Fast Bias-Removed Thêo1 Calculation with R

W.J. Riley
Hamilton Technical Services
Beaufort, SC 29907 USA
Rev. A June 28, 2020

• Introduction

This document describes a fast bias-removed Thêo1 calculation (ThêoBR) for the R statistical computing environment [1]. Some information about using R for frequency stability analysis will be found in Reference [2]. Thêo1 is an advanced statistic for analyzing frequency stability that provides results out to 75% of the phase data record length [3]. Some background about the evolution of the Thêo1, ThêoBR, and ThêoH statistics is given in Appendix 1. The ThêoBR calculation is implemented in C++ and is called from R using the Rcpp package [4]. It uses the fast Lewis recursive Thêo1 algorithm [5] along with the Taylor and Howe fast ThêoBR method [6]. Data sets of tens of thousands of points require only a few seconds rather than many minutes with the naive method. The calculation includes determination of one or two-sided error bars at a selected confidence factor. Tabulation and plotting of the results can be done within R, or an R driver function can be used for that purpose.

• ThêoBR Function

The ThêoBR function is defined as follows, and its C++ code is shown in Appendix 2.

```
double TheoBR(NumericVector x, NumericVector t, NumericVector u,
  NumericVector l, NumericVector tau, double tau0=1.0, double cf=0.683,
  int br=1)

Arguments:
  x    = NumericVector = Phase data input
  t    = NumericVector = Theo1BR output
  u    = NumericVector = Upper Theo1BR error bar
  l    = NumericVector = Lower Theo1BR error bar
  tau  = NumericVector = Tau output
  tau0 = double = Data sampling time input (default=1)
  cf   = double = Confidence factor
                  Negative = Single-Sided (default=0.683)
  br   = integer = Bias remove flag (0=n0, 1=yes, default=1)

Return = double = Theo1 Bias Factor
         or 1.0 if no bias correction
         or 0 if error
```

The five `NumericVectors` are Rcpp data types comprise the input and output arrays, and they must be created in R before calling the `TheoBR()` function. The N point `x` array is the phase data, and the (N-1)/2 point `t`, `u` and `l` arrays are respectively the raw Thêo1 or ThêoBR, upper and lower error bar, and tau results. The `cf` parameter sets the confidence factor (negative for single-sided error bars) and the `br` flag determines whether the t output array is raw Thêo1 or bias-corrected ThêoBR. The phase data array is unchanged.

• ThêoBR Calculation

The `TheoBR()` function (see Appendix 2) begins with an "all tau" Thêo1 calculation using the Lewis method (pasted directly from his Reference [5] paper except that the linear slope removal is moved inline), along with the corresponding tau values. While raw Thêo1 values are not required at every tau for the fast ThêoBR calculation, they are necessary for the fast recursive algorithm, and they can be made available without bias correction by setting the `br` argument to 0. The function then uses the Taylor and Howe fast ThêoBR method to determine the bias correction factor. That is applied to the raw Thêo1 values, and is also used to find the dominant integer power law noise type. Next, the noise type and number of data points is used to determine the edf, the number of equivalent chi-squared degrees of freedom that apply. Finally, the edf and confidence factor are used to calculate the applicable chi-square values and calculate the upper and lower error bars (the latter set equal to the nominal value for the single-ended case). The function ends by returning the correction factor.

• Overall Analysis

The overall ThêoBR analysis is performed in R, and a typical procedure is as follows (R console commands are shown in red – beware of using the wrong quote character):

```
To set up
  Install Rcpp with:
    > library(Rcpp)
  Compile with:
    > sourceCpp("TheoBR.cpp")
  after adding full path to filename e.g., "C:\\R\\Theo1BR.cpp"

To use
  Load phase data into vector x (the tau0 must be known)
  For example:
    > x<-scan("C:\\Data\\phase.dat")
  Determine the # of phase data points with > N=length(x)
  Allocate results vector t with:
    > t<-numeric((N-1)/2)
  Allocate u vector tau with:
    > u<-numeric((N-1)/2)
  Allocate l vector tau with:
    > l<-numeric((N-1)/2)
  Allocate tau vector tau with:
    > tau<-numeric((N-1)/2)

Call with
> Theo1BR(x, t, u, l, tau, tau0, cf, br)
where tau0, cf and br have defaults of 1.0, 0.683 and 1 respectively
(either set their values or use the defaults)

Display results
  > t

or create a data frame
  > r<-data.frame(tau,l,t,u)
```

```
To tabulate the tau, minimum, nominal and maximum Theo1, and display it with:
  > r

or plot the results with (for example, red line, no error bars):
  > plot(tau,t,type="l",main="Theo1BR Plot",xlab="Tau",ylab="Theo1BR",
  + log="xy", col="red")

To add lines representing upper and lower error bounds:
  > points(tau, u, type="l", col="blue")
  > points(tau, l, type="l", col="blue")

To show the error bounds as a grey area on the plot:
  > plot(tau,t,type="l",main="Theo1BR Plot",xlab="Tau",ylab="Theo1BR",
  + log="xy", col="red")
  > polygon(c(tau, rev(tau)), c(u, rev(l)), col = "grey70", border = NA)
  > points(tau,t,type="l",col="red")
```

Other R plot formats can be used as desired. An R driver function can encapsulate the complete analysis process (see Appendix 4).

• Examples

Note: These examples assume that the basic setup and array allocation has been done. If the results arrays are larger than required, the extra values should be ignored.

1. Check a basic Thêo1 calculation using test values from Reference [2].
   - Enter test values into x array.

   ```
   > ttest<-c(1.00,2.50,0.65,-3.71,-3.30,1.08,0.50,2.20,4.68,3.29)
   ```
   - Call `ThêoBR()` with `br=0`.

   ```
   > TheoBR(ttest,t,u,l,tau,1.0,0.683,0)
   ```
   - Examine the results in t array. The four raw Thêo1 results are for even averaging factors m from 2 to 8. The latter agrees with the m=8 value in the paper.

   ```
   > t
   [1] 2.055700408 1.509405466 1.412349249 1.148758425
   ```

2. Check the results of a `ThêoBR()` calculation using the 1000-point test suite of Reference [6] after conversion to 1001 points of phase data as compared with Stable32.

• Conclusions

The naïve Thêo1 algorithm that closely follows its definition is fine for determinations at a single tau or over a tau range for data set sizes up to 10,000 points or so. For example, an "All Tau" Thêo1 analysis for that number of phase data points takes about 20 seconds. But for 100,000 data points, the calculation takes about 30 minutes. Similar times apply to ThêoBR and ThêoH. The new Lewis fast Thêo1 algorithm changes that, making Thêo1 runs practical for very large data sets, as demonstrated here. The R/RStudio statistical computing environment combined with C++ via Rtools/Rcpp allows the Lewis Thêo1 algorithm to be implemented in a convenient way.

• References

1. The R Project for Statistical Computing.
2. W.J. Riley, "Frequency Stability Analysis Using R," Hamilton Technical Services web site, June 2020. This paper has information about using R, Rcpp, and Rtools.
3. D.A. Howe and T.K. Peppler, "Very Long Term Frequency Stability: Estimation Using a Special Purpose Statistic," *Proceedings of the 2003 IEEE International Frequency Control Symposium and PDA Exhibition Jointly with the 17th European Frequency and Time Forum*, May 2003, pp. 233-238. This paper introduced the Thêo1 statistic.
4. The easiest way to use C/C++ code in R is with the Rcpp and Rtools tool chain. Rcpp supports calling C++ from R, while Rtools compiles C++ code under R. Rcpp.
5. B. Lewis, "Fast Algorithm for Calculation of Theo1," Submitted *to IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, May 2020. This paper presents a fast, recursive algorithm for "all tau" Thêo1 calculations.
6. J. A. Taylor and D.A. Howe, "Fast TheoBR: A Method for Long Data Set Stability Analysis," *IEEE Transactions on Ultrasonics, Ferroelectrics, Frequency Control*, Vol. 57, No. 9. pp. 2091-2094, September 2010. This paper describes the "final" method for determining the Thêo1 bias factor.
7. D. A. Howe, J. McGee-Taylor, and T. Tassett, "TheoH Bias-Removal Method," in 2006 IEEE International Frequency Control Symposium Exposition, June. 2006, pp. 788–792, iSSN: 2327-1914. This paper describes the original Thêo1 bias factor determination.
8. D.A. Howe, "ThêoH: A Hybrid, High-Confidence Statistic that Improves on the Allan Deviation,", *Metrologia*, Vol. 43, pp. S322-S331, 2006. This paper effectively summarizes the work on devising the Thêo1, ThêoBR and ThêoH statistics.
9. D.A. Howe and T.N Tasset, "Thêo1: Characterization of Very long-Term Frequency Stability," *Proceedings of the 2004 European Frequency and Time Forum*. Thêo1 introduction in Europe,
10. W.J. Riley, *Handbook of Frequency Stability Analysis*, NIST Special Publication 1065, July 2008, Sections 5.2.15 – 5.2.16.
11. T.N Tasset, D.A. Howe, and D.B. Percival, "Thêo1 Confidence Intervals," *Proceedings of the 2004 IEEE International Ultrasonics, Ferroelectrics, and Frequency Control Joint 50th Anniversary Conference*, pp. 725-728.
12. J.A. McGee and D.A. Howe, "ThêoH and Allan Deviation as Power-Law Noise Estimators," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, Vol. 54, No. 2, February 2007, pp. 448-452.
13. W.J. Riley, "A Test Suite for the Calculation of Time Domain Frequency Stability", *Proceedings of the 1995 IEEE International Frequency Control Symposium*, June 1995, pp. 360-366.

Appendix 1

Notes for Thêo1, ThêoBR, and ThêoH

The Thêo1 statistic was formally introduced in the May 2003 Howe and Peppler paper of Reference [3], and by that time had been the subject of "discussions, interpretations and tests", including implementation into the Stable32 program. That code was created and running in early February 2003 and was revised to the paper's "final" version by the end of April of that year. The 2003 Howe and Peppler paper described Thêo1 as applying tor even averaging factors, m, from 10 to N-1 (where N is the number of phase data points – the Lewis fast Thêo1 algorithm covers m from 2). It discussed its stride of 0.75*m*tau0 where tau0 is the data sampling interval (and thus provides results out to 75% of the record length), compared its frequency response to that of the Allan variance, AVAR, gave simple estimates for its bias versus AVAR for various power law noise processes which ranged from factors between 0.77 to 1.50 for ADEV, and gave empirical formulae for its equivalent number of Chi-squared degrees of freedom (edf). Reference [9] followed with more details in 2004. Note that the Thêo1 name generally applies to the variance, while most usage is for its square root, the Thêo1 deviation, and it is important to keep that distinction in mind when discussing it.

Thêo1 is a biased estimator of the Allan variance for other than white FM noise, and, if the noise type is known, one way to determine the necessary correction factor is to use an empirical formula. References [3] and [7] included a preliminary version of that, and a later improved version is given in Reference [10], which also supported Thêo1 calculations down to m=2. The latter appears in the Stable32 as the Theo1Bias() function, and that Thêo1 bias expression also shown on page 30 of the *Handbook of Frequency Stability Analysis* [10].

Work continued on Thêo1, particularly regarding ways to determine its bias without knowledge of the power law noise type (or mixtures). The best approach was to simply calculate the average ratio of Thêo1 to AVAR over a range of averaging factors. That is the method used for bias-removed ThêoBR herein. The final approach for that is described in Reference [6].

However it is still necessary to know the dominant power law noise type in order to determine the edf and to set Chi-squared confidence limits for a Thêo1/BR result. The Thêo1/BR bias factor can be used to determine the noise type, and a function for that purpose was incorporated into Stable32 in December 2003 based on the T.N. Tasset & D.A. Howe paper, "A Practical Thêo1 Algorithm". Then the expressions in Reference [2] can be used to determine the edf, and the desired single or double-sided error bars set accordingly. Thêo1 and ThêoH actually have narrower confidence limits as discussed in References [11] and [12].

The ThêoBR calculation in Stable32 does phase averaging to speed up the bias factor determination. That process was the subject of several e-mails between D.A. Howe and T.N. Tassset and the author between May 2003 and July 2004, and their (unpublished?) paper mentioned above. The July 2004 Stable32 code for automatically determining the ThêoBR bias correction includes the following notes:

```
/*       This bias factor is the ratio of AVAR to Thêo1           */
/*       To correct the Thêo1 deviation, multiply it by the sqrt  */
/*       of this bias factor.                                     */
/*                                                                */
/*       Thêo1 is the original biased statistic of that name.     */
/*       NewThêo1 was the original bias-corrected version of      */
/*       Thêo1, and that bias correction was found with the       */
/*       AutoTheo1Bias() function.  This bias-corrected version   */
```

```
/*       of Thêo1 is called ThêoBR (for "Thêo1 Bias Removed")     */
/*       and uses a wider range of n.  When combined with Avar,   */
/*       ThêoBR becomes ThêoH (for "Hybrid").                     */
/*                                                                */
/*       The minimum # of phase data points, N, necessary to      */
/*       find a Thêo1 bias value is determined by the overlapping */
/*       Allan variance calculation, which is carried out at a    */
/*       minimum averaging factor of m=9, and which must yield    */
/*       one analysis point whose number is given by N-2m.  Thus  */
/*       N-2m=N-2*9=N-18=1, or N(min)=19.                         */
```

The ThêoBR bias correction function was later changed to calculate the bias correction from the longest averaging factor downwards.

The ThêoH statistic (see Reference [8]) is a combination of the overlapping Allan and ThêoBR deviations whereby bias-removed Thêo1 is used to extend an ADEV analysis to longer averaging factors. The combination is used instead of a purely ThêoBR analysis mainly because the ADEV computation is faster for the shorter averaging times where it applies. The advent of the fast Lewis Thêo1 algorithm, which has a speed advantage factor on the order of N (even though its recursive nature requires an "all tau" calculation), offers the possibility of replacing ThêoH with a bias-removed Thêo1 run like that described herein. In effect, the fast ThêoBR deviation becomes a better ADEV (at least for a compete analysis run).

The reason for this discussion is to point out that, as the family of Thêo1 statistics evolved, slightly different versions were developed from the same core Thêo1 definition that use varying averaging factor ranges, bias correction methods and other details. The fast Lewis algorithm produces (very nearly) the same Thêo1 estimate but the subsequent ThêoBR bias corrections and ThêoH presentations may differ between implementations.

The R ThêoBR results are obtained essentially instantly for $N=10^3$, and nearly so for $N=10^4$, while the latter takes about 20 seconds for a Stable32 "All Tau" (really "Many Tau") Thêo1 run, thus demonstrating the benefit of the Lewis fast algorithm. An $N=10^5$ run takes a reasonable $\approx$ 12 seconds for the R ThêoBR results, while it is a rather impractical $\approx$ 30 minutes for the naive method in Stable32.

This ThêoBR C++ code from Reference [5] does not use 128-bit integers to avoid potential numerical issues for large data sets (see that paper for details).

It is worth emphasizing that there is nothing inherently wrong about using a *biased* estimator of a statistic when it offers superior properties like wider applicable range or better confidence provided that the bias can be removed. Similarly, there is no disadvantage for a more complicated estimator as long as it can be implemented effectively (e.g., the fast Lewis algorithm for Thêo1).

As an aside, the R/RStudio computing environment, while it may not be ideal for most routine frequency stability analysis, is, along with Rcpp, an attractive experimental environment for developing C++ analysis software where the interactive R user interface serves as an effective top-level means for editing, compiling, test and evaluation.

