Neural Network Imputer Demonstration

Algorithm:

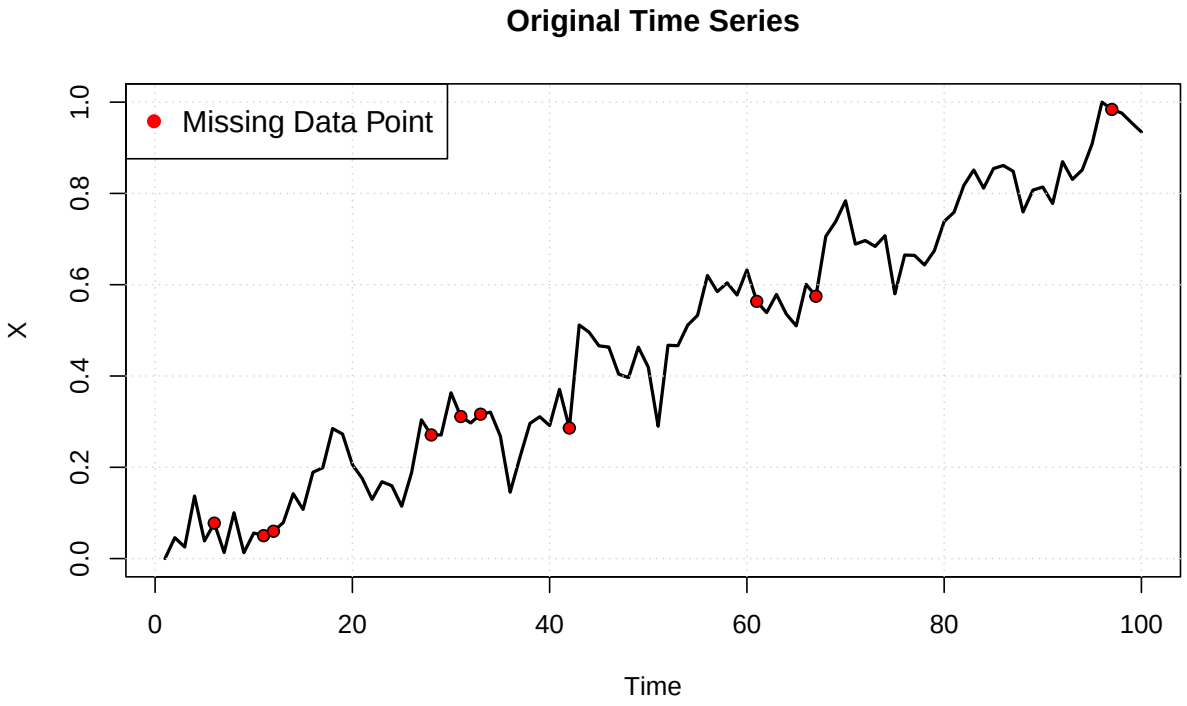
1. Interpolate missing values in the original time series using linear interpolation.
2. a. Estimate the trend component of the input time series
   b. Estimate the periodic component(s) of the input time series
3. Simulate a sufficient number of time series with the same trend and periodic structure by applying small perturbations to the residual noise in the frequency domain representation (will become neural network targets).
4. Impose a gap structure on the simulated time series (i.e., the same gap structure as observed in the original time series) and produce K unique 'gappy' time series (will become neural network inputs).
5. Construct, compile, and fit a neural network 'autoencoder' model using the newly created input and target time series. Using the autoencoder, predict on the original time series.
6. Extract the predicted values of the missing data points to complete the original time series.
7. Repeat steps 2-6 using the output of Step 6 as the input for Step 2, each time storing the predictions.
8. Return the final predictions as the average prediction over the total number of specified iterations.

