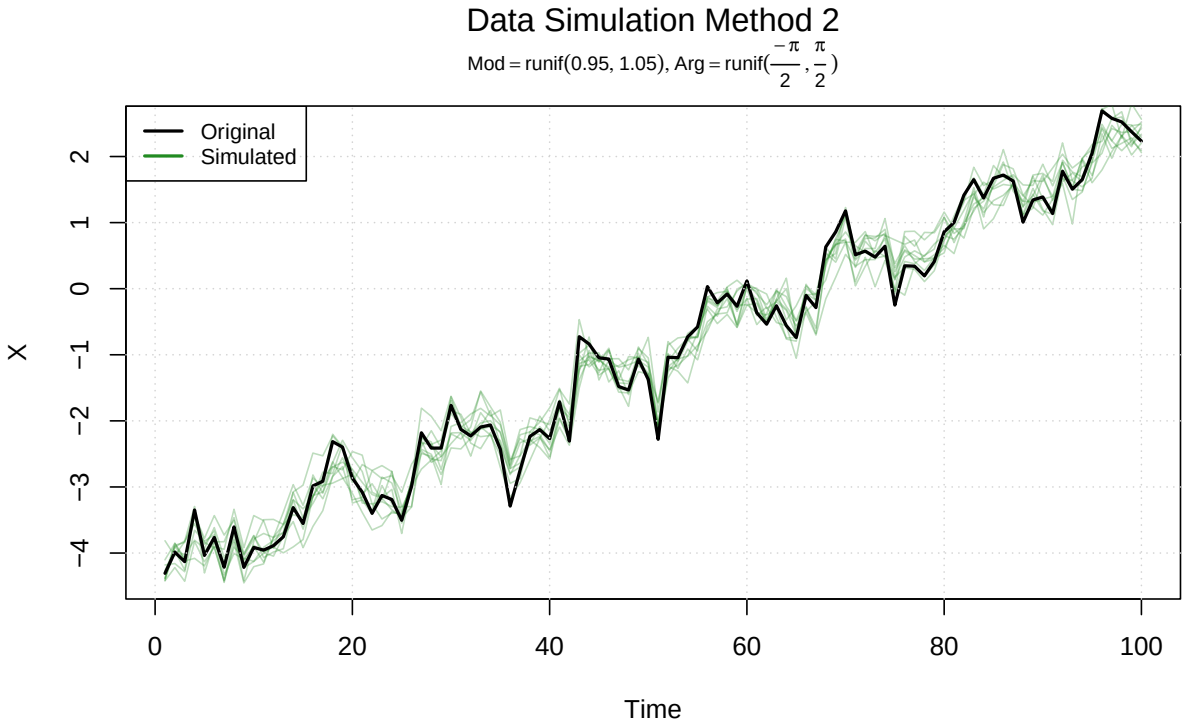
0. Input a complete time series.
1. Estimate trend component.
2. De-trend and zero-mean the input series.
3. Perturb the de-trended series (using multiplication of complex vector).
4. Add the mean and trend back to the perturbed series.