Appendix 2

C++ Code for ThêoBR with R

The following C++ code may be copied, pasted into a text editor, and saved as `TheoBR.cpp` in an appropriate folder, e.g., `C:\R\` on a Windows ® system, where it can be accessed from R.

```
/*************************************************************************/
/*                                TheoBR.cpp                             */
/*                 All Tau Bias Removed Theo1 Calculation for R          */
/*           Using Fast Lewis Theo1 and Taylor/Howe Fast TheoBR Methods  */
/*                                                                       */
/* This code calculates a bias-removed all tau version of Theo1 for R,   */
/* based on the Lewis fast Theo1 algorithm and the Taylor/Howe fast Theo1 */
/* method.  It is completely contained within this file.  The N point phase */
/* data x[], and the (N-1)/2 point Theo1 t[], upper [u] and lower [l] error */
/* bar, and Theo1 tau[] array arguments are NumericVectors.  The data     */
/* sampling interval tau0 is input as a double with a default value of 1, */
/* the confidence factor as a double with default value 0.683, and the bias */
/* removal flag as an integer (0 or 1) with defalut value 1.  The  x[] is */
/* unchanged, t[] receives the Theo1BR results, u[] and l[] receive the   */
/* upper and lower Theo1 error bar values, tau[] receives the corresponding */
/* Theo1 tau values, and the function returns a double, either the Theo1  */
/* bias factor, 0 if no correction, or an error code.  The tau values are */
/* 0.75 * tau0 * even averaging factors.  If the last br argument is 0,   */
/* the function returns 1.0 with the raw uncorrected Theo1 values.  Chi-  */
/* squared error bars are calculated for each AF based on the # of data   */
/* points, the Theo1 EDF, and the desired CF.  A negative CF entry denotes */
/* use of single-sided confidence limits.                                */
/*                                                                       */
/* To set up:                                                            */
/*   Install Rcpp with: > library(Rcpp)                                  */
/*   Compile with: > sourceCpp("Theo1BR.cpp")                            */
/*   after adding full path to filename e.g., "C:\\R\\Theo1BR.cpp"       */
/*                                                                       */
/* To use:                                                               */
/*   Load phase data into vector x (the tau0 must be known)              */
/*   e.g., > x<-scan("C:\\Data\\phase.dat")                              */
/*   Determine the # of phase data points with > N=length(x)             */
/*   Allocate results vector t with: > t<-numeric((N-1)/2)               */
/*   Allocate u vector tau with: > u<-numeric((N-1)/2)                   */
/*   Allocate l vector tau with: > l<-numeric((N-1)/2)                   */
/*   Allocate tau vector tau with: > tau<-numeric((N-1)/2)               */
/*                                                                       */
/* Call with:                                                            */
/* > Theo1BR(x, t, u, l, tau, tau0, cf, br)                              */
/* where tau0, cf and br have defaults of 1.0, 0.683 and 1 respectively   */
/*                                                                       */
/* Display results with:                                                 */
/*   > t or create a data frame                                          */
/*   > r<-data.frame(t,u,l,tau) and display it with > r                  */
```

```
/* or plot the results with (for example, red line, no error bars):          */
/*   > plot(tau,t,type="l",main="Theo1BR Plot",xlab="Tau",ylab="Theo1BR",    */
/*   + log="xy", col="red")                                                  */
/* To add lines representing upper and lower error bounds:                   */
/*   > points(tau, u, type="l", col="blue")                                  */
/*   > points(tau, l, type="l", col="blue")                                  */
/*                                                                           */
/* To show the error bounds as a grey area on the plot:                      */
/*   > plot(tau,t,type="l",main="Theo1BR Plot",xlab="Tau",ylab="Theo1BR",    */
/*   + log="xy", col="red")                                                  */
/*   > polygon(c(tau, rev(tau)), c(u, rev(l)), col = "grey70", border = NA)  */
/*   > points(tau,t,type="l",col="red")                                      */
/*                                                                           */
/* Other R plot formats can be used as desired                               */
/* An R wrapper function can encapsulate the complete analysis process       */
/*                                                                           */
/* References:                                                               */
/*  [1] B. Lewis, "W Fast Algorithm for Calculation of Theo1", Submitted     */
/*      to IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency   */
/*      Control, May 2020.                                                   */
/*  [2] J.A. Taylor and D.A. Howe, "Fast Theo1BR: A Method for Long Data     */
/*      Set Stability Analysis", IEEE Transactions on Ultrasonics, Ferro-    */
/*      electrics, and Frequency Control, September 2010. I                  */
/*                                                                           */
/* Revision Record:                                                          */
/*  06/10/20  Created and running                                            */
/*  06/16/20  Added draft error bar code                                     */
/*  06/17/20  Editing and debugging                                          */
/*  06/18/20  Changed name to TheoBR()                                       */
/*  06/19/20  Added provisions for single-sided confidence limits            */
/*  06/23/20  Validated and documented                                       */
/*                                                                           */
/* W.J. Riley, Hamilton Technical Services, Beaufort, SC 29907 USA © 2020    */
/* E-mail: bill@wriley.com                                  License: MIT     */
/*                                                                           */
/*****************************************************************************/

#include <Rcpp.h>
using namespace Rcpp;

/*****************************************************************************/
/* Calculate TheoBR for a phase dataset taken at time increments tau0.  The  */
/* phase data are copied to a working array X, and any linear component to   */
/* the dataset is removed as a first step.  The raw all tau Theo1 values are */
/* are calculated using the Lewis fast algorithm of Reference [1], the       */
/* corresponding tau values are written, and Theo1BR bias corrections are    */
/* made per Reference [2].  Chi-squared upper and lower error bars are       */
/* returned in the u and l arrays for the given confidence factor. The # of  */
/* Theo1 results is (N-1)/2.  This code should preferably be compiled with:  */
/* -O3 -ffast-math.                                                          */
/*                                                                           */
/* Arguments:                                                                */
/*   x    = NumericVector = Phase data input                                 */
```

```
/*   t    = NumericVector = Theo1BR output                                  */
/*   u    = NumericVector = Upper Theo1BR error bar                         */
/*   l    = NumericVector = Lower Theo1BR error bar                         */
/*   tau  = NumericVector = Tau output                                      */
/*   tau0 = double = Data sampling time input (default=1)                   */
/*   cf   = double = Confidence factor                                      */
/*                   Negative = Single-Sided (default=0.683)                */
/*   br   = integer = Bias remove flag (0=no, 1=yes, default=1)             */
/*                                                                          */
/* Return = double = Theo1 Bias Factor                                      */
/*          or 1.0 is no bias correction                                    */
/*          or 0 if error                                                   */
/**************************************************************************/

// [[Rcpp::export]]

// The core Theo1BR computational routine was created by
// Ben Lewis of the University of Strathclyde UK
// b.lewis@strayh.ac.uk per Reference [1] above © 2020 IEEE

// TheoBR function
double TheoBR(NumericVector x, NumericVector t, NumericVector u,
  NumericVector l, NumericVector tau, double tau0=1.0, double cf=0.683,
  int br=1){

  // Find phase array size
  int N=x.size();

  // Copy phase data to working array
  double* X = new double[N];
  for(int i = 0; i < N; i++){
    X[i] = x[i];
  }

  // Initializations
  int k_max = (N-1)/2;
  int single = 0; // Flag for single-sided confidence limits
  if(cf<0){
       single=1;
       cf = -cf;
  }
  double* C1 = new double[N];
  double* C3 = new double[k_max+1];
  double* C4 = new double[k_max*2];

  // Preprocess by removing linear part
  double midpoint = ((double) (N-1))/2;
  long double sum1 = 0;
  long double sum2 = 0;
  for(int i = 0; i < N; i++){
    sum1 += X[i];
    sum2 += X[i]*(i-midpoint);
  }
```

```
double a = sum1/N;
double b = sum2/N*12/((double)N*N-1);
for(int i = 0; i < N; i++){
  X[i] -= a + b*(i-midpoint);
}

// Calculate C1
double s=0;
for(int i = 0; i < N; i++){
  s += (X[i]*X[i]);
  C1[i] = s;
}

// Main loop
C3[0] = C1[N-1];
for(int k=1; k<=k_max; k++){
  //Calculate C2 values
  double C2_2k = 0;
  double C2_2k_1 = 0;
  for(int j = 0; j <= N-2*k-1; j++){
    C2_2k += (X[j]*X[j+2*k]);
    C2_2k_1 += (X[j]*X[j+2*k-1]);
  }
  C2_2k_1 += (X[N-2*k]*X[N-1]);

  // Update C3, C4 in place
  for(int v=0; v < k; v++){
    C3[v] -= (X[k-1-v]*X[k-1+v])
           + (X[N-k+v]*X[N-k-v]);
  }
  for(int v = 1;v<=2*k-2;v++){
    C4[v-1] -= (X[2*k-1-v]*X[2*k-1])
             + (X[2*k-2-v]*X[2*k-2])
             + (X[N-2*k]*X[N-2*k+v])
             + (X[N-2*k+1]*X[N-2*k+1+v]);
  }
  C3[k] = C2_2k;
  C4[2*k-2] = 2*C2_2k_1 - (X[0]*X[2*k-1])
            - (X[N-2*k]*X[N-1]);
  C4[2*k-1] = 2*C2_2k;

  //Calculate un-normalised T_k from C1-C4
  double T_k = 0;
  double A0 = C1[N-1] - C1[2*k-1]
            + C1[N-2*k-1] + 2*C2_2k;
  for(int v = 1;v<=k;v++){
    double A1 = A0 - C1[v-1]
              + C1[N-1-v] - C1[2*k-v-1]
              + C1[N-1-2*k+v];
    double A2 = C3[k-v] - C4[v-1]
              - C4[2*k-v-1];
    T_k += (A1+2*A2)/v;
  }
```

```
  // For testing - check for negative Theo1
  // if(T_k<0) return 0;

  // Apply normalisation to get Theo1
  // t[k-1] = (T_k/(3*(double)(N-2*k)*k*k));
  t[k-1] = (T_k/(3*(double)(N-2*k)*k*k*tau0*tau0));
  if(br==0){
    // Take square root for Theo1 deviation
    t[k-1] = sqrt(t[k-1]);
  }

  // Enter tau = averaging factor * tau0
  tau[k-1] = 0.75*2*k*tau0;
}

//Release memory
delete[] C1;
delete[] C3;
delete[] C4;

// Return raw Theo1 if no bias correction
if(br==0){
  return 1.0;
}

// Now have raw all tau Theo1 variances and their corresponding tau values.
// Next need to determine the bias correction factor kf using n=(0.1*N/3)-3
// AVAR/Theo1 variance ratio averages per Ref [2] (other n's have been used,
// e.g., (N/6)-3).  The AVAR and Theo1 variance values used are for the same
// taus, with AFs of 9+3i and 12+4i respectively where i goes from 0 to n
// (sometimes n to 0 has been used instead).  The Theo1 variance values come
// from the t[] array with the index above.
double kf = 0.0;
int n = ((0.1*N)/3)-3;
int m; // Averaging factor

// Correction factor summation loop
for(int i=0; i<=n; i++){
  // Calculate AVAR at AF=m=9+3i
  m=9+3*i;
  double avar = 0.0;
  for(int j=0; j<N-2*m; j++){
    // avar += pow((X[j+2*m] - 2*X[j+m] + X[j]), 2);
    avar += ((X[j+2*m] - 2*X[j+m] + X[j])*(X[j+2*m] - 2*X[j+m] + X[j]));
  }

  // Apply scale factor
  double M=m;
  avar /= (2*M*M*(N-2*M)*tau0*tau0);

  // For testing
```

```
    // Rprintf("i=%d, m=%d, adev=%e, theo1 dev=%e\n",
i,m,sqrt(avar),sqrt(t[5+2*i]));

    // Divide AVAR by corresponding Theo1 variance
    // and add that to correction factor sum
    kf += (avar / t[5+2*i]);
  }
  // Divide kf sum by # ratios
  kf /= (n+1);

  // For testing
  // Rprintf("kf=%e\n", kf);

  // For testing - Force kf=1.0
  // kf=1.0;

  // Apply bias corrections and convert to Theo1 deviation.
   for(int i=0; i < k_max; i++){
    t[i] *= kf;
    t[i] = sqrt(t[i]);
   }

  // To set error bars, must determine
  // dominant power law noise type from
  // Theo1 bias correction factor, then
  // calculate Theo1 edf, and use that to
  // determine upper and lower chi-squared
  // confidence limits at (say) 68%
  // 1-sigma Confidence Factor

  // Use Theo1 bias factor to determine noise type
  // There is one bias factor value and one noise
  // type determination for the whole run
  // Thresholds for noise sorting are geometric means
  // of nominal Theo1 variance bias values

  // Note: Nominal Theo1 variance bias factors in
  // original 2003 FCS paper were:
  // WPM=0.4, FPM=0.6, WFM=1.0, FFM=1.71, RWFM=2.24
  // These are not used here

  // Error bar determination loop
  for(int i=0; i < k_max; i++){
    // Get nominal Theo1 var bias factors for various
    // power law noise types versus averaging factor
    m=i+1;
    // double mm = 0.75*m;
    double mm = 0.75*2*m;
    double bWPM = (0.09 + (0.74/pow(mm, 0.40)));
    double bFPM = (0.14 + (0.82/pow(mm, 0.30)));
    double bWFM = 1.00;
    double bFFM = (1.87 + (-1.05/pow(mm, 0.79)));
    double bRWFM = (2.70 + (-1.53/pow(mm, 0.85)));
```

```
  // For testing
  // Display the nominal bias factors
  // Rprintf("mm=%d, WPM=%f, FPM=%f, WFM=%f, FFM=%f, RWFM=%f\n",
  //   mm,bWPM,bFPM,bWFM,bFFM,bRWFM);

  // Find noise type
  int a; // Power law noise exponent alpha
  if(kf<sqrt(bWPM*bFPM)) a=2;
  else if(kf<sqrt(bFPM*bWFM)) a=1;
  else if(kf<sqrt(bWFM*bFFM)) a=0;
  else if(kf<sqrt(bFFM*bRWFM))a=-1;
  else a=-2;

  // Find edf
  // N is # phase data points
  // mm is tau-s the Theo1 stride = 0.75*m
  double edf; // Theo1 EDF
  if(a==2) edf=(0.86*(N+1)*(N-((4.0/3.0)*mm)))/
    ((N-mm)*(mm/(mm+1.14)));
  if(a==1) edf=((4.798*N*N)-(6.374*N*mm)+(12.387*mm))/
    (sqrt(mm+36.6)*(N-mm)*(mm/(mm+0.3)));
  if(a==0) edf=(((4.1*N+0.8)/mm)-((3.1*N+6.5)/N))*
    (pow(mm,1.5)/(pow(mm,1.5)+5.2));
  if(a==-1) edf=(2.0*N*N-1.3*N*mm-3.5*mm)/
    (N*mm)*((mm*mm*mm)/(mm*mm*mm+2.3));
  if(a==-2) edf=(4.4*N-2)/(2.9*mm)*(((4.4*N-1)*
    (4.4*N-1)-8.6*mm*(4.4*N-1)+
    11.4*mm*mm)/((4.4*N-3)*(4.4*N-3)));

  edf=((((4.1*N)+0.8)/mm)-(((3.1*N)+6.5)/N))*(pow(mm,1.5)/(pow(mm,1.5)+5.2));

  // Find inverse upper and lower Chi-squared values for given confidence
  // factor and edf with an embedded R calls.
  // See: https://teuder.github.io/rcpp4everyone_en/
  // 220_dpqr_functions.html#chi-squared-distribution
  // Set confidence interval and fill in upper and lower error bar arrays
  // Lower error bar equals nominal TheoBR for single-sided conf interval
  if(single){
      double upper = R::qchisq((cf),edf,0,0);
    u[i] = sqrt(t[i]*t[i]*edf/upper);
    l[i] = t[i];
  }
  else{ // Double-sided confidence intervals:
    double lower = R::qchisq((1-cf)/2,edf,0,0);
    double upper = R::qchisq(1-(1-cf)/2,edf,0,0);
    u[i] = sqrt(t[i]*t[i]*edf/upper);
    l[i] = sqrt(t[i]*t[i]*edf/lower);
  }

  // For testing
  // if(i==4) Rprintf("tau=%f, min=%f, nom=%f, max=%f\n", tau[i],l[i],t[i],u[i]);
}
```

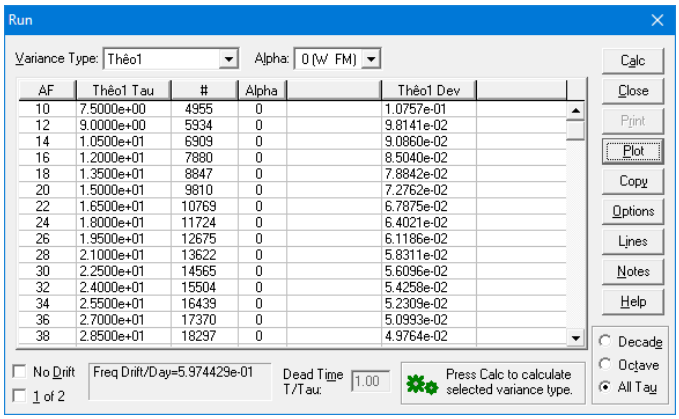
```
  // Done
  return kf;
}
```

Appendix 3

Validation of the ThêoBR() Function

1. Comparison of (raw, no bias correction, W FM) Thêo1 deviation results at m=10, tau=7.5 for 12-point test suite of Reference [8] with that paper, Stable32, and this ThêoBR() function with br=0: All results agree (0.6623816).

2. Comparison of (raw, no bias correction, W FM) Thêo1 deviation results for 1000-point test suite of Reference [13] converted to 1001 points of phase data between Stable32, and this ThêoBR() function with br=0: All results agree.



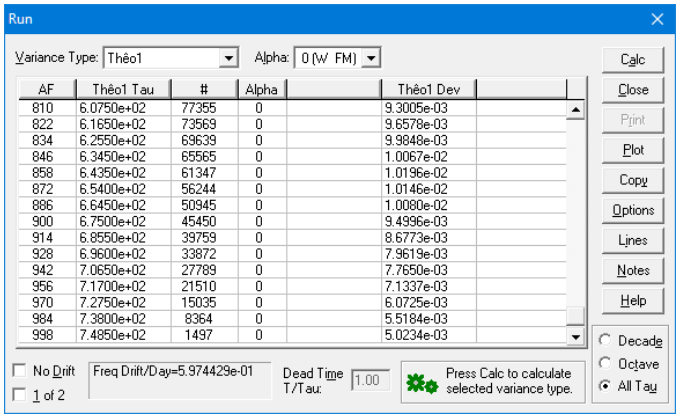
AF	Thêo1 Tau	#	Alpha	Thêo1 Dev
10	7.5000e+00	4955	0	1.0757e-01
12	9.0000e+00	5934	0	9.8141e-02
14	1.0500e+01	6909	0	9.0860e-02
16	1.2000e+01	7880	0	8.5040e-02
18	1.3500e+01	8847	0	7.8842e-02
20	1.5000e+01	9810	0	7.2762e-02
22	1.6500e+01	10769	0	6.7875e-02
24	1.8000e+01	11724	0	6.4021e-02
26	1.9500e+01	12675	0	6.1186e-02
28	2.1000e+01	13622	0	5.8311e-02
30	2.2500e+01	14565	0	5.6096e-02
32	2.4000e+01	15504	0	5.4258e-02
34	2.5500e+01	16439	0	5.2309e-02
36	2.7000e+01	17370	0	5.0993e-02
38	2.8500e+01	18297	0	4.9764e-02

Note that these Stable32 Thêo1 results begin at AF=10

```
> TheoBR(x,t,u,l,tau,1.0,0.683,0)
[1] 1
> r<-data.frame(tau,t)
> r
      tau          t
1     1.5 0.238606329
2     3.0 0.165495896
3     4.5 0.135919826
4     6.0 0.119990934
5     7.5 0.107573989
6     9.0 0.098141065
7    10.5 0.090859630
8    12.0 0.085040334
9    13.5 0.078842085
10   15.0 0.072762345
11   16.5 0.067875271
12   18.0 0.064020929
13   19.5 0.061185935
14   21.0 0.058311245
15   22.5 0.056095559
16   24.0 0.054258251
17   25.5 0.052309314
18   27.0 0.050993418
19   28.5 0.049763631
```

Stable32 Results Thru tau=28,5

ThêoBR()Results thru tau=28.5



AF	Thêo1 Tau	#	Alpha	Thêo1 Dev
810	6.0750e+02	77355	0	9.3005e-03
822	6.1650e+02	73569	0	9.6578e-03
834	6.2550e+02	69639	0	9.9848e-03
846	6.3450e+02	65565	0	1.0067e-02
858	6.4350e+02	61347	0	1.0196e-02
872	6.5400e+02	56244	0	1.0146e-02
886	6.6450e+02	50945	0	1.0080e-02
900	6.7500e+02	45450	0	9.4996e-03
914	6.8550e+02	39759	0	8.6773e-03
928	6.9600e+02	33872	0	7.9619e-03
942	7.0650e+02	27789	0	7.7650e-03
956	7.1700e+02	21510	0	7.1337e-03
970	7.2750e+02	15035	0	6.0725e-03
984	7.3800e+02	8364	0	5.5184e-03
998	7.4850e+02	1497	0	5.0234e-03

Note that these Stable32 Thêo1 results are spaced for "Many Tau" rather than "All Tau"

```
405 607.5 0.009300453
406 609.0 0.009332935
407 610.5 0.009386774
408 612.0 0.009458310
409 613.5 0.009544457
410 615.0 0.009592978
411 616.5 0.009657847
412 618.0 0.009737940
413 619.5 0.009799095
414 621.0 0.009865181
415 622.5 0.009918776
416 624.0 0.009965051
417 625.5 0.009984786
418 627.0 0.009997636
419 628.5 0.009993586
420 630.0 0.009971768
421 631.5 0.009986769
422 633.0 0.010041175
423 634.5 0.010067155
424 636.0 0.010108137
425 637.5 0.010146816
426 639.0 0.010170753
427 640.5 0.010180560
428 642.0 0.010213050
429 643.5 0.010196017
430 645.0 0.010169830
431 646.5 0.010105974
432 648.0 0.010082754
433 649.5 0.010081004
434 651.0 0.010090376
435 652.5 0.010110674
436 654.0 0.010145663
437 655.5 0.010179714
438 657.0 0.010216115
439 658.5 0.010214143
440 660.0 0.010187944
441 661.5 0.010138962
442 663.0 0.010094536
443 664.5 0.010080277
444 666.0 0.010038586
445 667.5 0.009990607
446 669.0 0.009941371
447 670.5 0.009858708
448 672.0 0.009761737
449 673.5 0.009638484
450 675.0 0.009499602
451 676.5 0.009349175
452 678.0 0.009216360
453 679.5 0.009072138
454 681.0 0.008969255
455 682.5 0.008867328
456 684.0 0.008775413
457 685.5 0.008677284
458 687.0 0.008564149
459 688.5 0.008396734
460 690.0 0.008218497
461 691.5 0.008099450
462 693.0 0.008040388
463 694.5 0.007988755
464 696.0 0.007961929
465 697.5 0.007922637
466 699.0 0.007917815
467 700.5 0.007917877
468 702.0 0.007908711
469 703.5 0.007814565
470 705.0 0.007785950
471 706.5 0.007764972
472 708.0 0.007731564
473 709.5 0.007679465
474 711.0 0.007594388
475 712.5 0.007446931
476 714.0 0.007334288
477 715.5 0.007282217
478 717.0 0.007133738
479 718.5 0.006946253
480 720.0 0.006798398
481 721.5 0.006676008
482 723.0 0.006561134
483 724.5 0.006419391
484 726.0 0.006267325
485 727.5 0.006072483
486 729.0 0.005891612
487 730.5 0.005787067
488 732.0 0.005739924
489 733.5 0.005720128
490 735.0 0.005671489
491 736.5 0.005632529
492 738.0 0.005518350
493 739.5 0.005428192
494 741.0 0.005338444
495 742.5 0.005281195
496 744.0 0.005199800
497 745.5 0.005111238
498 747.0 0.005040579
499 748.5 0.005023363
500 750.0 0.005052400
```

Stable32 Results at largest taus

ThêoBR()Results at largest taus

3. Add a test code statement to show the ADEV values used for the ThêoBR bias factor determination: `Rprintf("i=%d, m=%d, adev=%e\n", i,m,sqrt(avar));` The ADEV loop goes from i=0 to n where n = ((0.1*N)/3)-3 = 30 and N=# phase data points (1001). The AVAR values used for the average bias factor ratio are at m=9+3i (9 thru 99) and the Thêo1 variance are at m=12+4i (12 thru 132) which correspond to 0-bases t[] array indices 5+2i (5 thru 65). The Stable32 results are edited to remove AFs not applicable. Note that some ThêoBR AFs (m) are missing from the Stable32 "Many Tau" AFs. All results agree.