Setup Steps:

The Neural Network Imputer relies on these four libraries specifically:

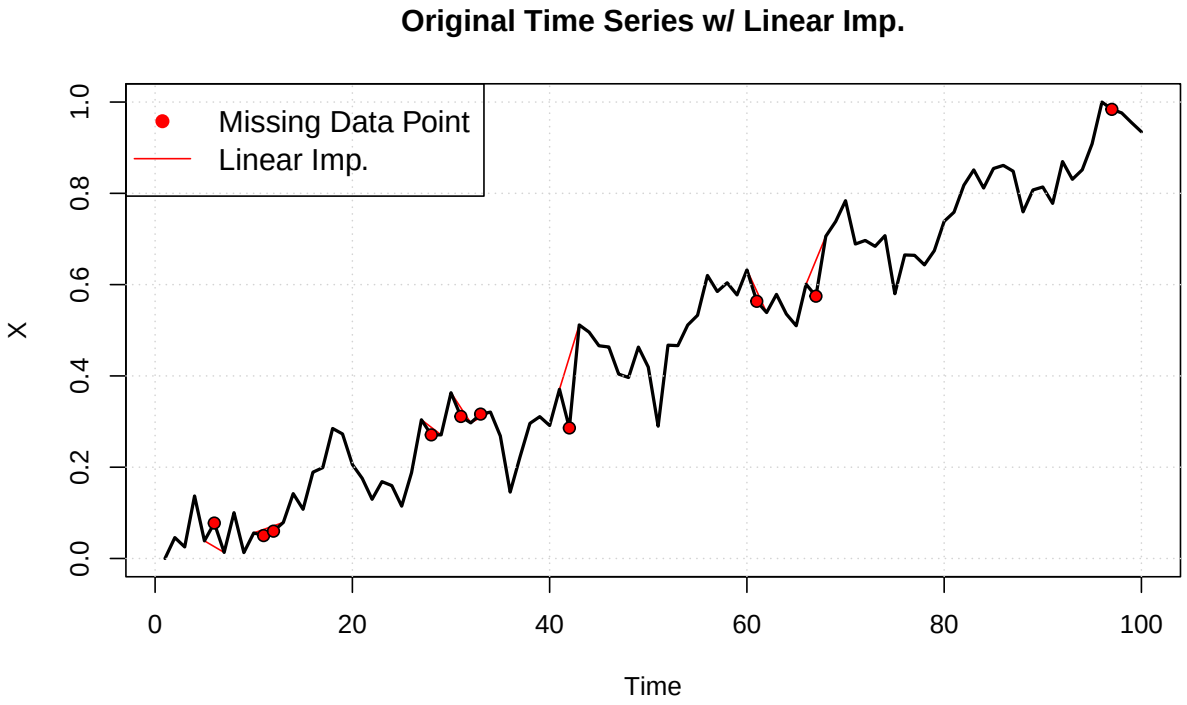
```
library(tsinterp)
library(interpTools)
library(tensorflow)
library(keras)
```

We can simulate a time series and go through the algorithm to visualize what is happening in each step of the process.



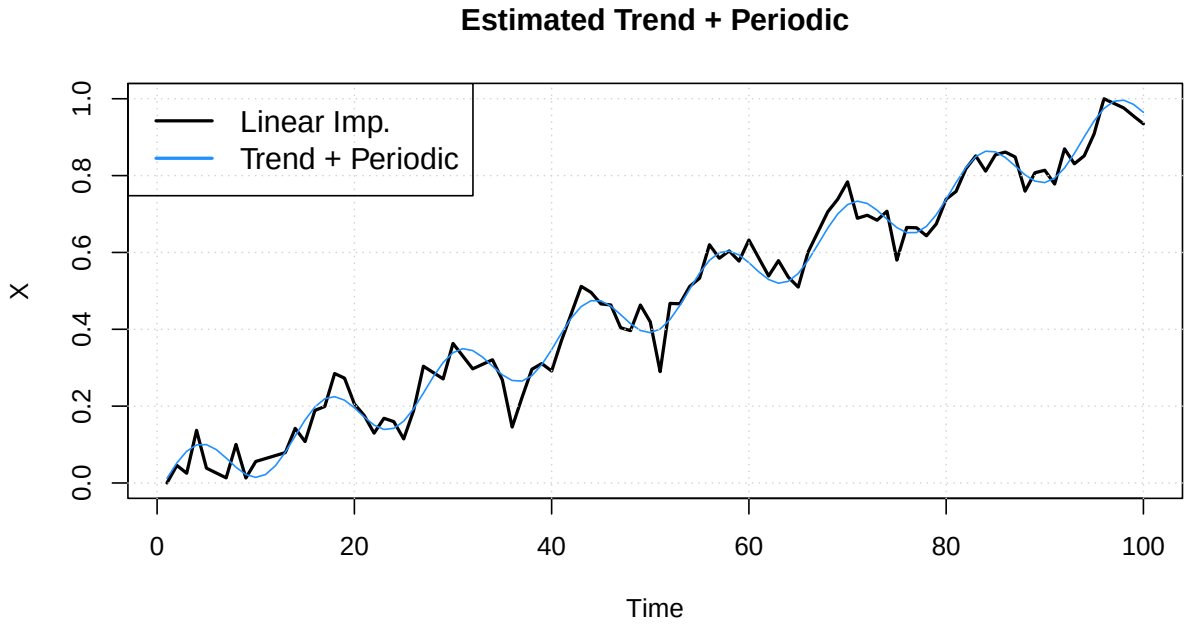
Initialization Step:

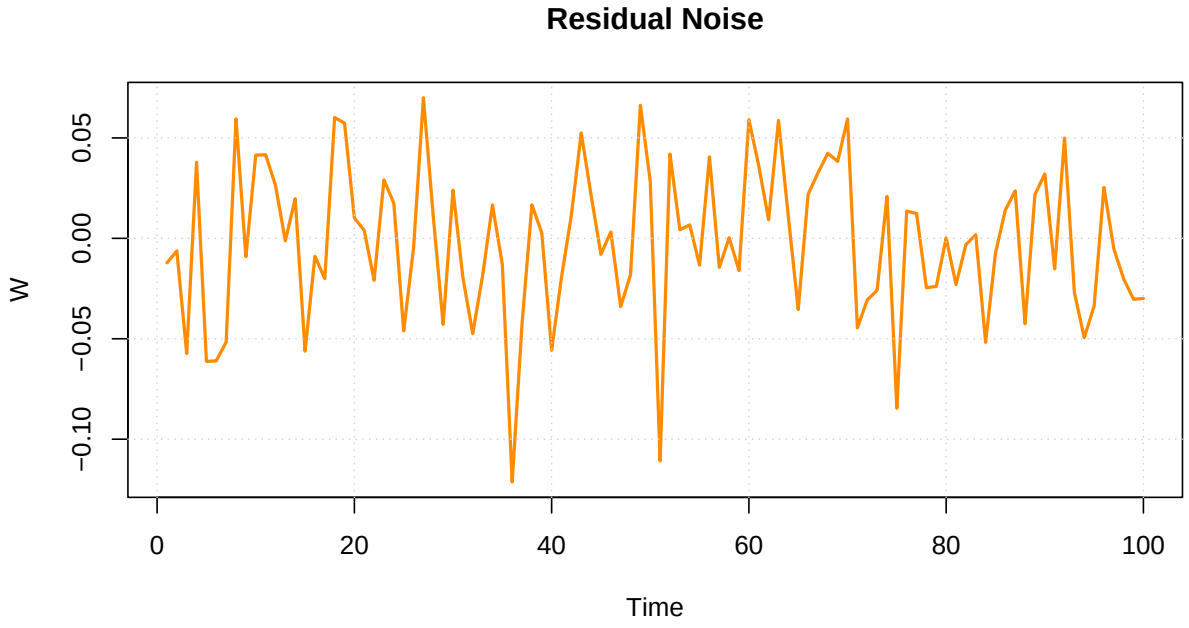
The first step of the algorithm is to use linear imputation to fill in the gaps for the original input time series.



Trend and Periodic Estimation:

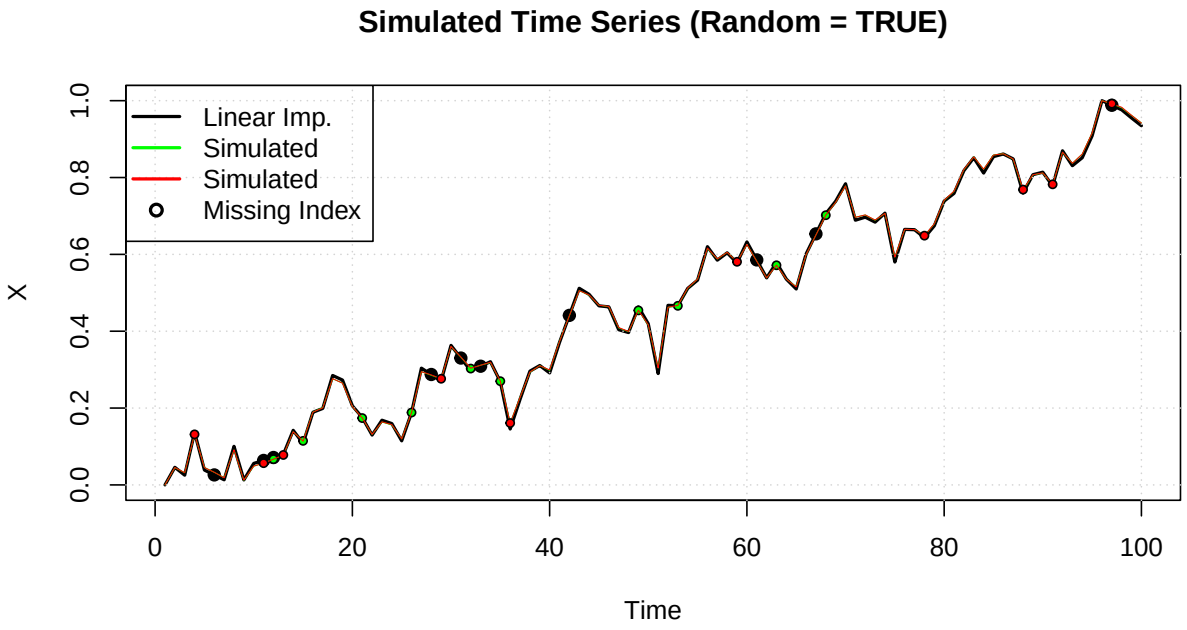
The next step of the algorithm is to use functions from the TSinterp package to estimate the trend and periodic components of the input time series. This step returns the trend + periodic series, as well as the residual noise.

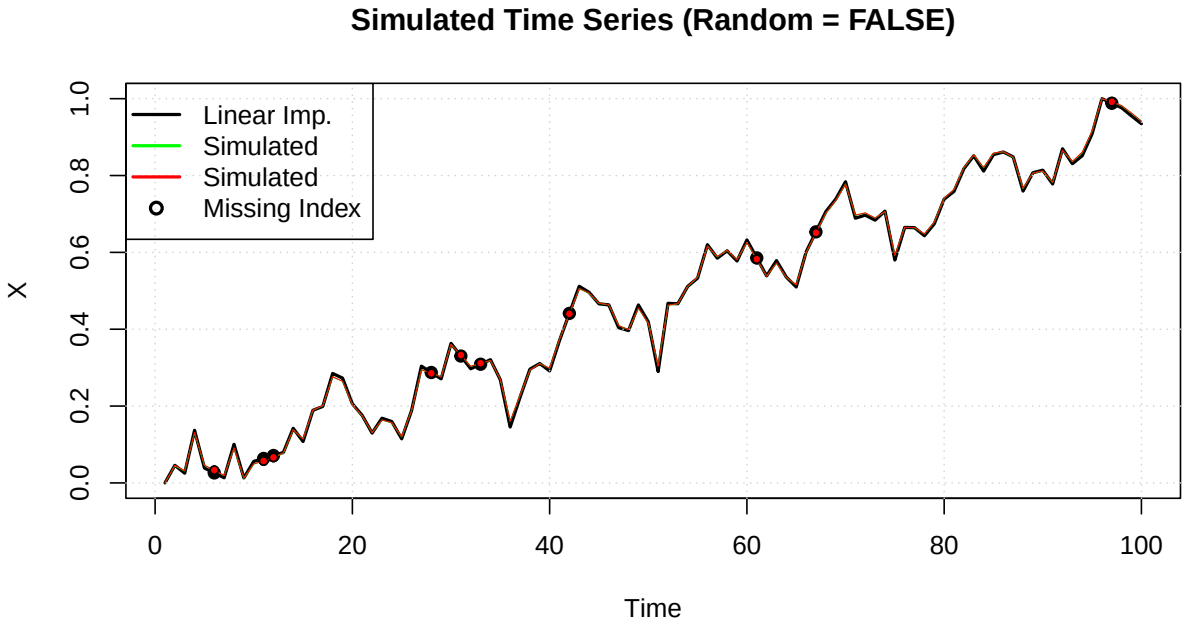




Time Series Simulation:

The next steps are to generate an appropriate number of time series with the same trend and periodic structure by applying small perturbations to the residual noise in the frequency domain and then inverting it back into the time domain. Next, we impose a gap structure similar to that of the original input time series. The gap structure is the same, but there is a parameter which indicates whether or not the placement of the gaps is randomized.