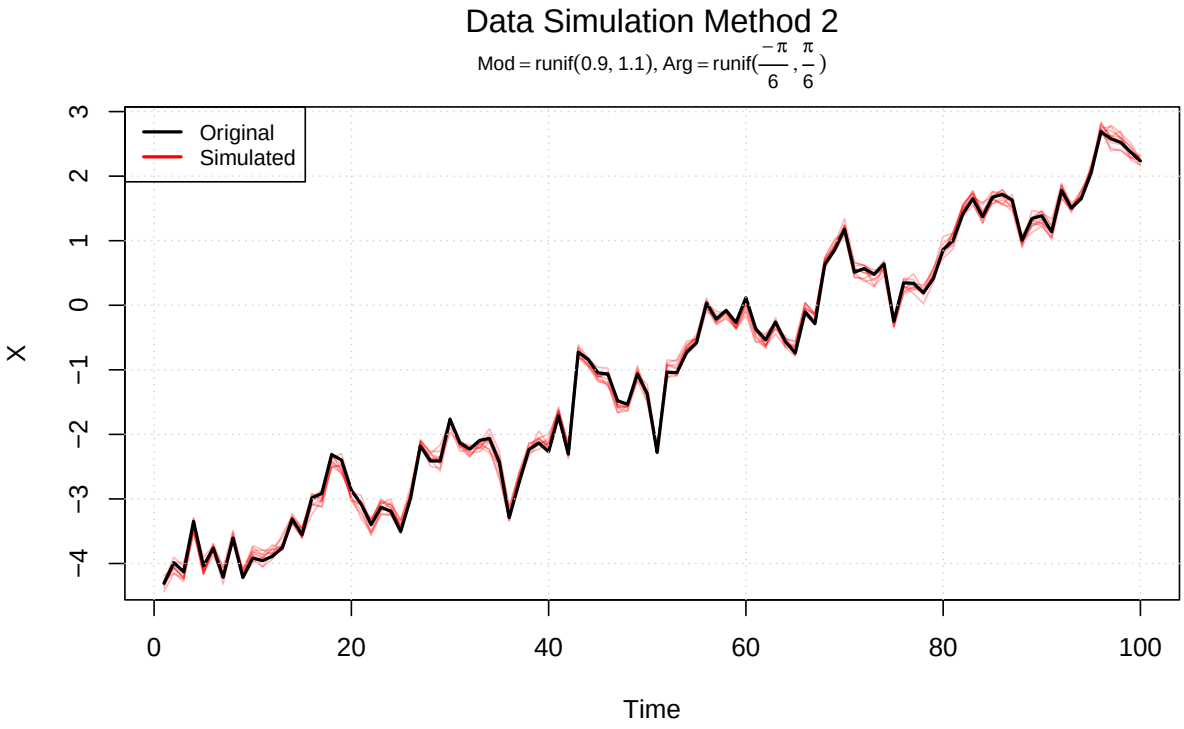
```
Mt = estimator(x, method = 'Mt')      ## Estimating trend component
Y = x - Mt                             ## De-trending
Z = Y - mean(Y)                        ## Removing the mean
Z_f = fft(Z, inverse = FALSE)          ## Converting to frequency domain
inputs = c()                           ## Initializing vector to store values

Mod_a = 0.95; Mod_b = 1.05             ## Defining Modulus and Argument values
Arg_a = -pi/6; Arg_b = pi/6

for (i in 1:10){

  ## Creating small perturbation
  Z_p = Z_f * complex(modulus = runif(N, Mod_a, Mod_b), argument = runif(N, Arg_a, Arg_b))

  Z_t = as.numeric(fft(Z_p, inverse = TRUE)) / N  ## Converting back to time domain
  x_p = Z_t + mean(Y) + Mt                        ## Adding mean and trend back
  inputs = c(inputs, x_p)                         ## Saving the new series
}
```