AF	Tau	#	Alpha	Min Sigma	Sigma	Max Sigma
9	9.0000e+00	983	0	0.0000e+00	9.8404e-02	0.0000e+00
12	1.2000e+01	977	0	0.0000e+00	7.9314e-02	0.0000e+00
15	1.5000e+01	971	0	0.0000e+00	6.5606e-02	0.0000e+00
18	1.8000e+01	965	0	0.0000e+00	5.6539e-02	0.0000e+00
21	2.1000e+01	959	0	0.0000e+00	5.2900e-02	0.0000e+00
24	2.4000e+01	953	0	0.0000e+00	5.1584e-02	0.0000e+00
27	2.7000e+01	947	0	0.0000e+00	5.0223e-02	0.0000e+00
30	3.0000e+01	941	0	0.0000e+00	4.8872e-02	0.0000e+00
33	3.3000e+01	935	0	0.0000e+00	4.7873e-02	0.0000e+00
36	3.6000e+01	929	0	0.0000e+00	4.6902e-02	0.0000e+00
39	3.9000e+01	923	0	0.0000e+00	4.5898e-02	0.0000e+00
42	4.2000e+01	917	0	0.0000e+00	4.4347e-02	0.0000e+00
45	4.5000e+01	911	0	0.0000e+00	4.2290e-02	0.0000e+00
48	4.8000e+01	905	0	0.0000e+00	4.0434e-02	0.0000e+00
51	5.1000e+01	899	0	0.0000e+00	3.9042e-02	0.0000e+00
54	5.4000e+01	893	0	0.0000e+00	3.8031e-02	0.0000e+00
57	5.7000e+01	887	0	0.0000e+00	3.7372e-02	0.0000e+00
60	6.0000e+01	881	0	0.0000e+00	3.6771e-02	0.0000e+00
63	6.3000e+01	875	0	0.0000e+00	3.6339e-02	0.0000e+00
66	6.6000e+01	869	0	0.0000e+00	3.6055e-02	0.0000e+00
69	6.9000e+01	863	0	0.0000e+00	3.5795e-02	0.0000e+00
75	7.5000e+01	851	0	0.0000e+00	3.5429e-02	0.0000e+00
81	8.1000e+01	839	0	0.0000e+00	3.5100e-02	0.0000e+00
87	8.7000e+01	827	0	0.0000e+00	3.4528e-02	0.0000e+00
93	9.3000e+01	815	0	0.0000e+00	3.3540e-02	0.0000e+00
99	9.9000e+01	803	0	0.0000e+00	3.2616e-02	0.0000e+00

```
> TheoBR(x,t,u,l,tau,1.0,0.683,1)
i=0, m=9, adev=9.840403e-02
i=1, m=12, adev=7.931448e-02
i=2, m=15, adev=6.560594e-02
i=3, m=18, adev=5.653874e-02
i=4, m=21, adev=5.290013e-02
i=5, m=24, adev=5.158361e-02
i=6, m=27, adev=5.022271e-02
i=7, m=30, adev=4.887241e-02
i=8, m=33, adev=4.787326e-02
i=9, m=36, adev=4.690239e-02
i=10, m=39, adev=4.589837e-02
i=11, m=42, adev=4.434722e-02
i=12, m=45, adev=4.228999e-02
i=13, m=48, adev=4.043448e-02
i=14, m=51, adev=3.904201e-02
i=15, m=54, adev=3.803126e-02
i=16, m=57, adev=3.737163e-02
i=17, m=60, adev=3.677058e-02
i=18, m=63, adev=3.633950e-02
i=19, m=66, adev=3.605453e-02
i=20, m=69, adev=3.579533e-02
i=21, m=72, adev=3.558817e-02
i=22, m=75, adev=3.542898e-02
i=23, m=78, adev=3.526787e-02
i=24, m=81, adev=3.509985e-02
i=25, m=84, adev=3.483282e-02
i=26, m=87, adev=3.452850e-02
i=27, m=90, adev=3.411870e-02
i=28, m=93, adev=3.354003e-02
i=29, m=96, adev=3.307885e-02
i=30, m=99,  adev=3.261585e-02
```

Edited Overlapping ADEV Results for Test Suite Phase Data — `ThêoBR()` Results

4. For this white FM noise test data, the ADEV and Thêo1 deviation values at the same tau should be approximately the same. At the minimum τ=9, they are 9.8404e-2 and 9.8141e-2 respectively, close to the same, and have an Allan/ Thêo1 variance ratio of 1.0054, very close to the expected 1. At the maximum τ=99, they are 3.2616e-02 and 2.9858e-2 respectively, quite close to the same, and have an Allan/ Thêo1 variance ratio of 1.1933, reasonably close to 1. The complete set of these values is shown below.

```
i=0,  m=9,  adev=9.840403e-02, theo1 dev=9.814106e-02
i=1,  m=12, adev=7.931448e-02, theo1 dev=8.504033e-02
i=2,  m=15, adev=6.560594e-02, theo1 dev=7.276234e-02
i=3,  m=18, adev=5.653874e-02, theo1 dev=6.402093e-02
i=4,  m=21, adev=5.290013e-02, theo1 dev=5.831125e-02
i=5,  m=24, adev=5.158361e-02, theo1 dev=5.425825e-02
i=6,  m=27, adev=5.022271e-02, theo1 dev=5.099342e-02
i=7,  m=30, adev=4.887241e-02, theo1 dev=4.865169e-02
i=8,  m=33, adev=4.787326e-02, theo1 dev=4.682970e-02
i=9,  m=36, adev=4.690239e-02, theo1 dev=4.572499e-02
i=10, m=39, adev=4.589837e-02, theo1 dev=4.402408e-02
i=11, m=42, adev=4.434722e-02, theo1 dev=4.220960e-02
i=12, m=45, adev=4.228999e-02, theo1 dev=4.070391e-02
i=13, m=48, adev=4.043448e-02, theo1 dev=3.979878e-02
i=14, m=51, adev=3.904201e-02, theo1 dev=3.874388e-02
i=15, m=54, adev=3.803126e-02, theo1 dev=3.771616e-02
i=16, m=57, adev=3.737163e-02, theo1 dev=3.697014e-02
i=17, m=60, adev=3.677058e-02, theo1 dev=3.571784e-02
i=18, m=63, adev=3.633950e-02, theo1 dev=3.454916e-02
i=19, m=66, adev=3.605453e-02, theo1 dev=3.376912e-02
i=20, m=69, adev=3.579533e-02, theo1 dev=3.313844e-02
i=21, m=72, adev=3.558817e-02, theo1 dev=3.245458e-02
i=22, m=75, adev=3.542898e-02, theo1 dev=3.178931e-02
i=23, m=78, adev=3.526787e-02, theo1 dev=3.142167e-02
i=24, m=81, adev=3.509985e-02, theo1 dev=3.088395e-02
i=25, m=84, adev=3.483282e-02, theo1 dev=3.062463e-02
i=26, m=87, adev=3.452850e-02, theo1 dev=3.030143e-02
i=27, m=90, adev=3.411870e-02, theo1 dev=2.993241e-02
i=28, m=93, adev=3.354003e-02, theo1 dev=2.993314e-02
i=29, m=96, adev=3.307885e-02, theo1 dev=2.996312e-02
i=30, m=99, adev=3.261585e-02, theo1 dev=2.985803e-02
```

ADEV and Thêo1 Deviation Values Used for Bias Factor Determination

5. The computed Thêo1 bias factor (as returned by the `ThêoBR()` function) for the 1001 phase data point test suite is 1.085666, consistent with its white FM noise type.

6. Simulated 1001-point phase data sets were generated for each of the five power law noise types, and the resulting Thêo1 bias factors were compared with their original values as given in Reference [3], the range of their more accurate ones that have a slight dependence on m as given in Reference [8] for m=12 thru 132, and the bias factor results of Stable32:

Noise Type		W PM	F PM	W FM	F FM	RW FM
Thêo1 Bias Factor	Original	0.4	0.6	1.0	1.71	2.24
	Accurate	0.364-0.195	0.529 -0.330	1.000	1.722 -1.848	2.515 -2.676
	Stable32	0.191	0.356	0.733	1.600	1.854
	Actual	0.255	0.428	1.180	1.515	2.339

There clearly is a wide variation in the bias factor results, but they all have the same general trend.

7. The edfs for setting ThêoBR confidence limits are computed with the same C code as used in Stable32. They were verified as the same.

8. The Chi-squared values for setting single and double-sided error bars are computed using the `qchisq()` R function. That function's usage was confirmed against published tables and verified against the Stable32 values.

9. Selected single and double-sided ThêoBR minimum, nominal, and maximum values were verified against Stable32.

10. Regardless of all of the above checks, this program is new and it should be used with care. Please report any problems to bill@wriley.com.

Appendix 4

R Top-Level Driver Function for a ThêoBR Run

With R, it is easy to write a top-level driver function for a ThêoBR run according to your preferences. This is especially convenient for processing phase data files of differing sizes, and to display and/or plot the analysis results. An example of such an R driver function is shown below:

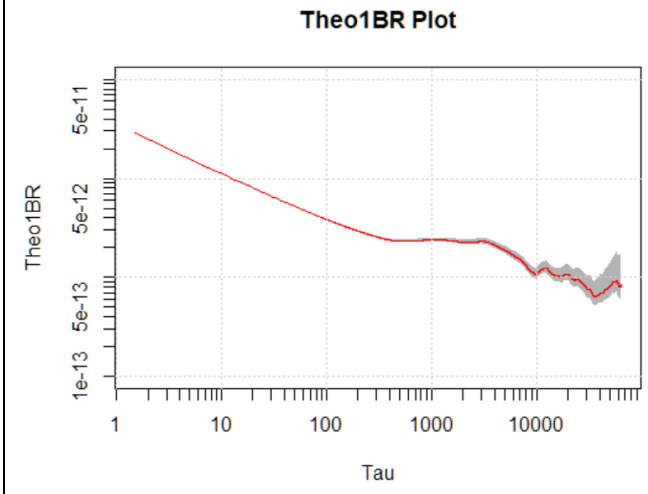
The trun() "ThêoBR Run" function requires the x argument with the phase data array name (one assumes that it has already been read into R with, for example, `> x<-scan("C:\\Data\\phase.dat")`. The trun() function has optional arguments for setting the data sampling interval and error bar confidence interval. It always sets br=1 for bias removal and generates double-sided error bars. The ThêoBR results are plotted along with their error bar region, and a data frame with the tabulated tau, minimum, nominal, and maximum. The bias correction factor and estimated power law noise type exponent are printed, and the ThêoBR results are returned invisibly. To get those results, assign the function return to a variable name, e.g., `> r<-trun(x)`. The x array is not changed and the tau, l,t, and u arrays are not returned or visible to the caller. The printed bias correction factor implies the dominant noise type. This driver function can be invoked on the R command line and saved as a trun.R file.

```
trun <- function(x, tau0=1.0, ci=0.683)
{
  # Allocate arrays
  N=length(x)
  t<-1:((N-1)/2)-1
  u<-1:((N-1)/2)-1
  l<-1:((N-1)/2)-1
  tau<-1:((N-1)/2)-1

  # Call TheoBR()
  cf=TheoBR(x,t,u,l,tau,tau0,ci,1)

  # Print the bias correction factor
  print(cf)

  # Crudely estimate noise type using original nominal cf
   # WPM=0.4, FPM=0.6, WFM=1.0, FFM=1.71, RWFM=2.24
  if(cf<sqrt(0.4*0.6)) a=2
  else if(cf<sqrt(0.6*1.0)) a=1
  else if(cf<sqrt(1.0*1.71)) a=0
  else if(cf<sqrt(1.71*2.24))a=-1
  else a=-2

  # Print power law noise exponent
  print(a)

  # Create data frame for results
  r<-data.frame(tau,l,t,u)

  # Get plot y scale
  top=10^(ceiling(log10(max(t))))
  bot=10^(floor(log10(min(t))))

  # Plot results
  plot(tau,t,type="l",ylim=c(bot,top),main="Theo1BR Plot",xlab="Tau",
ylab="Theo1BR",log="xy",col="red")
  polygon(c(tau, rev(tau)), c(u, rev(l)), col = "grey70", border = NA)
  points(tau,t,type="l",col="red")
  grid()
```

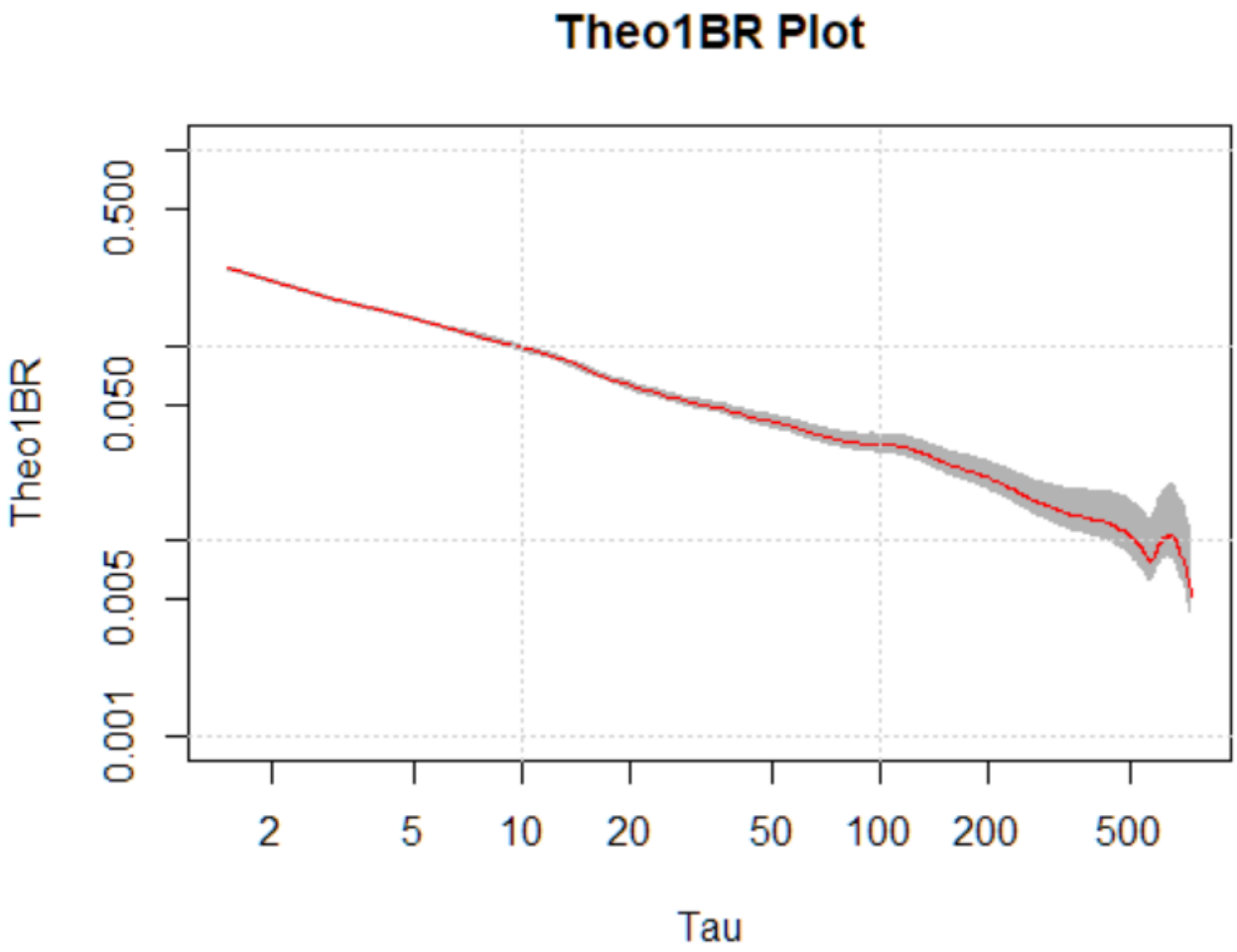
```
  # Return data frame without printing it
  invisible(r)
}
```

If desired, log scale ticks can then be added to the plot with another R function and example (not used in the following plots):

```
logtics <- function(x, y)
{
  # Get log scale limits
  left = floor(log10(min(x)))
  right = ceiling(log10(max(x)))
  bot = floor(log10(min(y)))
  top = ceiling(log10(max(y)))
  # Draw X-Axis Tics
  for(xx in left:right)
  {
    axis(side=1,at=(2:9)*10^xx,labels=FALSE)
  }
  # Draw Y-Axis Tics
  for(yy in bot:top)
  {
    axis(side=2,at=(2:9)*10^yy,labels=FALSE)
  }
}
```



This function was invoked for the 1001-point phase data test suite with the confidence interval set to 95% to emphasize the error bars. The resulting ThêoBR plot is shown below where the grey shading denotes the 2-sigma (95%) confidence interval region. More elaborate plot formats are, of course, possible. The ThêoBR bias factor was 1.085666 (W FM).