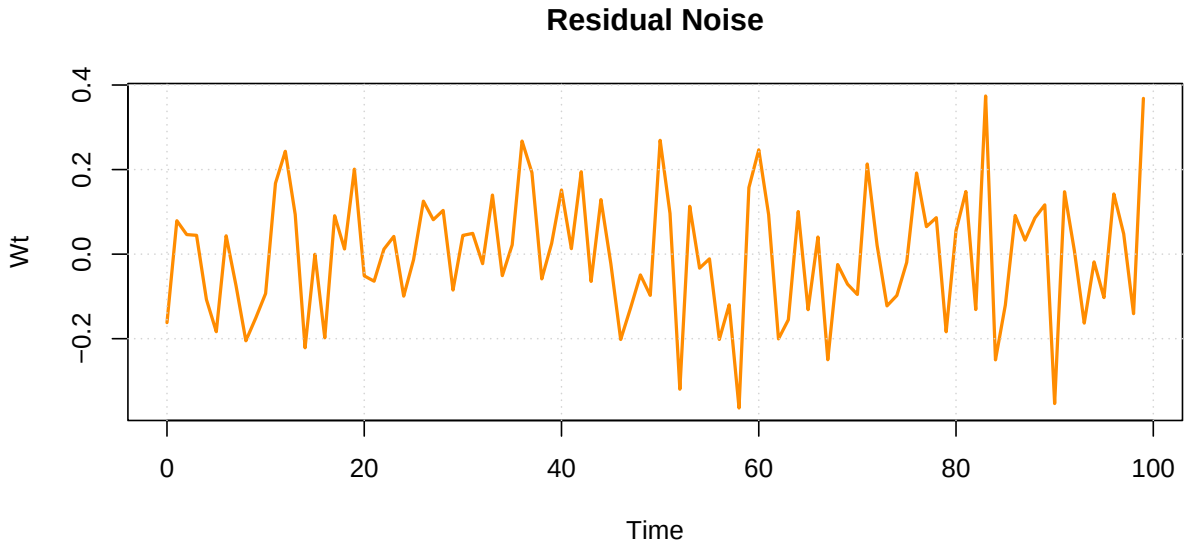


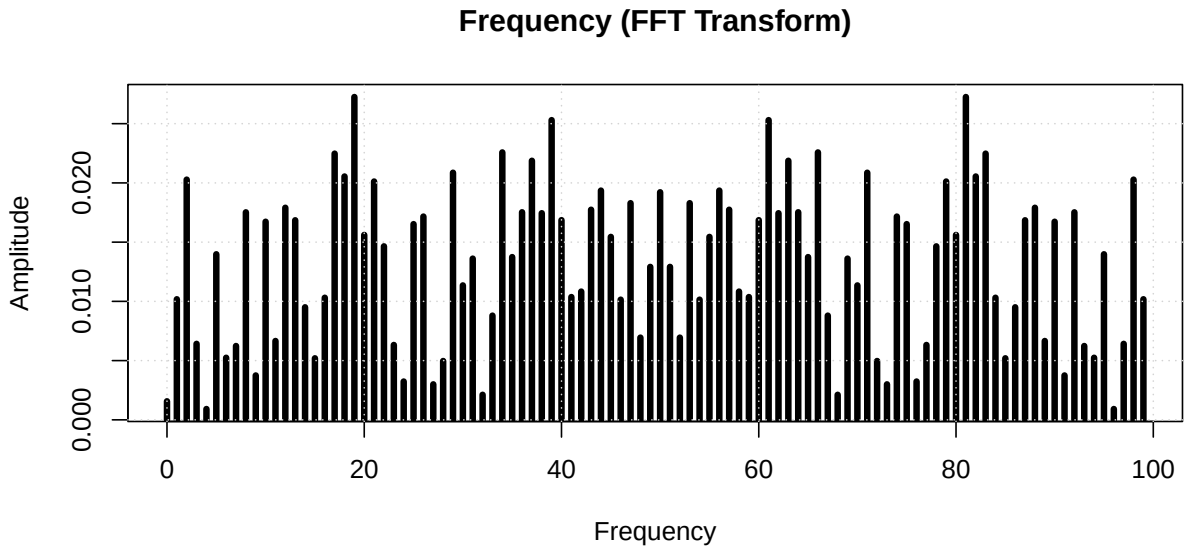


Noise Perturbation Process:

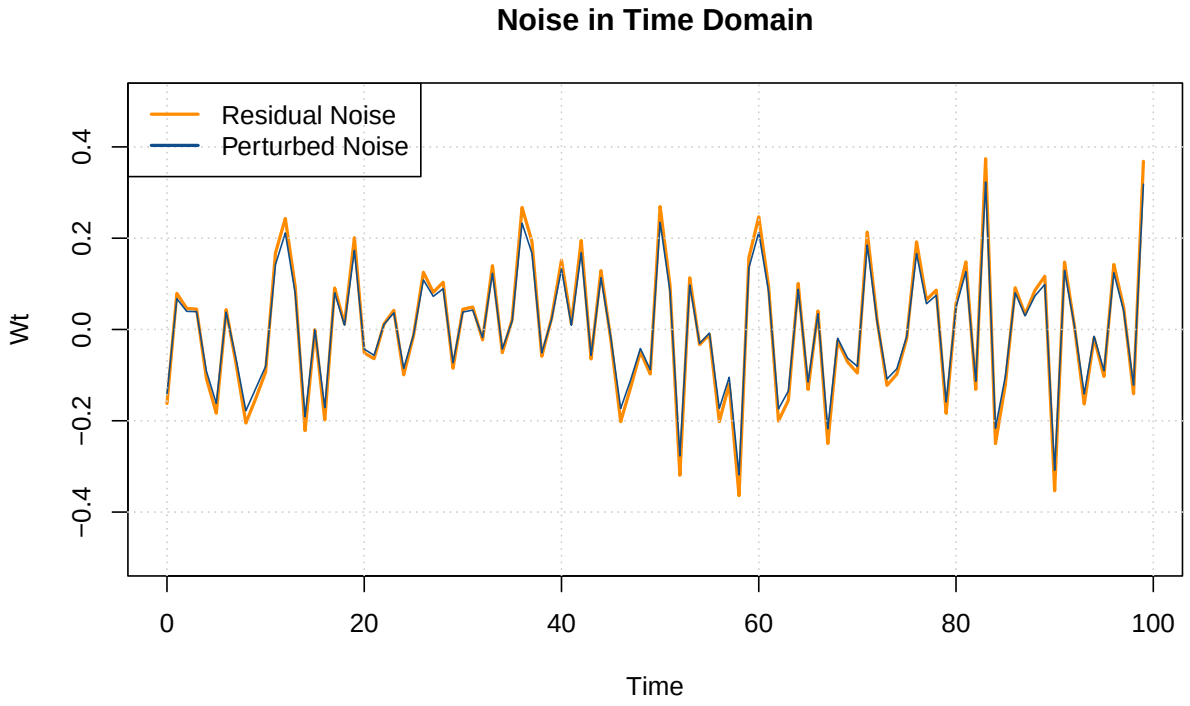
1. Subtract the estimated trend + periodic series from the linear imputed series to form the residual noise series.
2. Convert the residual noise series to the frequency domain using the fft() function.
3. The output of the fft() function is a vector of complex numbers with length N. To perform the perturbation, generate another vector of length N with complex values of amplitude 1 and phase somewhere between $(0, 2\pi)$.
4. Multiply the two complex vectors. This applies a small perturbation to the original residual noise in the frequency domain.
5. Convert the output of Step 4 back into the time domain and add it to the trend + periodic series.
6. This has created one time series which will be an element of the neural network training data. Repeat this process enough times as to generate a sufficient set of training data.

```r
## Generate a time series
Xt = simXt(N = 100, mu = 0, numTrend = 0, numFreq = 2)$Xt

## Estimate trend + periodic and noise components
estimate = estimator_demo(Xt)
Mt = estimate[[1]]; Tt = estimate[[2]]; Wt = Xt - Mt - Tt;

## Converting the residual noise series to the frequency domain
Wt_freq = fft(Wt)
```





```r
## Generating new vector and multiplying the two complex vectors
Wt_freq_p = Wt_freq * complex(modulus = runif(100, 0.98, 1.02), argument = runif(100, -pi/6, -pi/6))

## Converting perturbation back into time domain
Wt_p = fft(Wt_freq_p, inverse = TRUE) / length(Wt)
```



```
## Adding the perturbed noise to the trend and periodic estimations
Xt_p = Mt + Tt + Wt_p
```