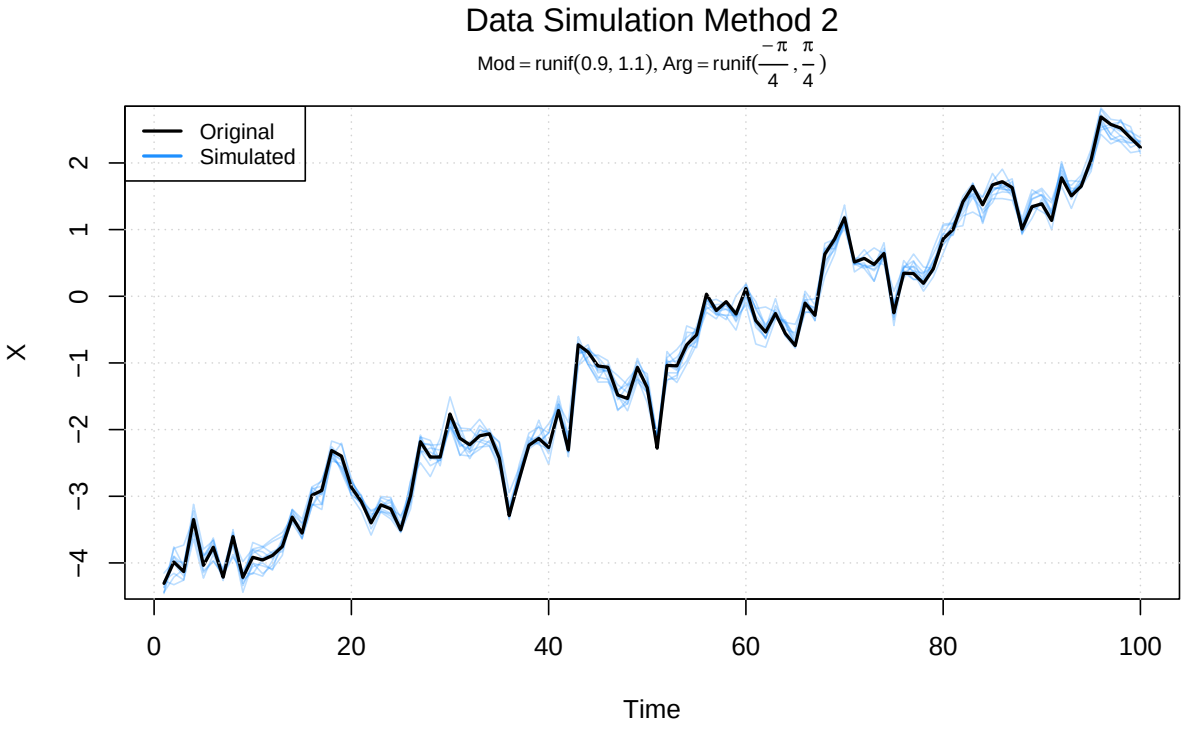


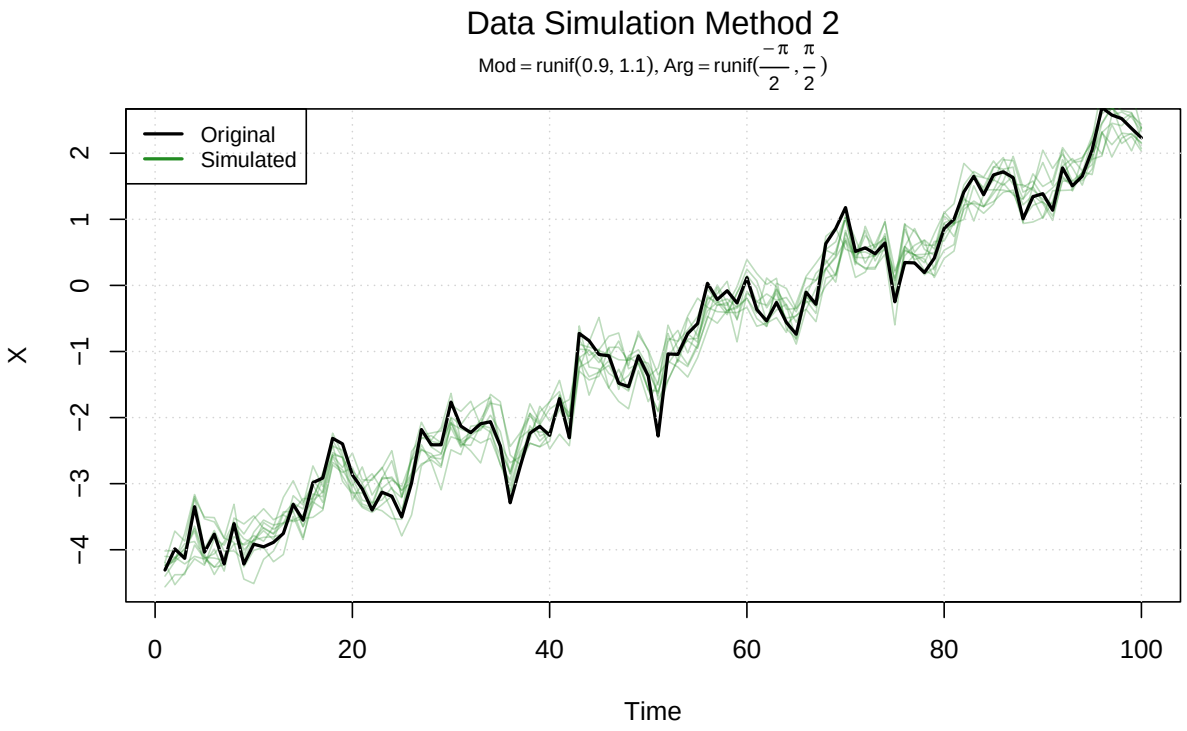
Method 2 w/ different Mod and Arg values:






Data Simulation Method 2
Mod = runif(0.9, 1.1), Arg = runif(−π/6, π/6)
Original
Simulated
X
Time
0
20
40
60
80
100
−4
−3
−2
−1
0
1
2
3

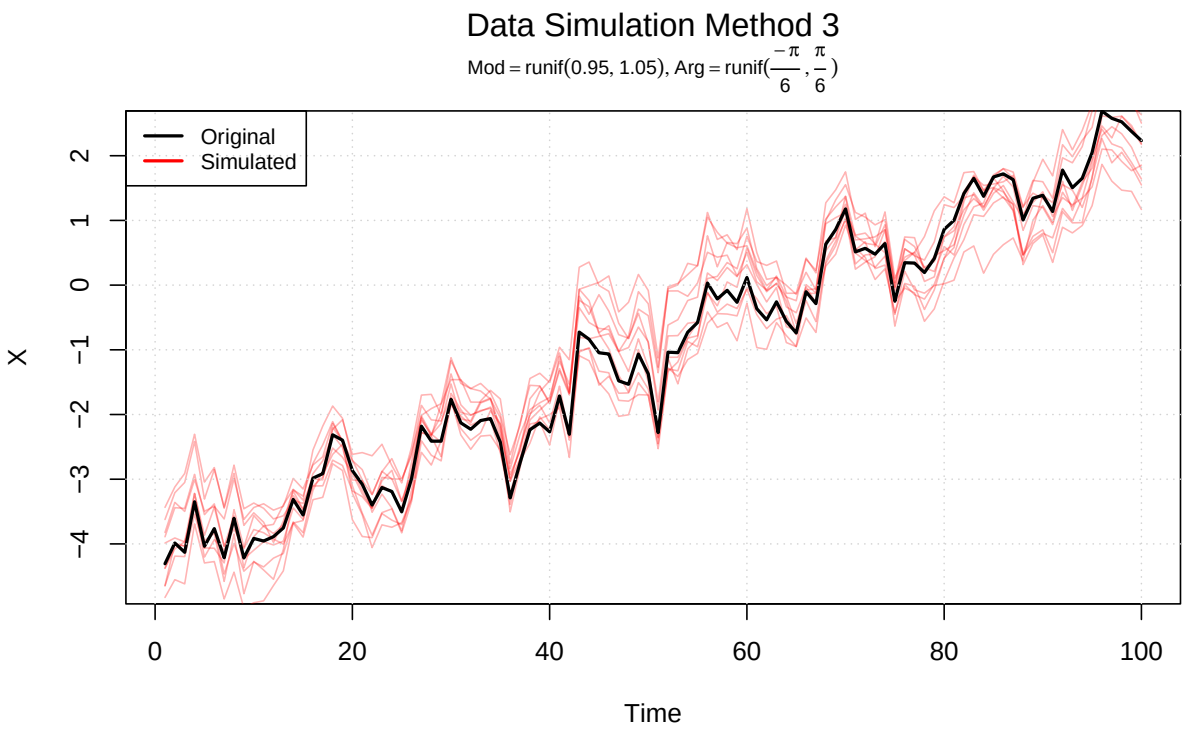

Data Simulation Method 2
Mod = runif(0.9, 1.1), Arg = runif(−π/4, π/4)
Original
Simulated
X
Time
0
20
40
60
80
100
−4
−3
−2
−1
0
1
2


Data Simulation Method 2
Mod = runif(0.9, 1.1), Arg = runif(−π/2, π/2)
Original
Simulated
X
Time
0
20
40
60
80
100
−4
−3
−2
−1
0
1
2