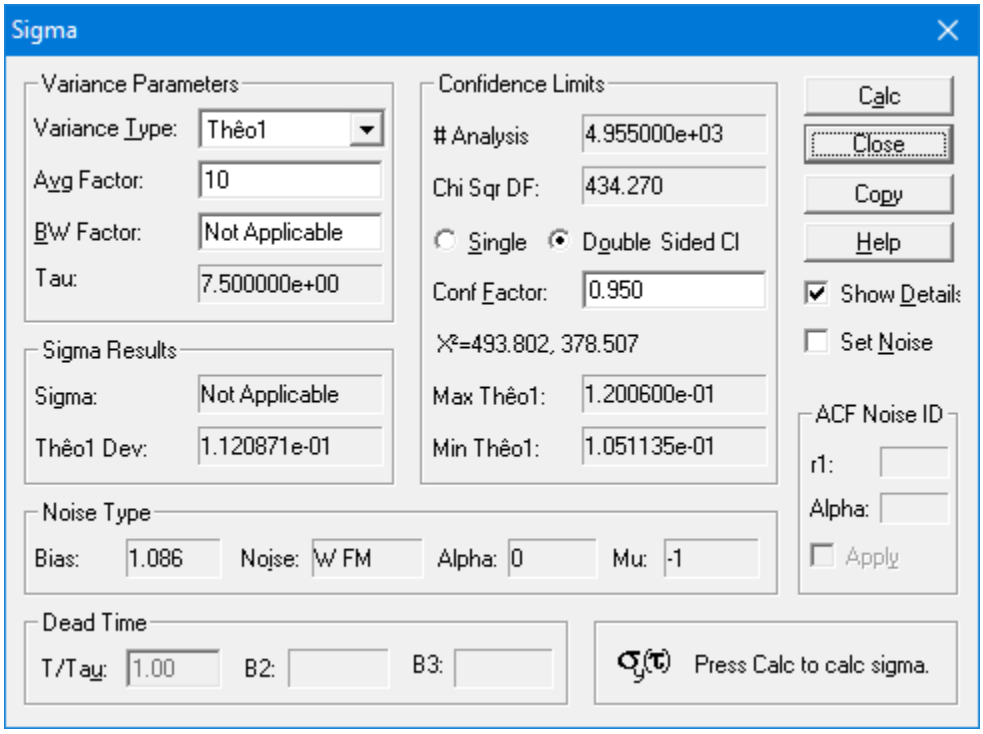


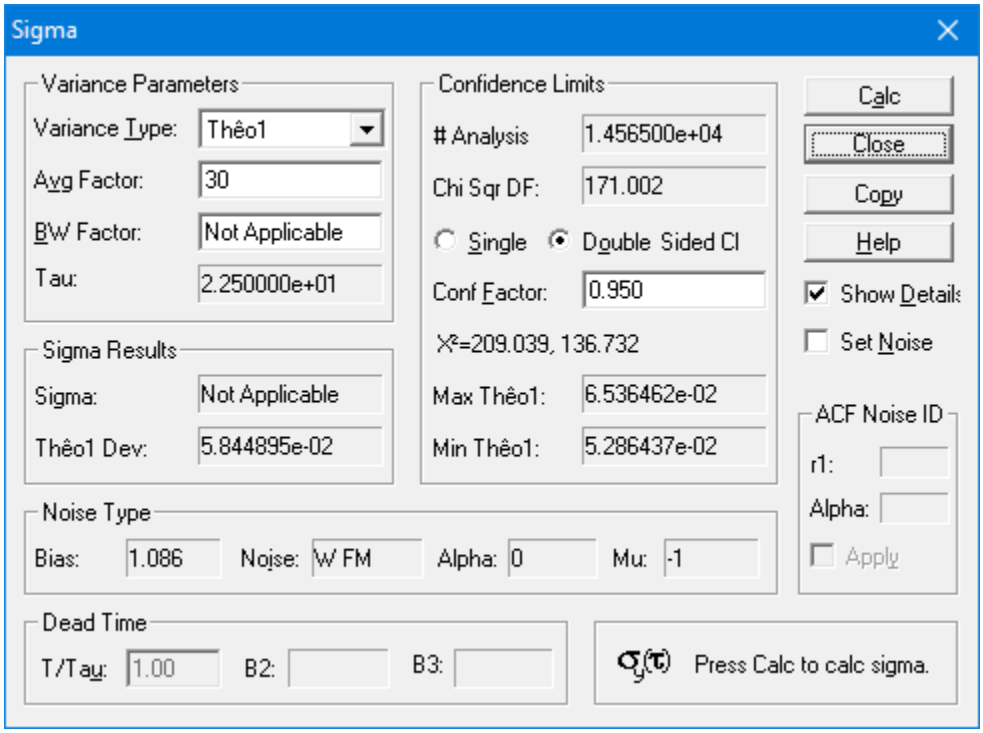
Create and display the first few rows of the data frame with the ThêoBR results (tau, minimum, nominal and maximum ThêoBR):

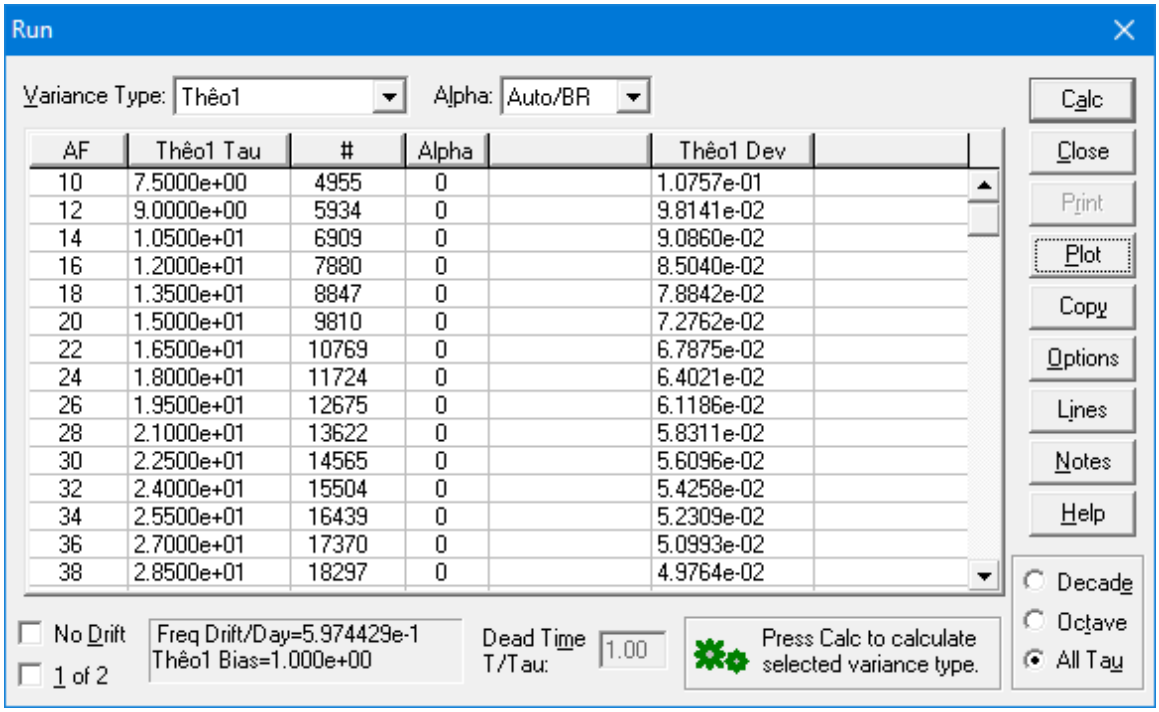
```
> r<-data.frame(tau,l,t,u)
> r
      tau           l          t          u
1     1.5 0.236361199 0.24861662 0.26222248
2     3.0 0.163755974 0.17243897 0.18210164
3     4.5 0.133973395 0.14162209 0.15020401
4     6.0 0.117754145 0.12502493 0.13325995
5     7.5 0.105102919 0.11208706 0.12007305
6     9.0 0.095475220 0.10225839 0.11008707
7    10.5 0.088026711 0.09467148 0.10240968
8    12.0 0.082062725 0.08860804 0.09629674
9    13.5 0.075792700 0.08214976 0.08967987
10   15.0 0.069693163 0.07581495 0.08312496
11   16.5 0.064784317 0.07072285 0.07786934
12   18.0 0.060899059 0.06670681 0.07374865
13   19.5 0.058012463 0.06375288 0.07076398
14   21.0 0.055112441 0.06075759 0.06770129
```

```
15    22.5 0.052856231 0.05844895 0.06537565
```

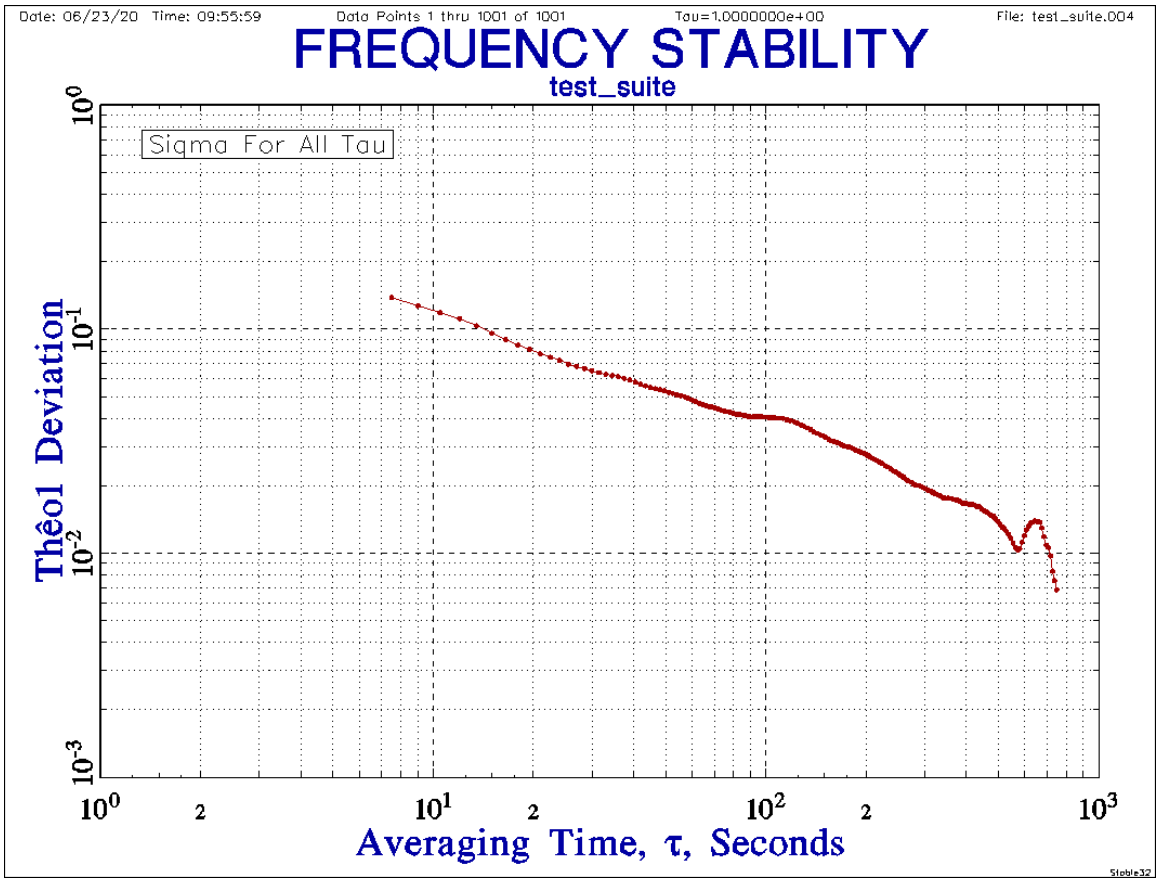
The same data set was analyzed in Stable32 which produced similar ThêoBR results.







AF	Théo1 Tau	#	Alpha		Théo1 Dev
10	7.5000e+00	4955	0		1.0757e-01
12	9.0000e+00	5934	0		9.8141e-02
14	1.0500e+01	6909	0		9.0860e-02
16	1.2000e+01	7880	0		8.5040e-02
18	1.3500e+01	8847	0		7.8842e-02
20	1.5000e+01	9810	0		7.2762e-02
22	1.6500e+01	10769	0		6.7875e-02
24	1.8000e+01	11724	0		6.4021e-02
26	1.9500e+01	12675	0		6.1186e-02
28	2.1000e+01	13622	0		5.8311e-02
30	2.2500e+01	14565	0		5.6096e-02
32	2.4000e+01	15504	0		5.4258e-02
34	2.5500e+01	16439	0		5.2309e-02
36	2.7000e+01	17370	0		5.0993e-02
38	2.8500e+01	18297	0		4.9764e-02

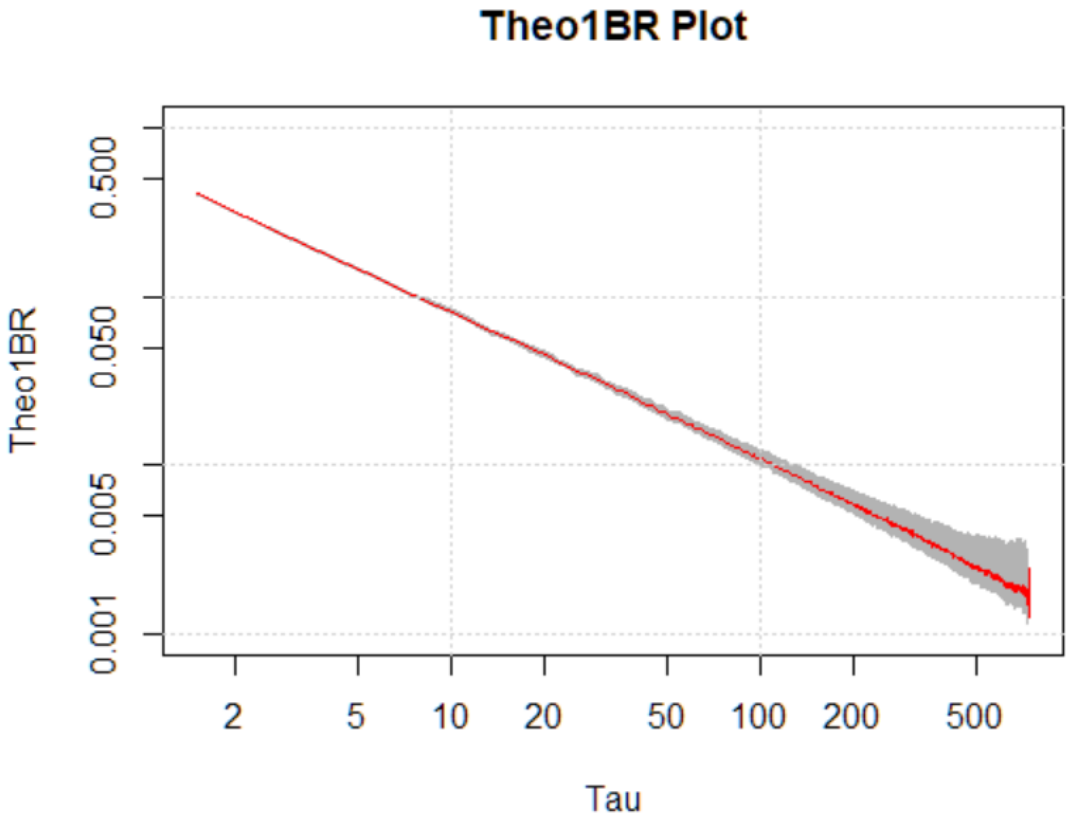


Note that this is actually a “Many Tau” plot with N=500.

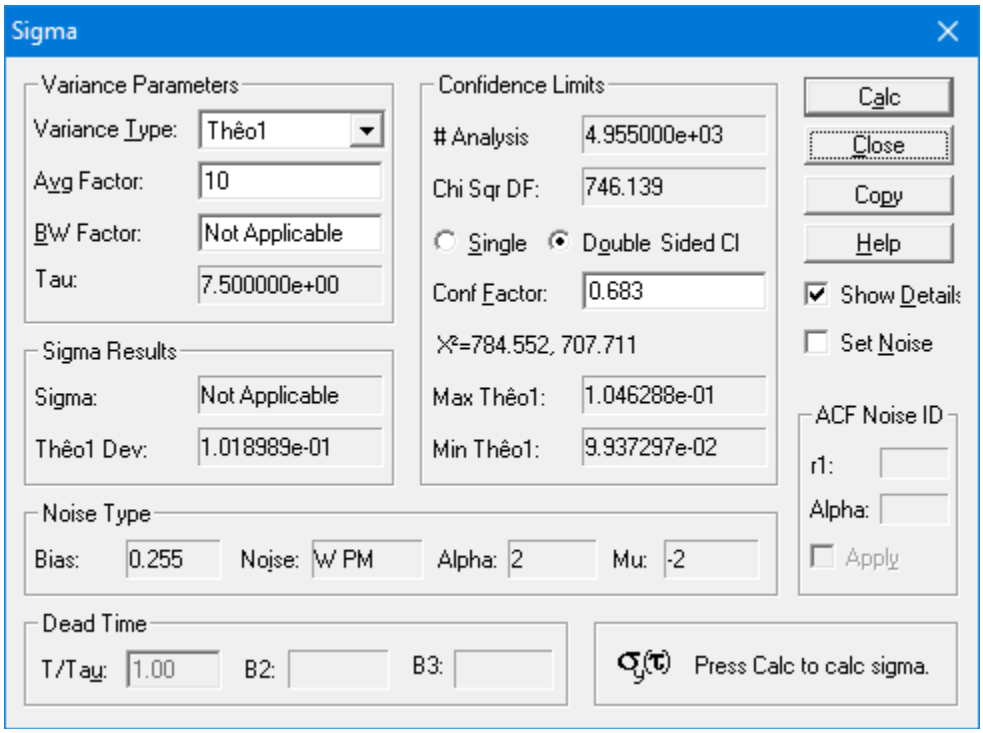
Appendix 5

Additional Examples

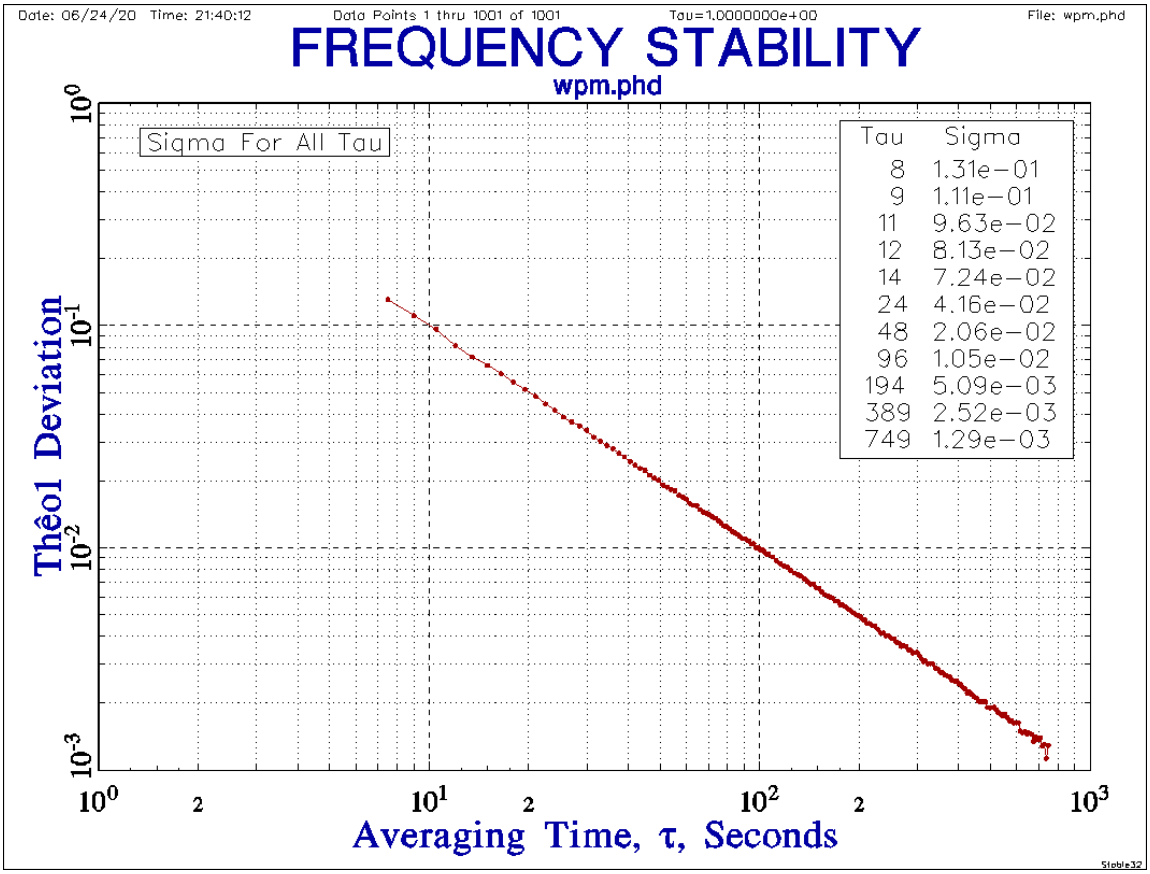
White PM wpm.phd wpm

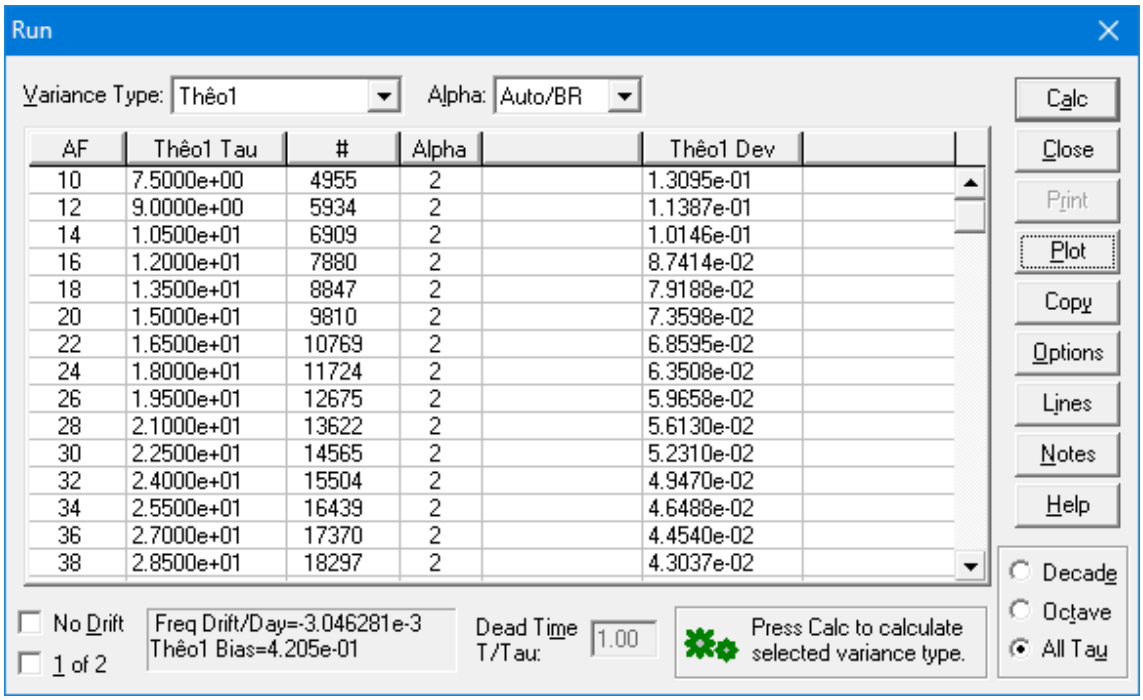



```
> r<-trun(wpm)
[1] 0.2546437
[1] 2
> r
      tau           l           t           u
1     1.5 0.390243138 0.400441682 0.411483464
2     3.0 0.215129195 0.220876926 0.227111109
3     4.5 0.150534255 0.154860301 0.159581855
4     6.0 0.121114878 0.124874884 0.129008079
5     7.5 0.098608189 0.101898904 0.105542464
6     9.0 0.085560115 0.088609398 0.092009566
7    10.5 0.076077409 0.078954987 0.082185663
8    12.0 0.065410399 0.068021821 0.070973019
9    13.5 0.059140764 0.061621141 0.064442045
10   15.0 0.054863471 0.057270859 0.060025515
```



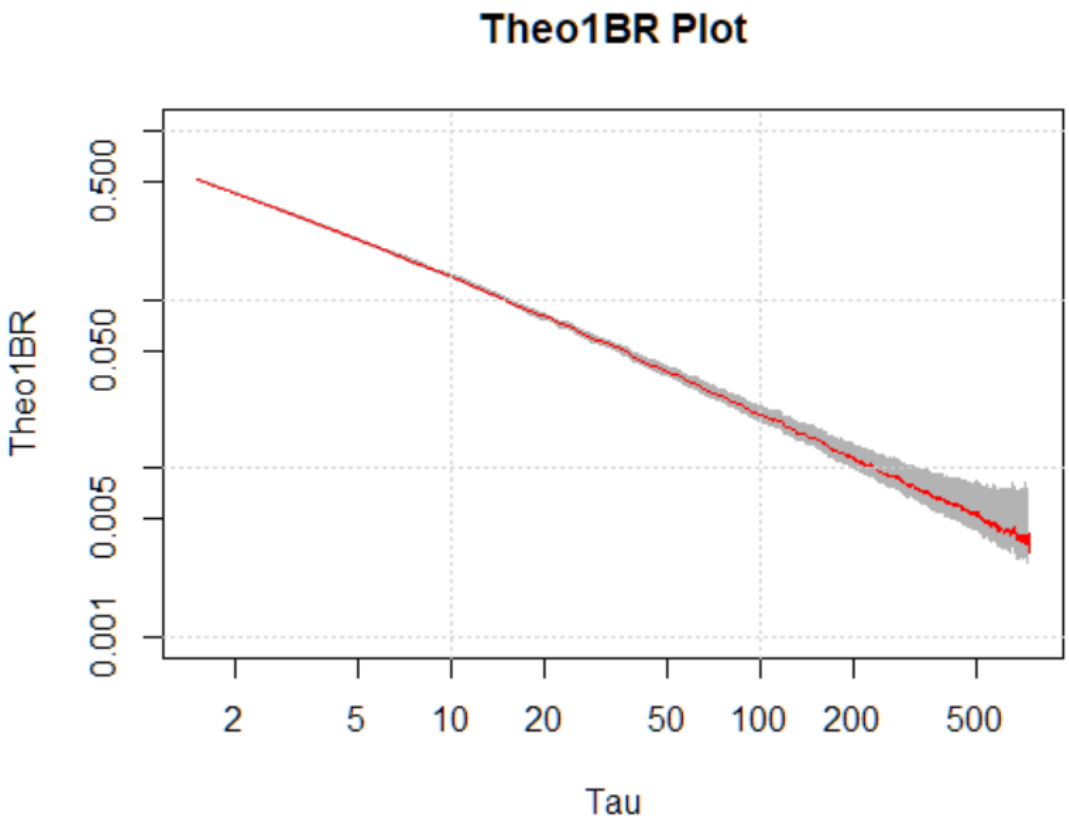
The ThêoBR and Stable32 Sigma functions produce the same nominal Thêo1 and bias factor for the W PM noise, and nearly the same 1-sigma error bars.



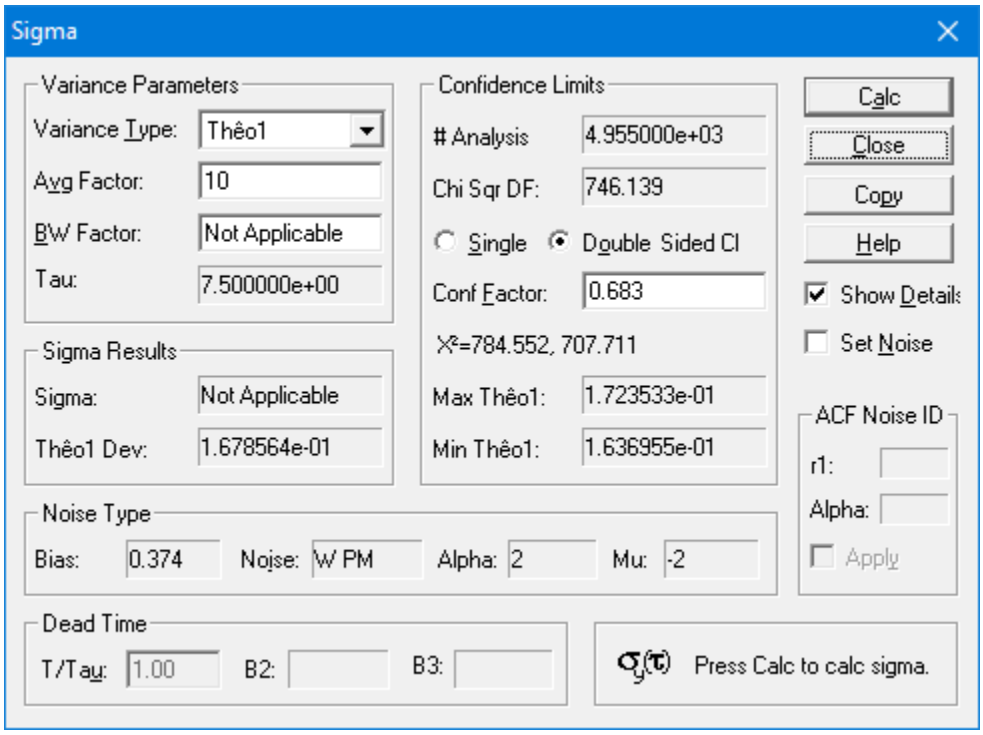


The Stable32 ThêoBR Run and Plot produce slightly different results because of a different bias factor.

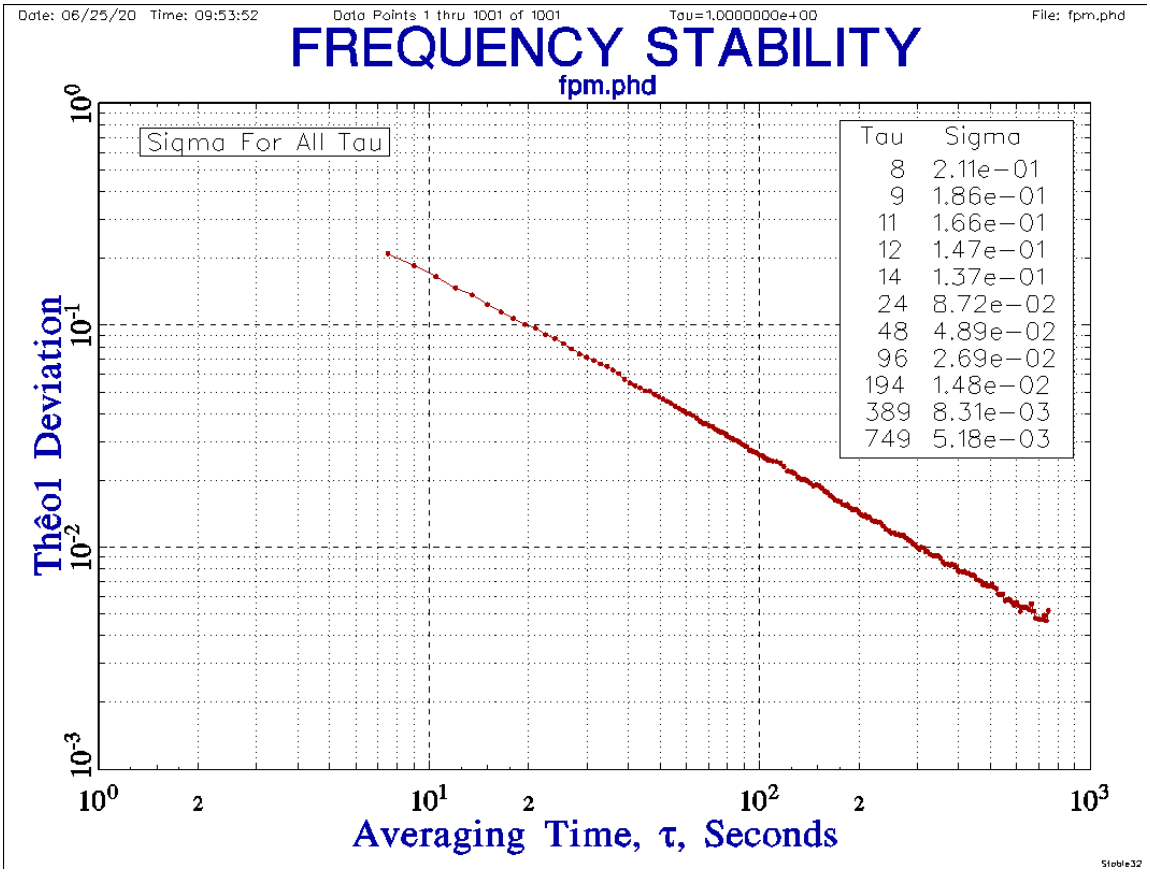
Flicker PM fpm.phd fpm

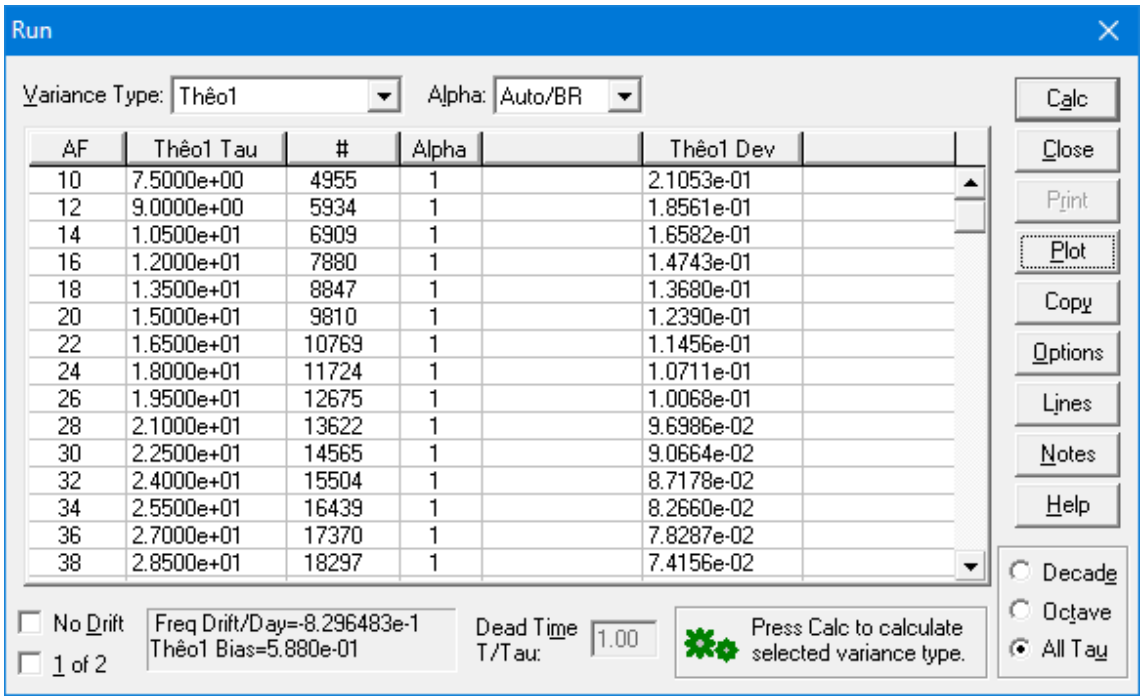



```
> r<-trun(fpm)
[1] 0.3737958
[1] 2
> r
      tau           l           t           u
1     1.5 0.505634967 0.518849140 0.533155890
2     3.0 0.315628560 0.324061390 0.333207922
3     4.5 0.233480032 0.240189769 0.247512944
4     6.0 0.189619689 0.195506425 0.201977430
5     7.5 0.162435703 0.167856446 0.173858425
6     9.0 0.142895464 0.147988126 0.153666807
7    10.5 0.127386652 0.132204967 0.137614524
8    12.0 0.113034665 0.117547423 0.122647341
9    13.5 0.104683239 0.109073678 0.114066873
10   15.0 0.094632242 0.098784669 0.103536086
```

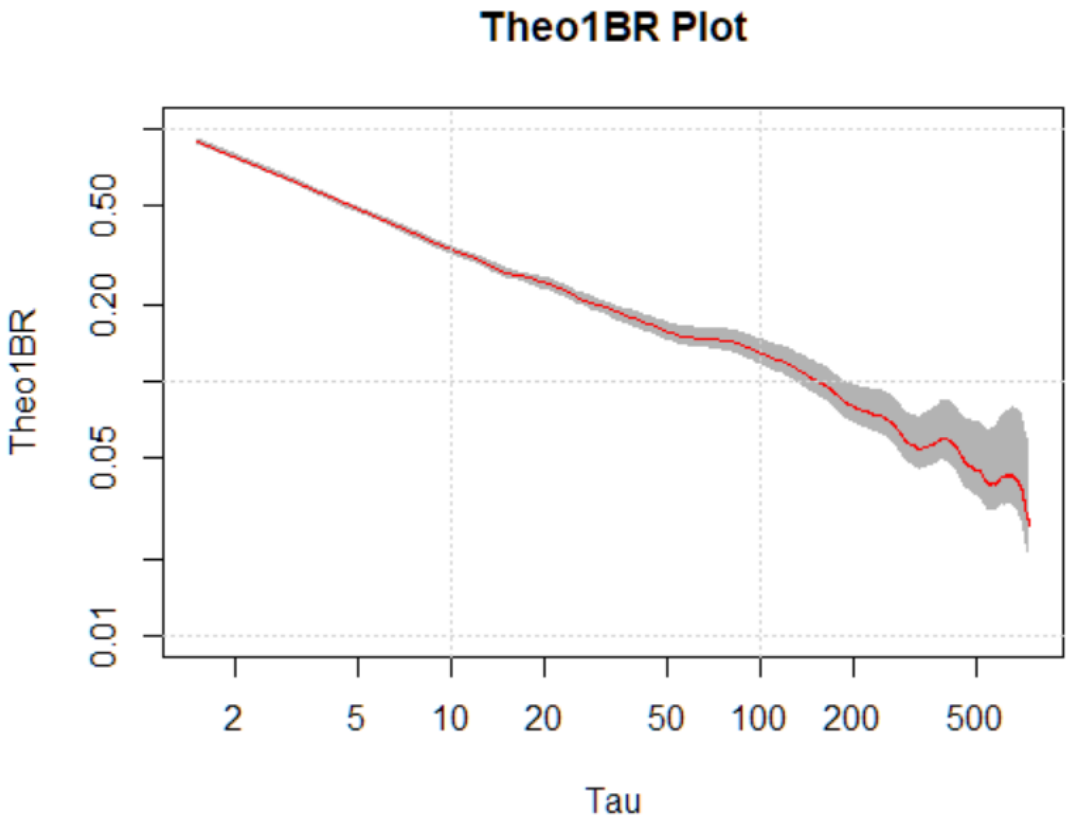


The ThêoBR and Stable32 Sigma functions produce the same nominal Thêo1 and bias factor for the F PM noise, and nearly the same 1-sigma error bars. The noise is identified as W PM rather than F PM.