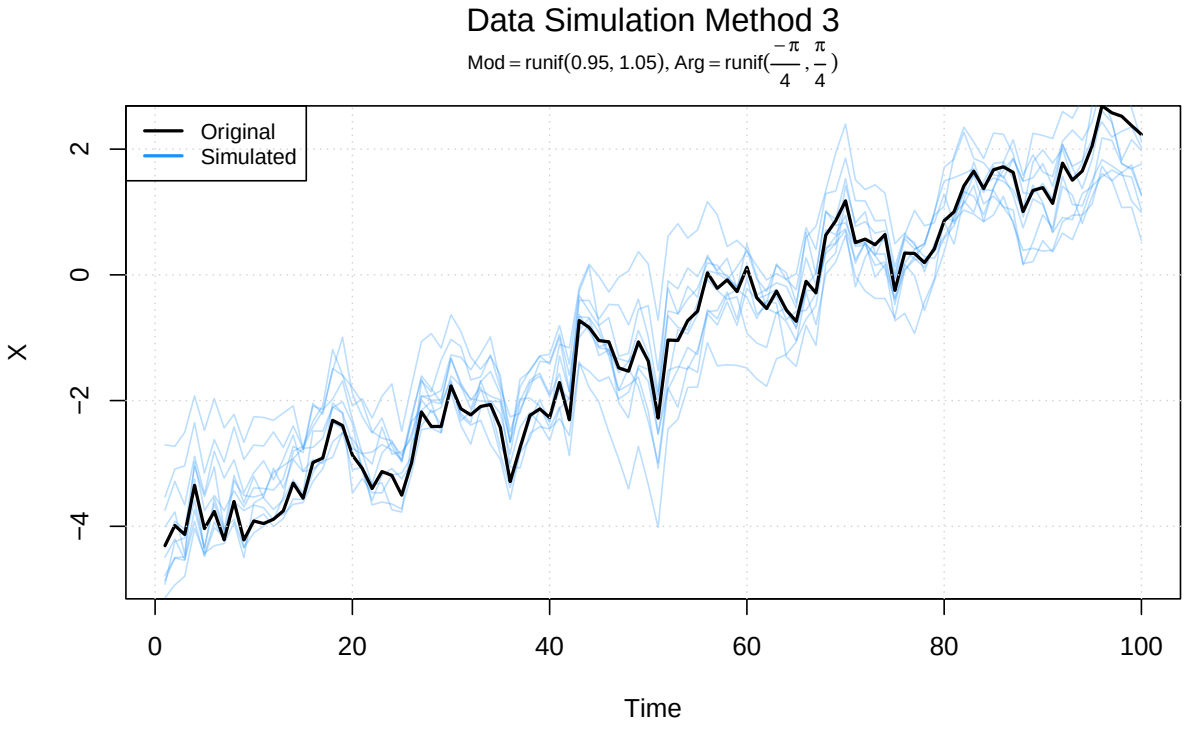
Method 3:

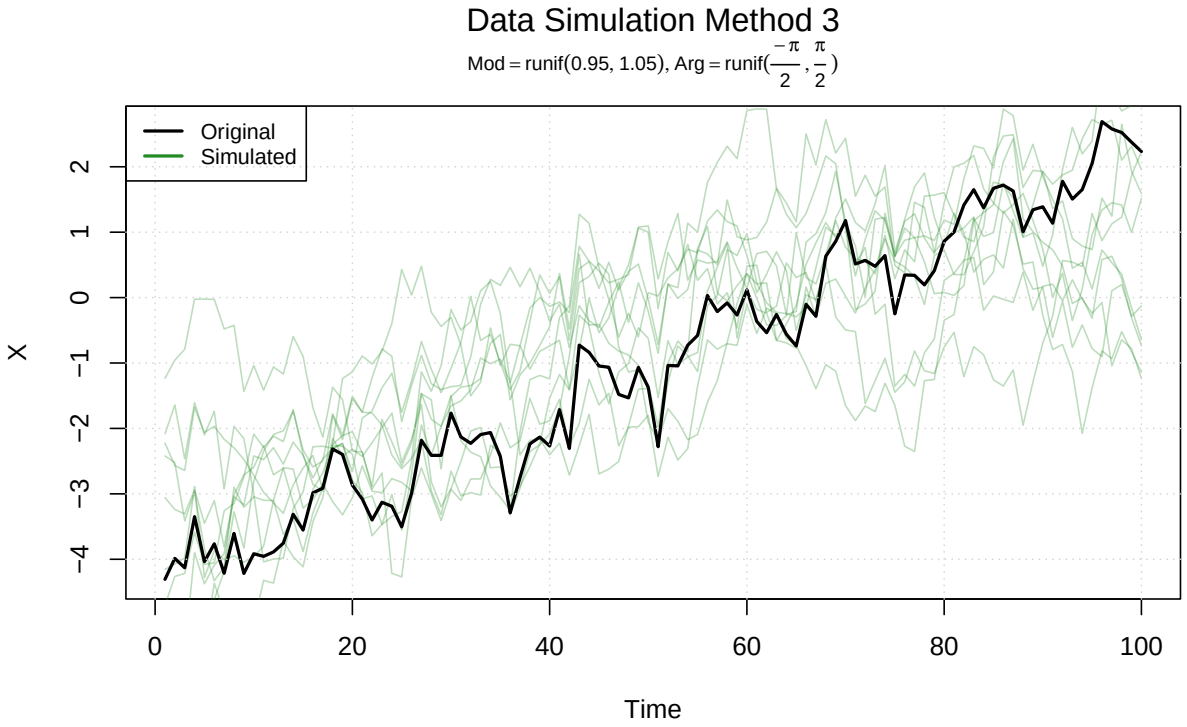
0. Input a complete time series.
1. Estimate the trend and periodic component(s).
2. Compute the residual noise between the estimate series and the input series.
3. Perturb the noise series (using multiplication of complex vector).
4. Perturb the estimate series (using multiplication of complex vector).
5. Add the perturbed noise back to the perturbed estimate series.

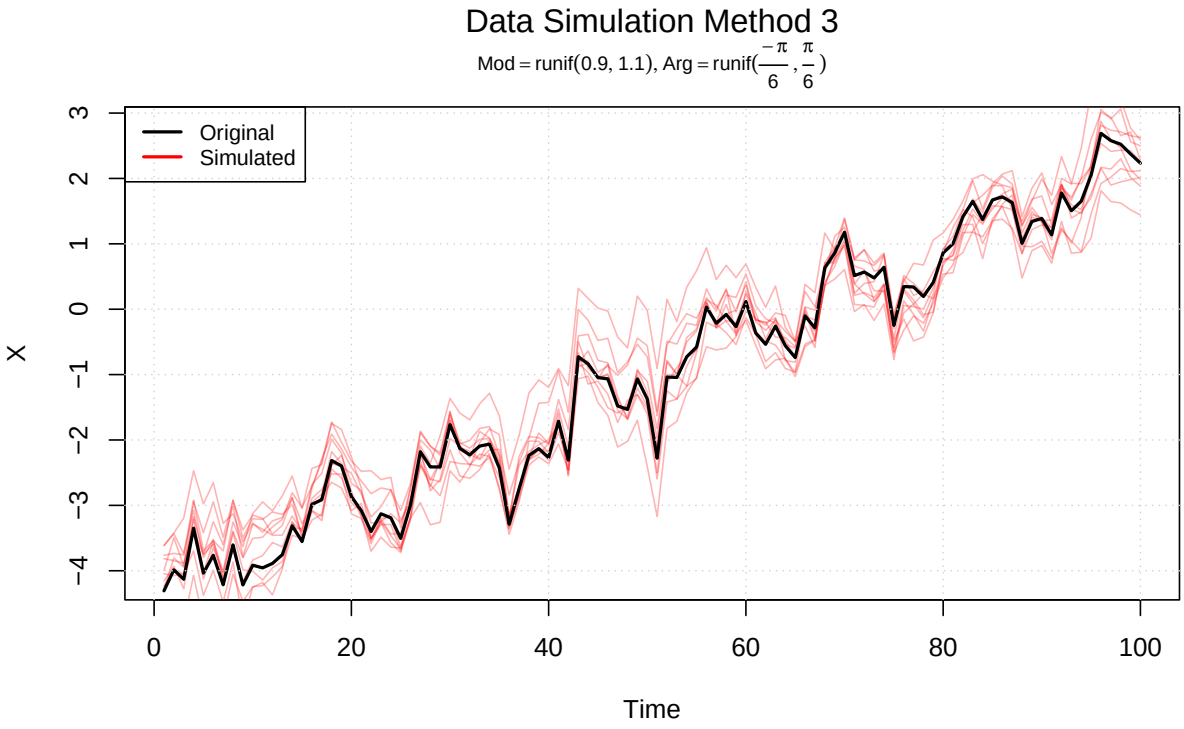
```
xE = estimator(x, method = 'Xt')       ## Estimate trend + periodic and noise components
w = x - xE                             ## Computing the residual noise
w_f = fft(w, inverse = FALSE)          ## Converting noise to frequency domain
xE_f = fft(xE, inverse = FALSE)        ## Converting estimate to frequency domain
inputs = c()                           ## Initializing vector to store values

Mod_a = 0.95; Mod_b = 1.05             ## Defining Modulus and Argument values
Arg_a = -pi/6; Arg_b = pi/6

for (i in 1:10){

  ## Creating small perturbation
  w_p = w_f * complex(modulus = runif(N, Mod_a, Mod_b), argument = runif(N, Arg_a, Arg_b))
  xE_p = xE_f * complex(modulus = runif(N, Mod_a, Mod_b), argument = runif(N, Arg_a, Arg_b))

  w_t = as.numeric(fft(w_p, inverse = TRUE)) / N    ## Converting back to time domain
  xE_t = as.numeric(fft(xE_p, inverse = TRUE)) / N

  x_p = xE_t + w_t                                  ## Adding perturbed noise back
  inputs = c(inputs, x_p)                           ## Saving the new series
}
```