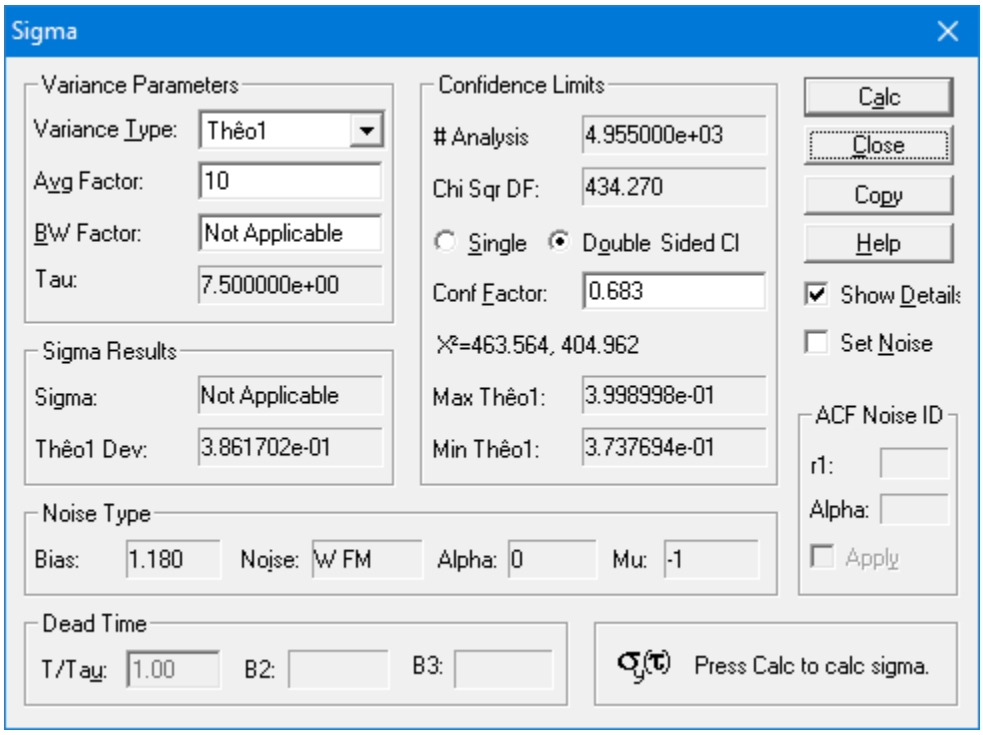




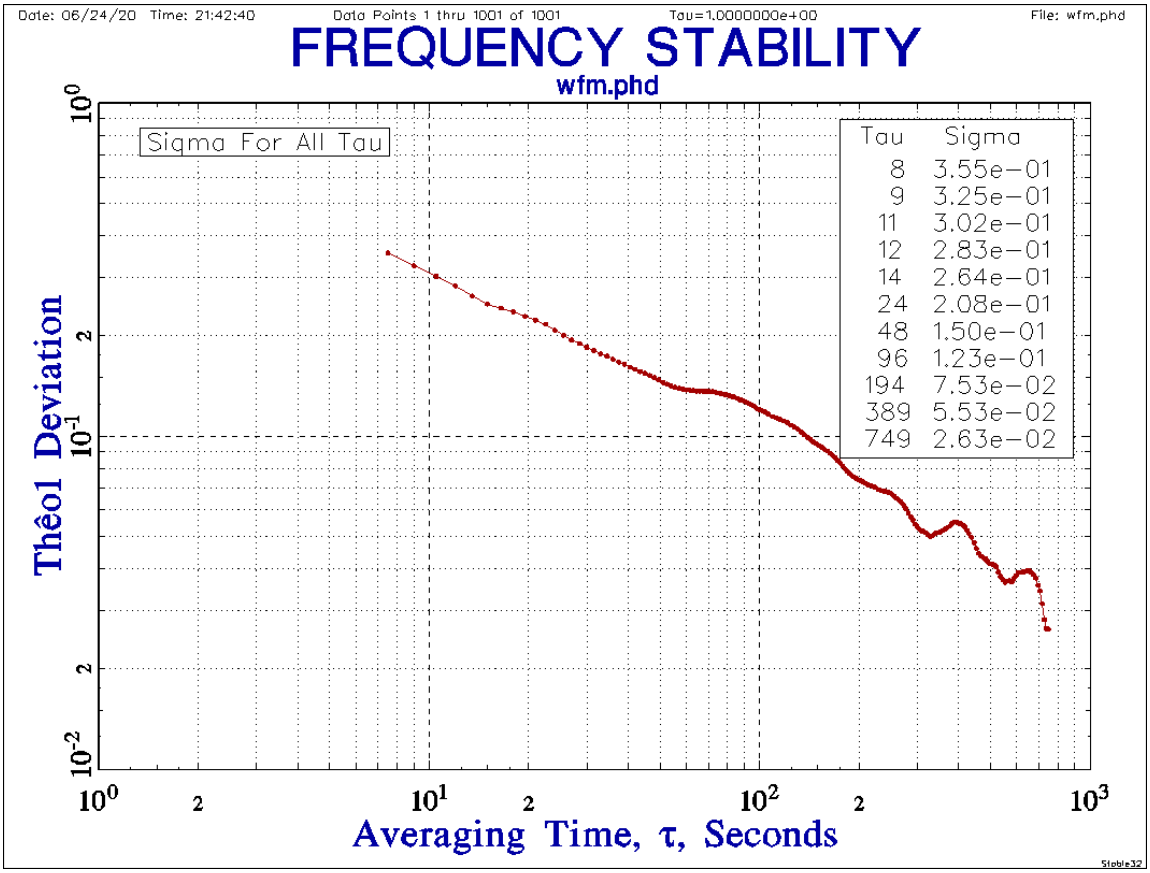
The Stable32 ThêoBR Run and Plot produce slightly different results because of a different bias factor.

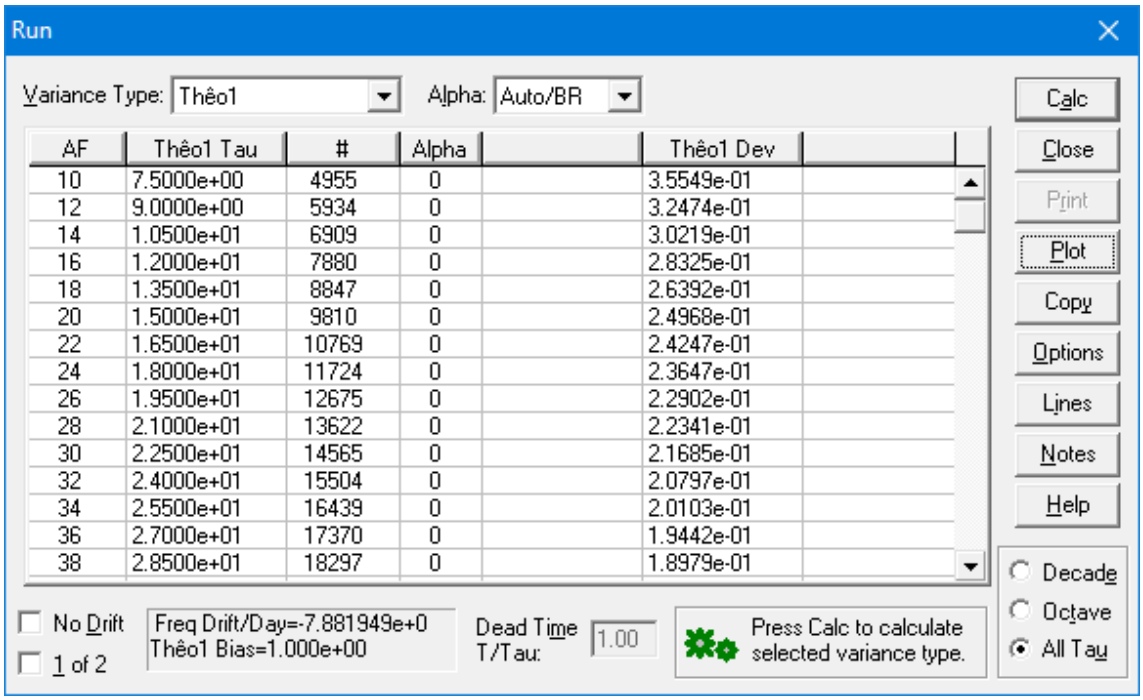



```
> r<-trun(wfm)
[1] 1.180065
[1] 0
> r
      tau          l          t          u
1     1.5 0.86912455 0.89183809 0.91642964
2     3.0 0.61581492 0.63226801 0.65011358
3     4.5 0.49274597 0.50690647 0.52236161
4     6.0 0.42333857 0.43648110 0.45092805
5     7.5 0.37369928 0.38617023 0.39997837
6     9.0 0.34062912 0.35276883 0.36630546
7    10.5 0.31630424 0.32826824 0.34170030
8    12.0 0.29588356 0.30769631 0.32104604
9    13.5 0.27515789 0.28669808 0.29982259
10   15.0 0.25982648 0.27122757 0.28427327
```

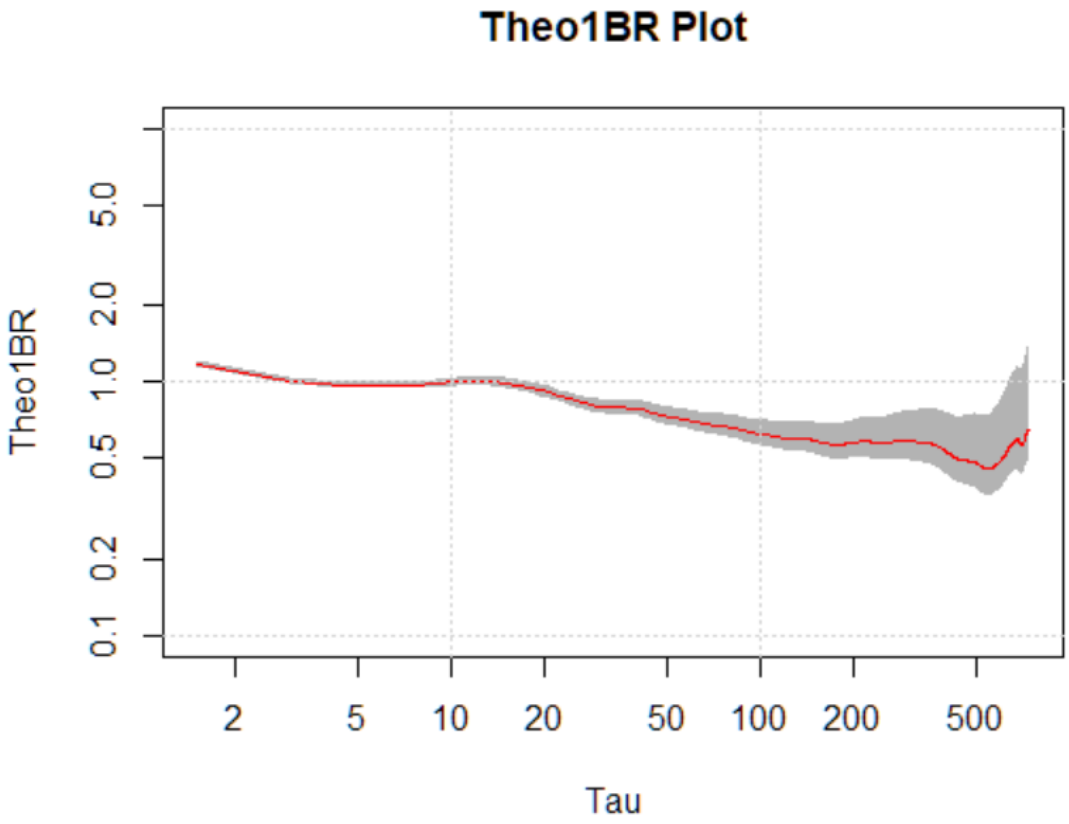


The ThêoBR and Stable32 Sigma functions produce the same nominal Thêo1 and bias factor for the W FM noise, nearly the same 1-sigma error bars.

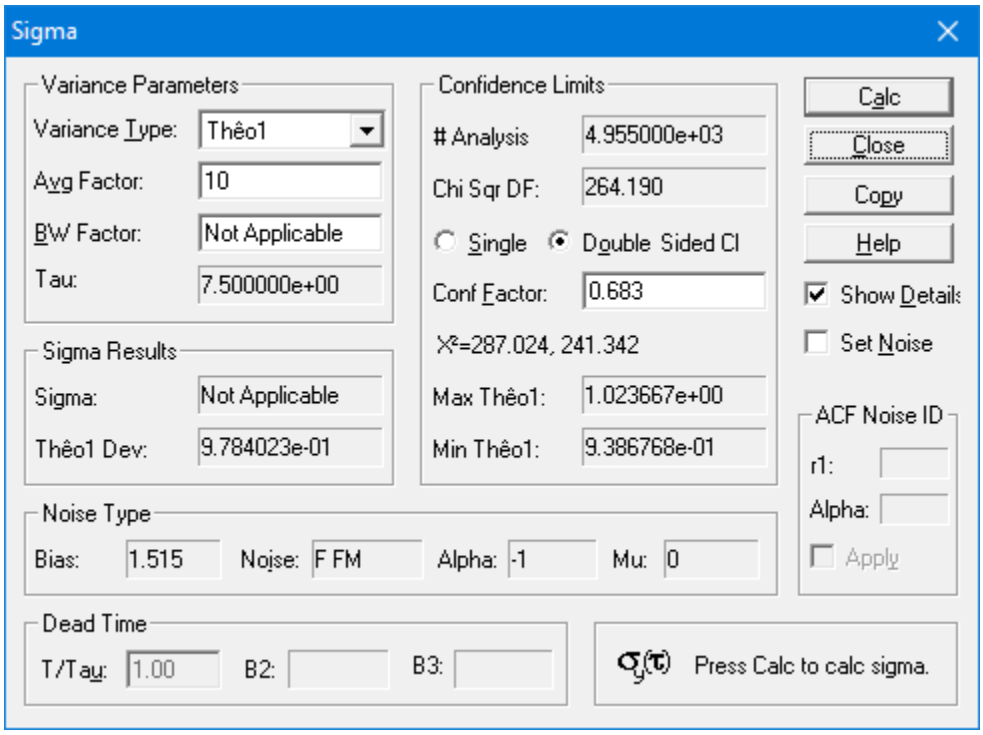




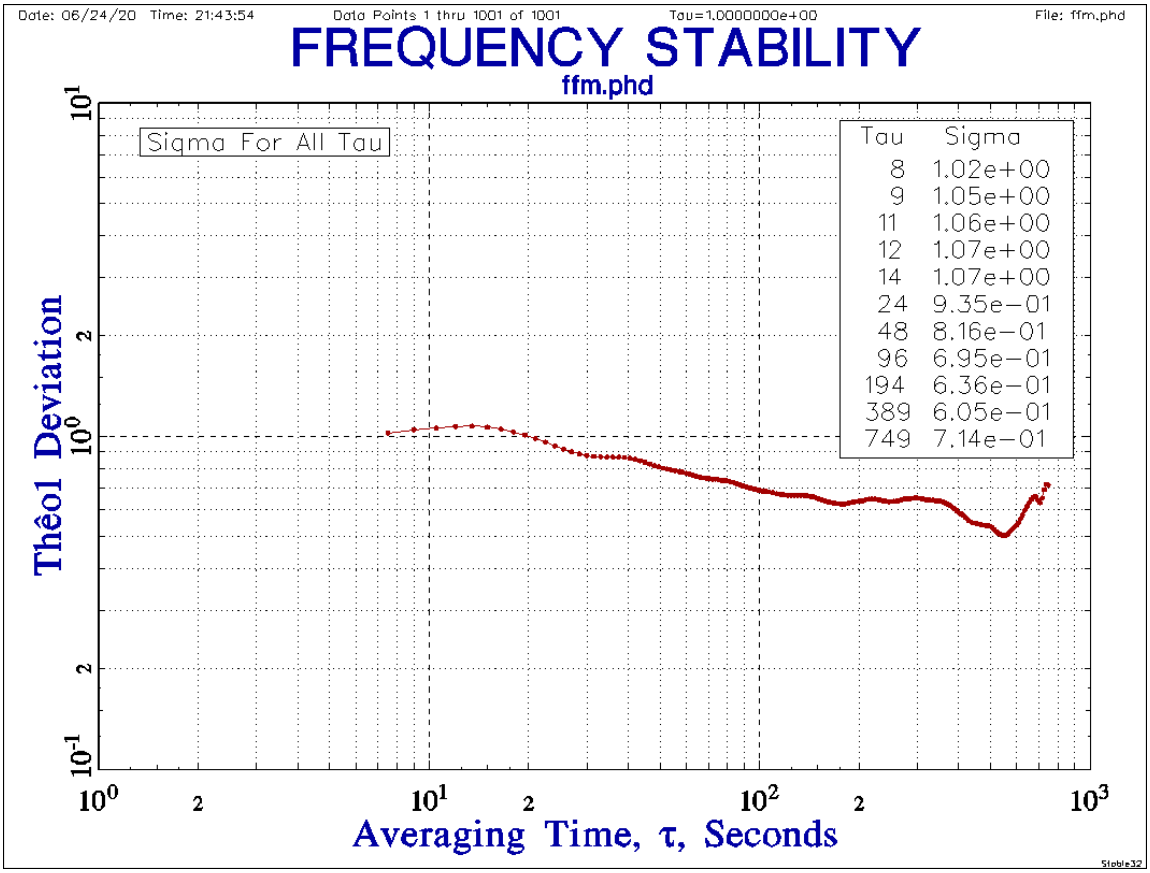
The Stable32 ThêoBR Run and Plot produce slightly different results because of a different bias factor.

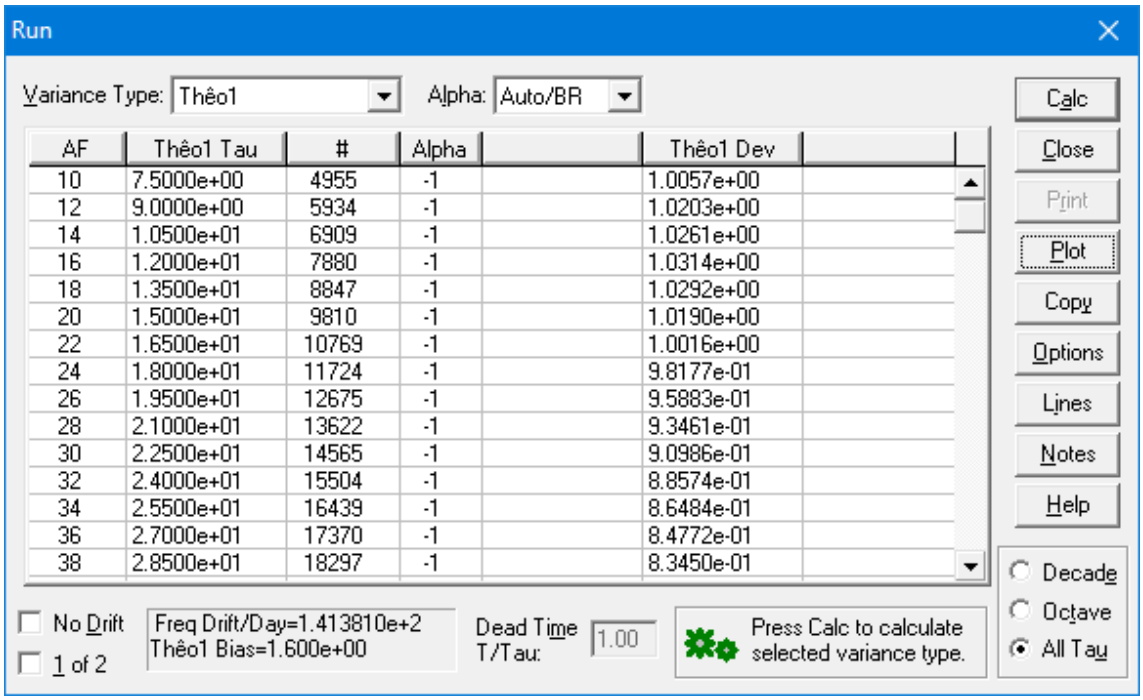



```
> r<-trun(ffm)
[1] 1.514804
[1] -1
> r
      tau         l         t         u
1     1.5 1.1546793 1.1848555 1.2175267
2     3.0 0.9838932 1.0101804 1.0386924
3     4.5 0.9474265 0.9746536 1.0043700
4     6.0 0.9425662 0.9718281 1.0039944
5     7.5 0.9468058 0.9784023 1.0133866
6     9.0 0.9584494 0.9926077 1.0306966
7    10.5 0.9618735 0.9982557 1.0391023
8    12.0 0.9649376 1.0034615 1.0469977
9    13.5 0.9609765 1.0012801 1.0471169
10   15.0 0.9496628 0.9913337 1.0390156
```

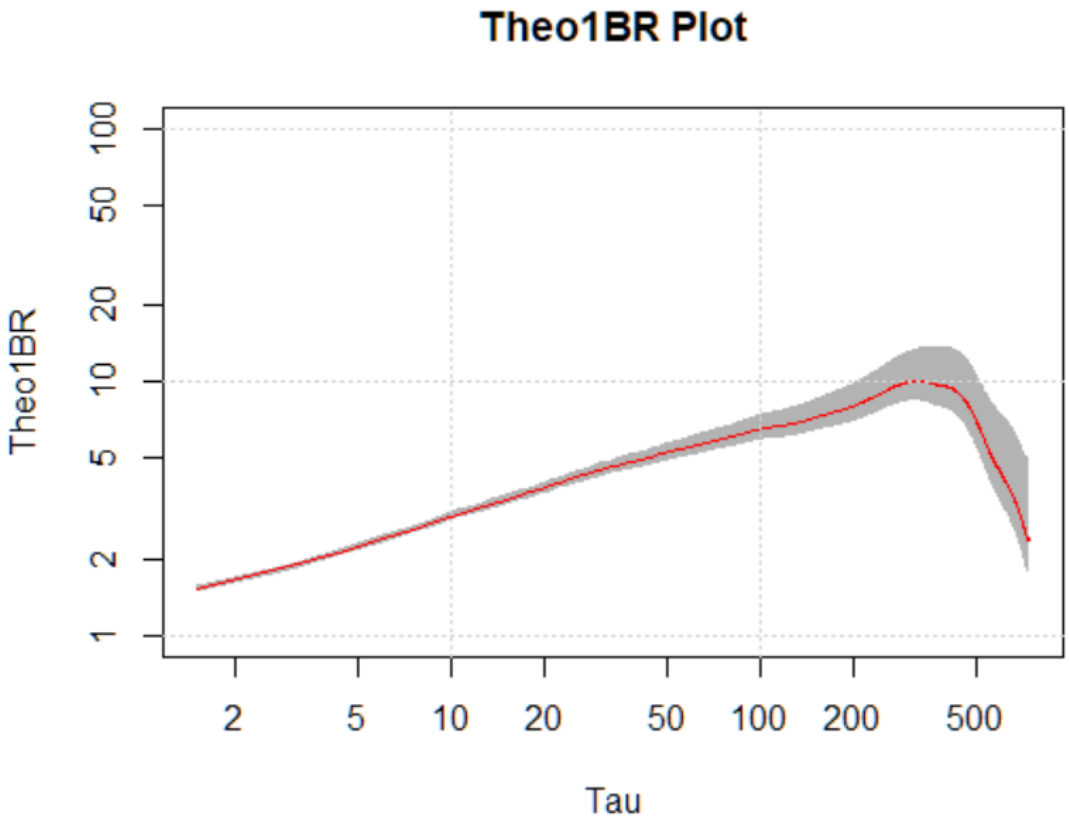


The ThêoBR and Stable32 Sigma functions produce the same nominal Thêo1 and bias factor for the F FM noise, nearly the same 1-sigma error bars.



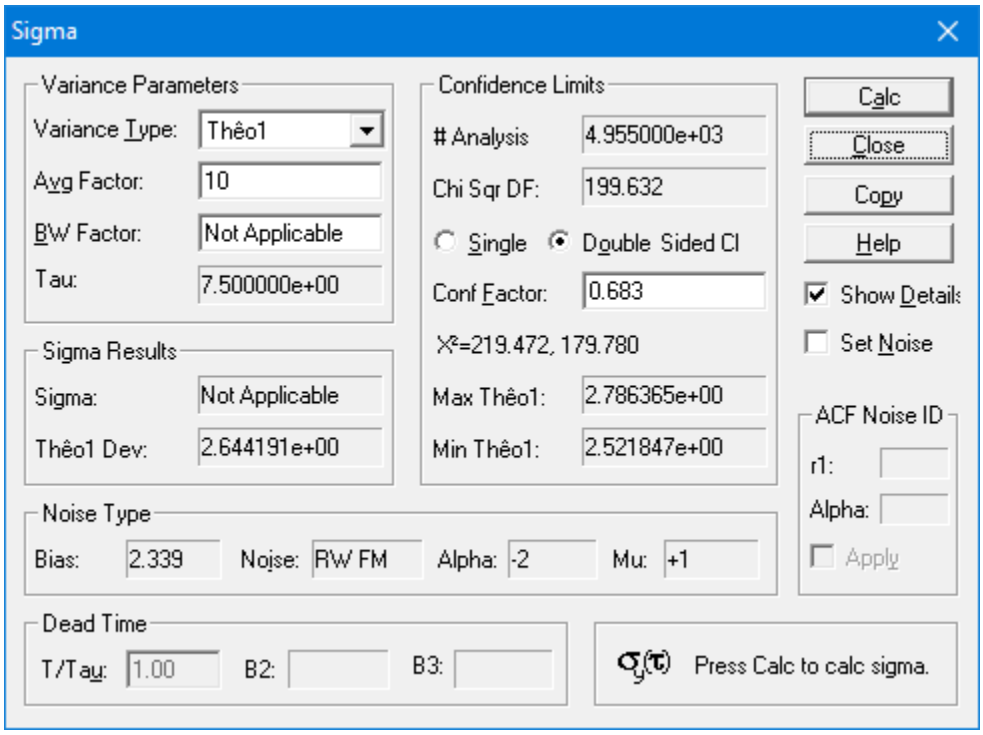


The Stable32 ThêoBR Run and Plot produce slightly different results because of a different bias factor.

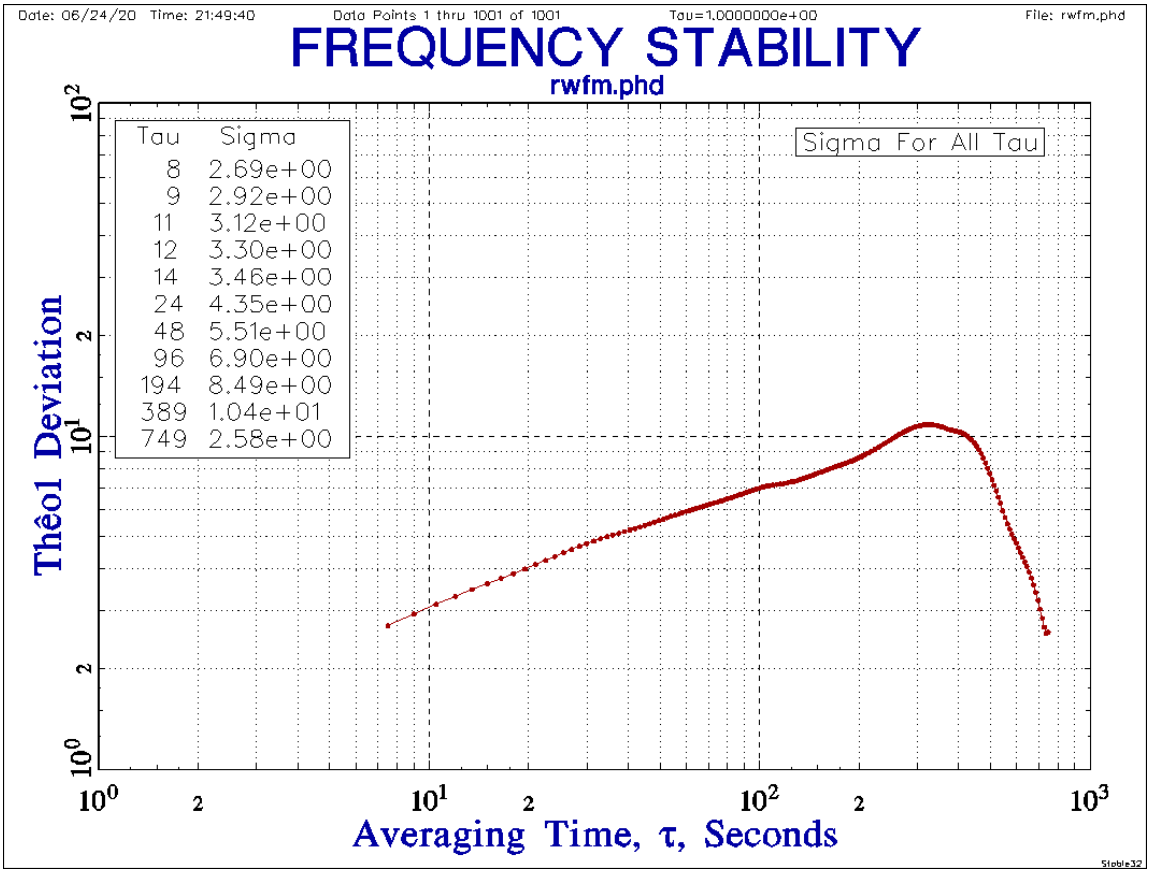


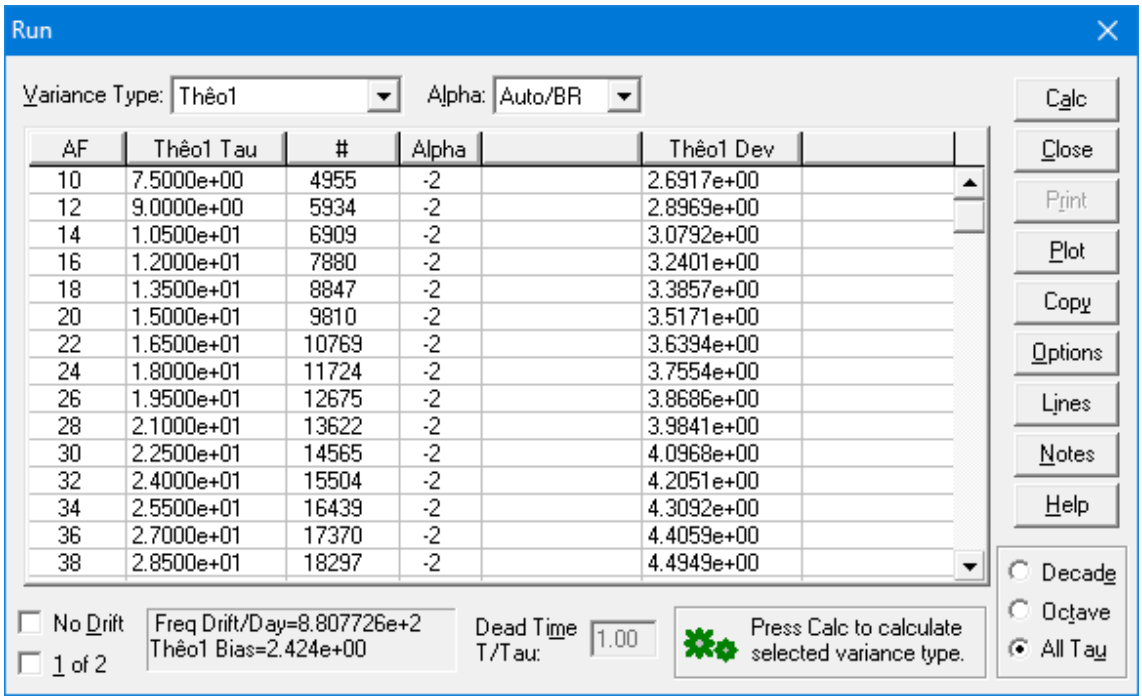

```
> r<-trun(rwfm)
[1] 2.339171
[1] -2
```

	tau	l	t	u
1	1.5	1.496699	1.535814	1.578162
2	3.0	1.829164	1.878034	1.931041
3	4.5	2.099929	2.160276	2.226141
4	6.0	2.343586	2.416342	2.496320
5	7.5	2.558800	2.644191	2.738738
6	9.0	2.747783	2.845712	2.954909
7	10.5	2.914554	3.024795	3.148563
8	12.0	3.060710	3.182905	3.320998
9	13.5	3.192064	3.325940	3.478196
10	15.0	3.309724	3.454953	3.621132



The ThêoBR and Stable32 Sigma functions produce the same nominal Thêo1 and bias factor for the RW FM noise, nearly the same 1-sigma error bars.