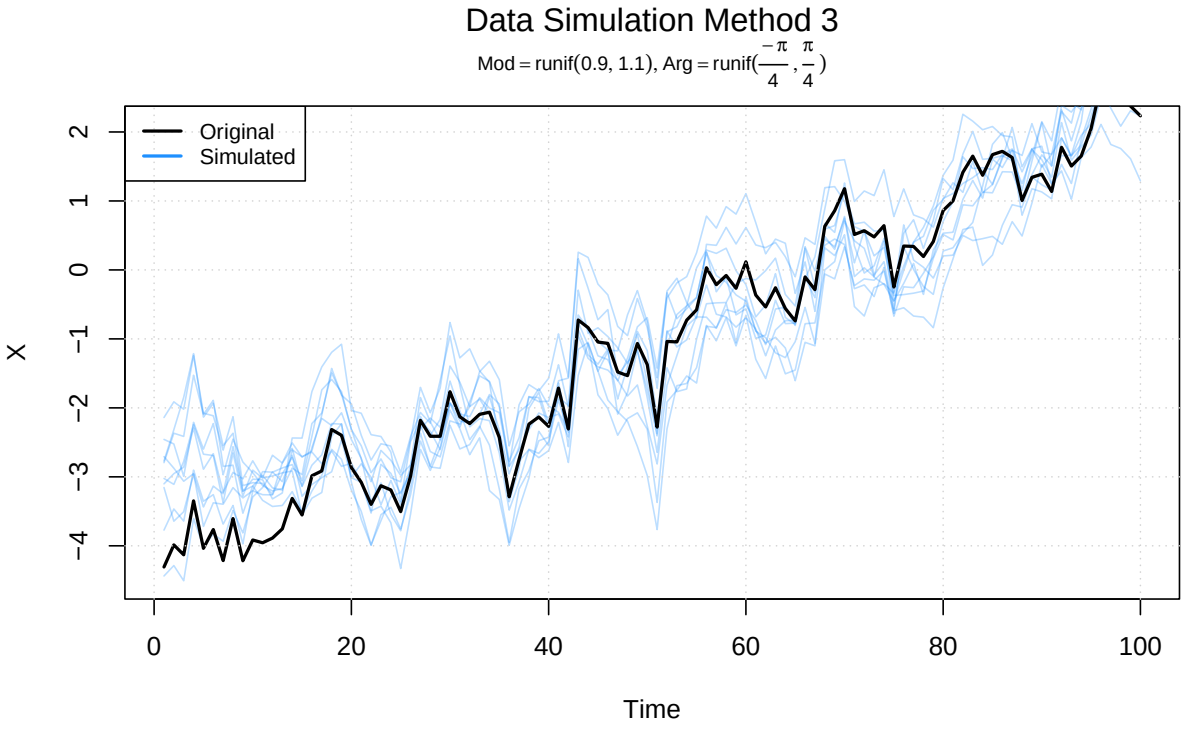

Data Simulation Method 3
Mod = runif(0.95, 1.05), Arg = runif(−π/6, π/6)
Original
Simulated
X
Time
2
1
0
−1
−2
−3
−4
0
20
40
60
80
100

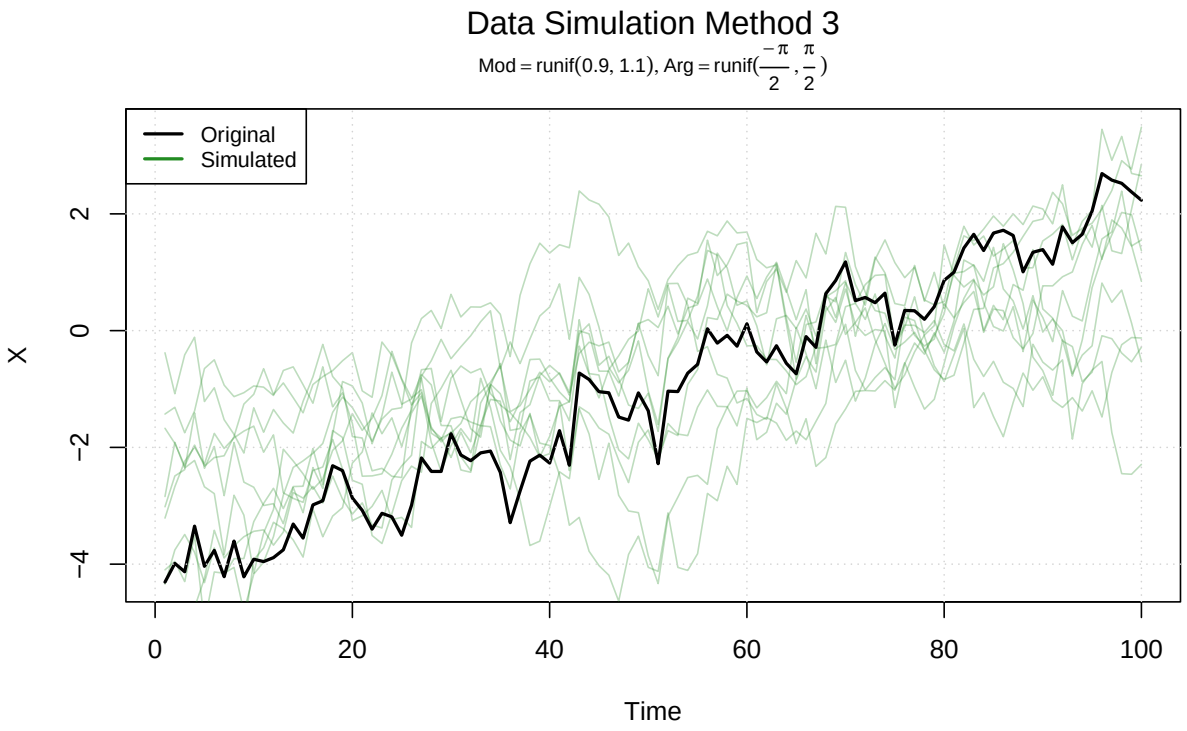
Method 3 w/ different Mod and Arg values:






Data Simulation Method 3
Mod = runif(0.9, 1.1), Arg = runif(−π/6, π/6)
Original
Simulated
X
Time
0
20
40
60
80
100
−4
−3
−2
−1
0
1
2
3


Data Simulation Method 3
Mod = runif(0.9, 1.1), Arg = runif(−π/4, π/4)
Original
Simulated
X
Time
0
20
40
60
80
100
−4
−3
−2
−1
0
1
2


Data Simulation Method 3
Mod = runif(0.9, 1.1), Arg = runif(−π/2, π/2)
Original
Simulated
X
2
0
−2
−4
0
20
40
60
80
100
Time