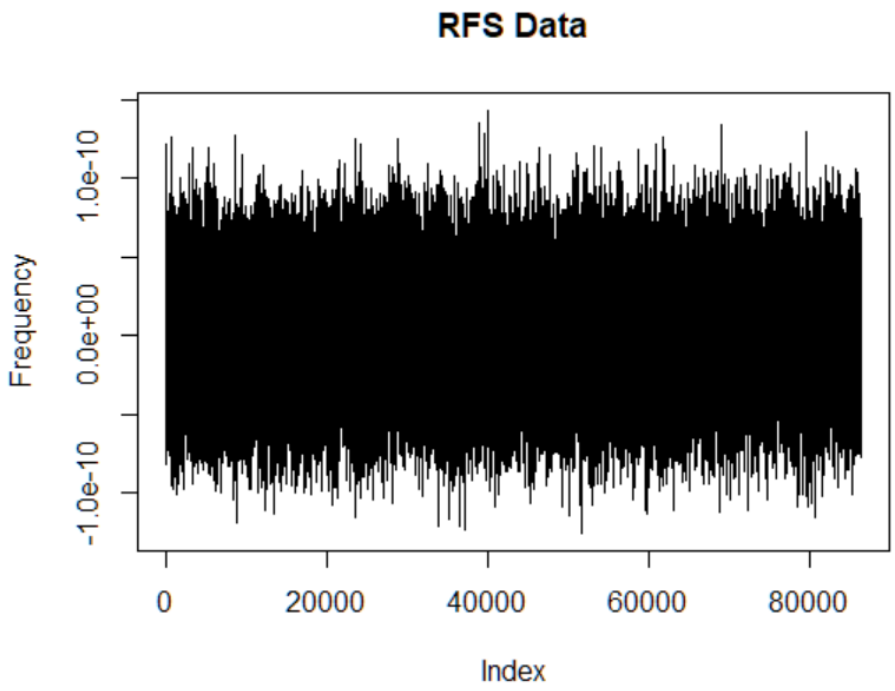


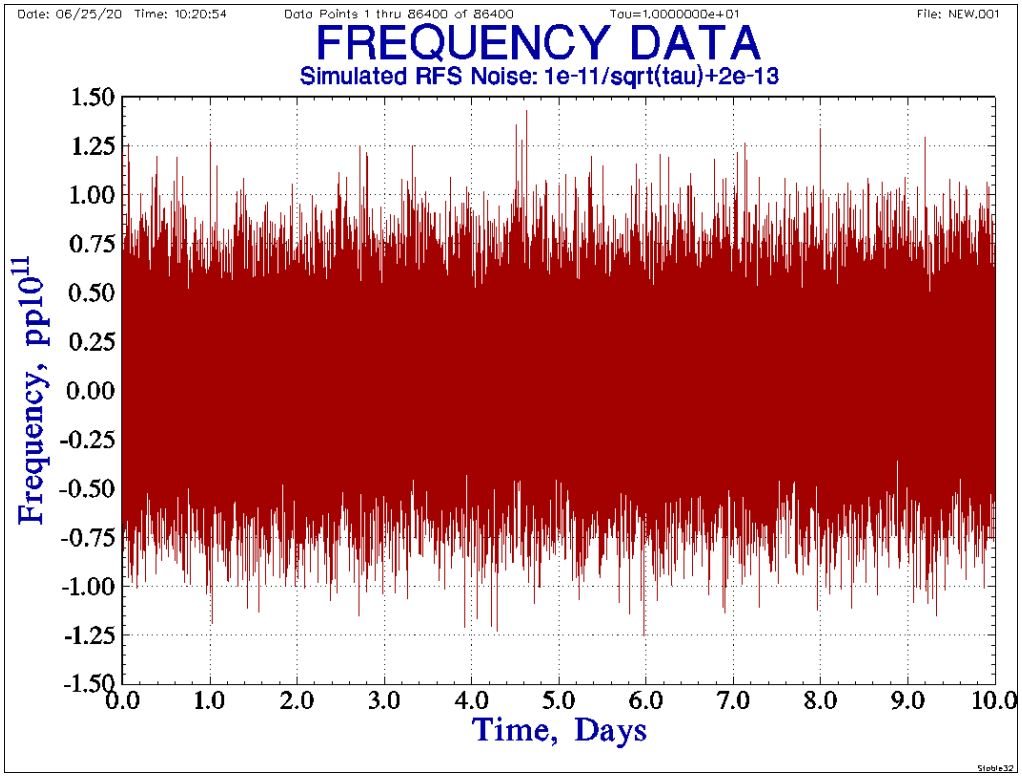


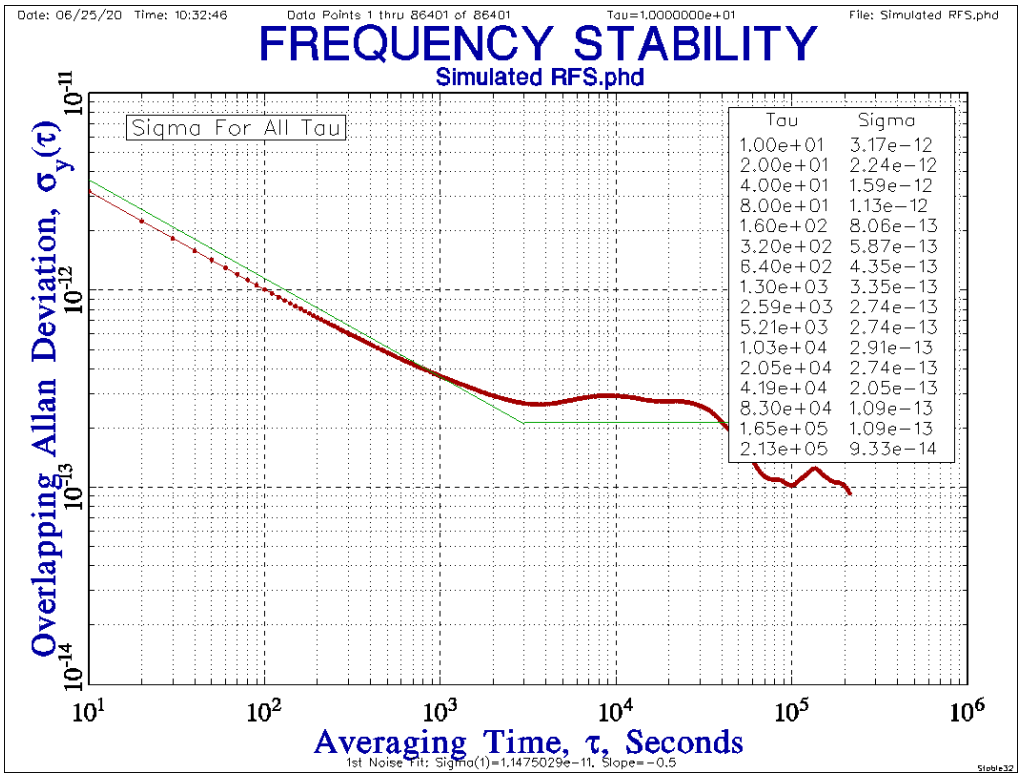
The Stable32 ThêoBR Run and Plot produce slightly different results because of a different bias factor.

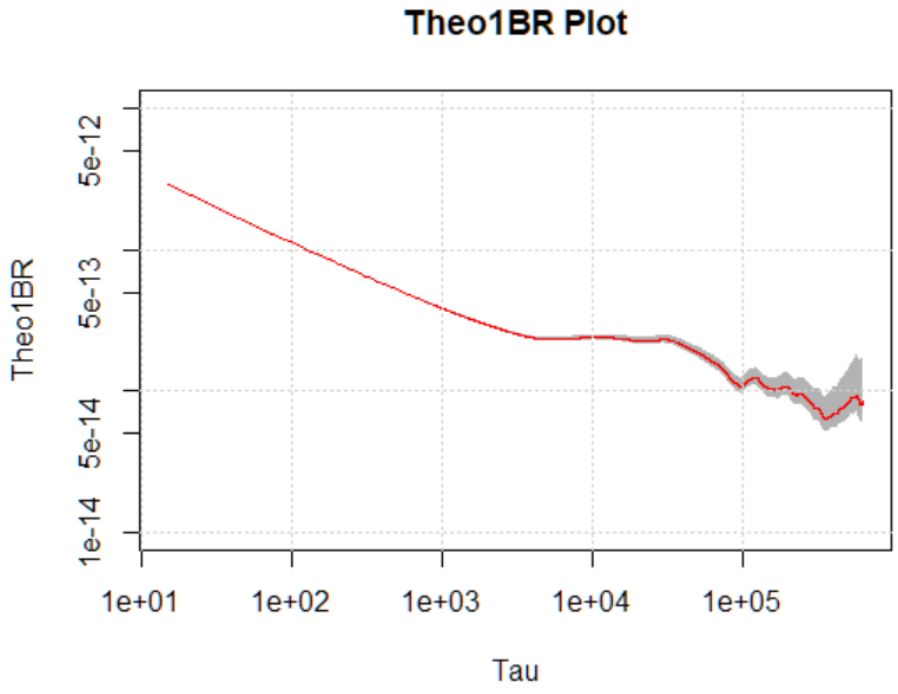
Simulated Rubidium Frequency Standard Noise

10 Days (86,400 points) of 10-second W and F FM data:
$\sigma_y(\tau)=1x10^{-11}\tau^{-1/2}+2x10^{-13}$ No Drift

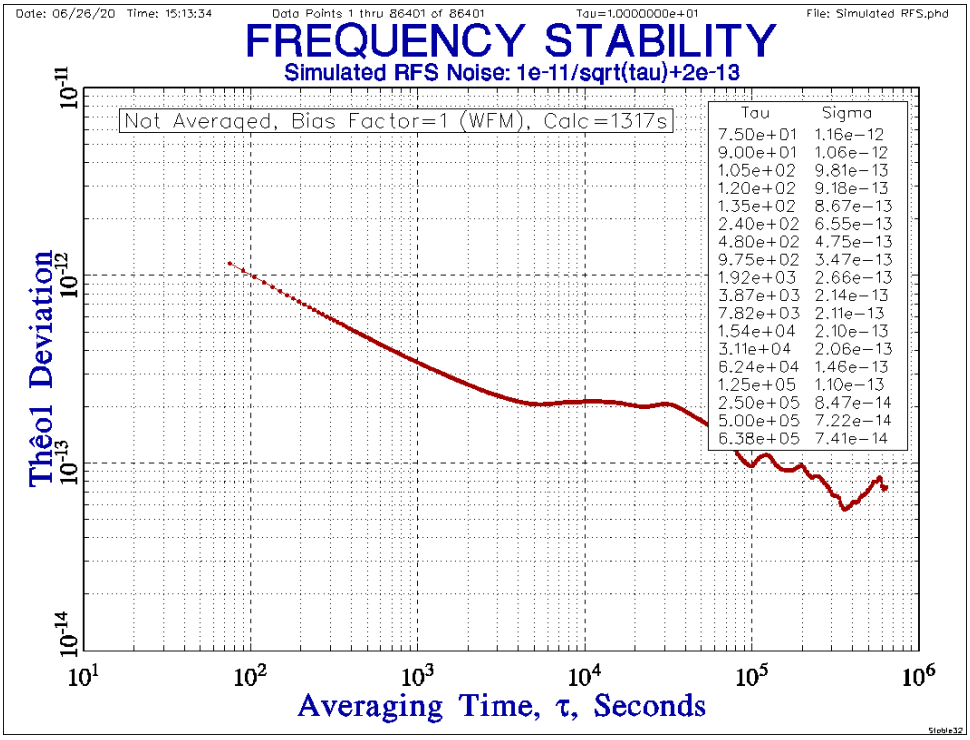




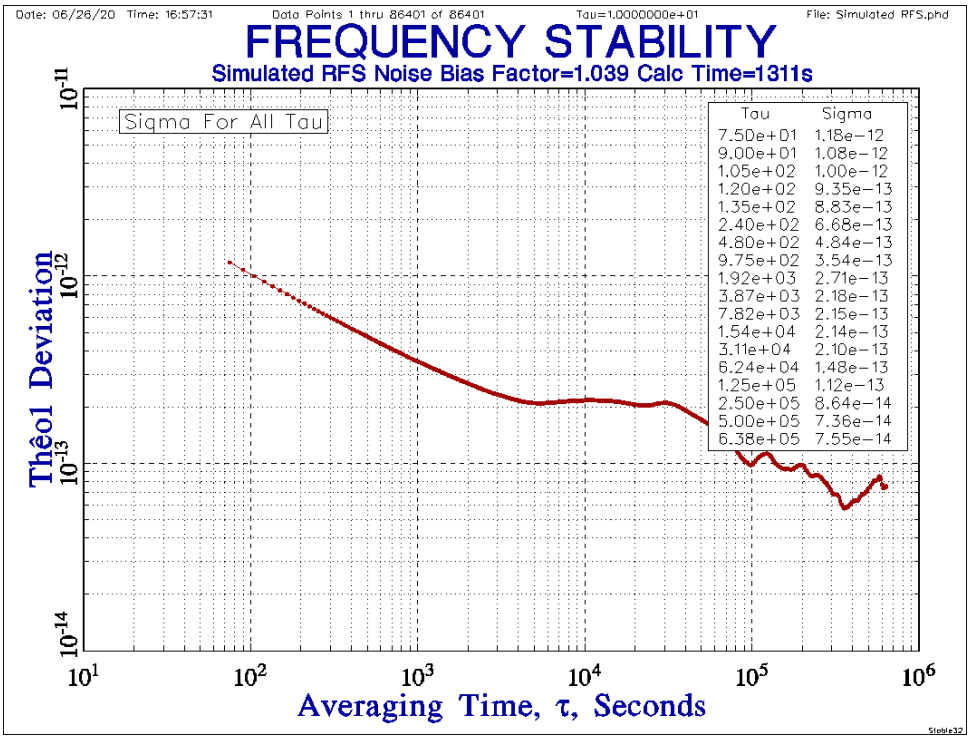




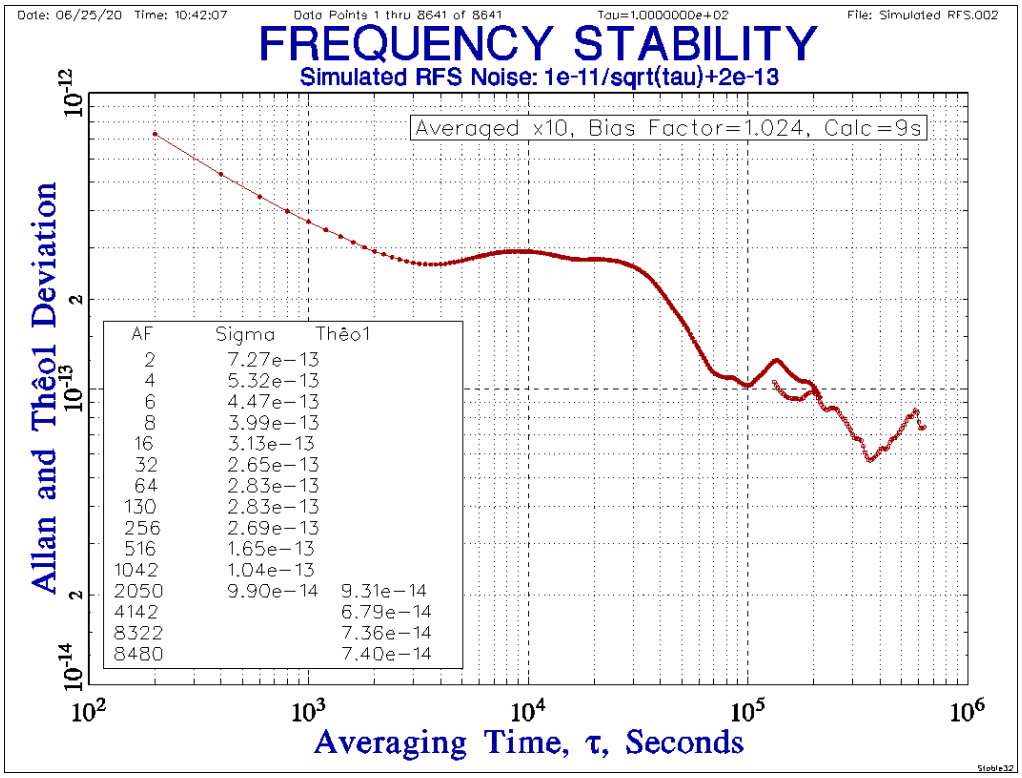

```
> trun(rb,10)
[1] 1.250157
[1] 0
```

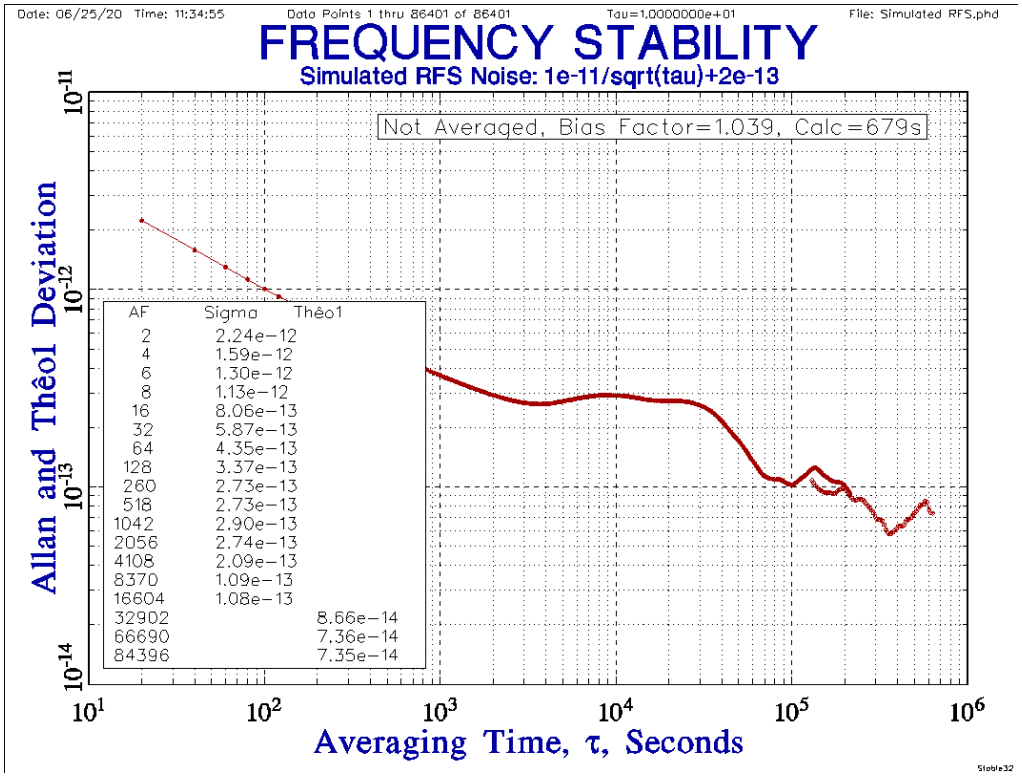


This basic 86,400 point All Tau Thêo1 calculation (no bias removal or error bars) in R using the Lewis fast algorithm takes about 5 seconds, while the same calculation using the naïve method in Stable32 takes 1317 seconds (22 minutes), about x260 times longer. An All Tau Thêo1 calculation takes longer than a ThêoH calculation because most of the points are ADEV values in the latter.

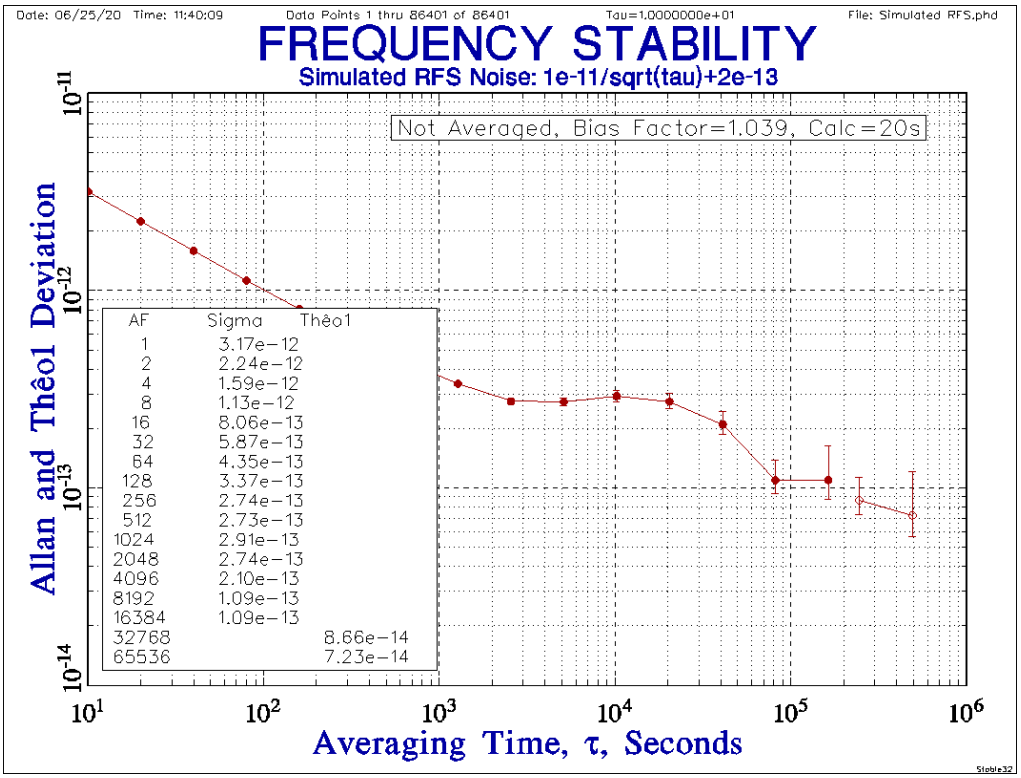


The otherwise same ThêoBR calculation and plot is nearly identical, with a Bias Factor of 1.039 (α=0, W FM noise) and calculation time (1311s), and, for practical purposes, the same as the R results.





The 86,400-point ADEV calculation is practically instantaneous, the 8640-point x10 averaged ThêoH calculation takes a brief 9 seconds, and the un-averaged ThêoH calculation is calculation takes a bit over 11 minutes. The averaging does not affect the longer tau ThêoH results, which extend to about 600,000 seconds compared with about 200,000 seconds for ADEV.



Performing the un-averaged ThêoH run in Stable32 at octave tau increments takes only 20 seconds.

As a final example, this plot shows the results of a 1 million point, τ=1s, $\sigma_y(1)$=1.0 ThêoBR run under R. The simulated white FM phase noise was generated in Stable32, and it took about 40 minutes to process, representing the practical size limit to this approach, one that would be totally impractical with the naïve algorithm. The plot shows that the instability falls somewhat faster that $\tau^{-1/2}$, and the analysis identifies it as closest to F PM noise.

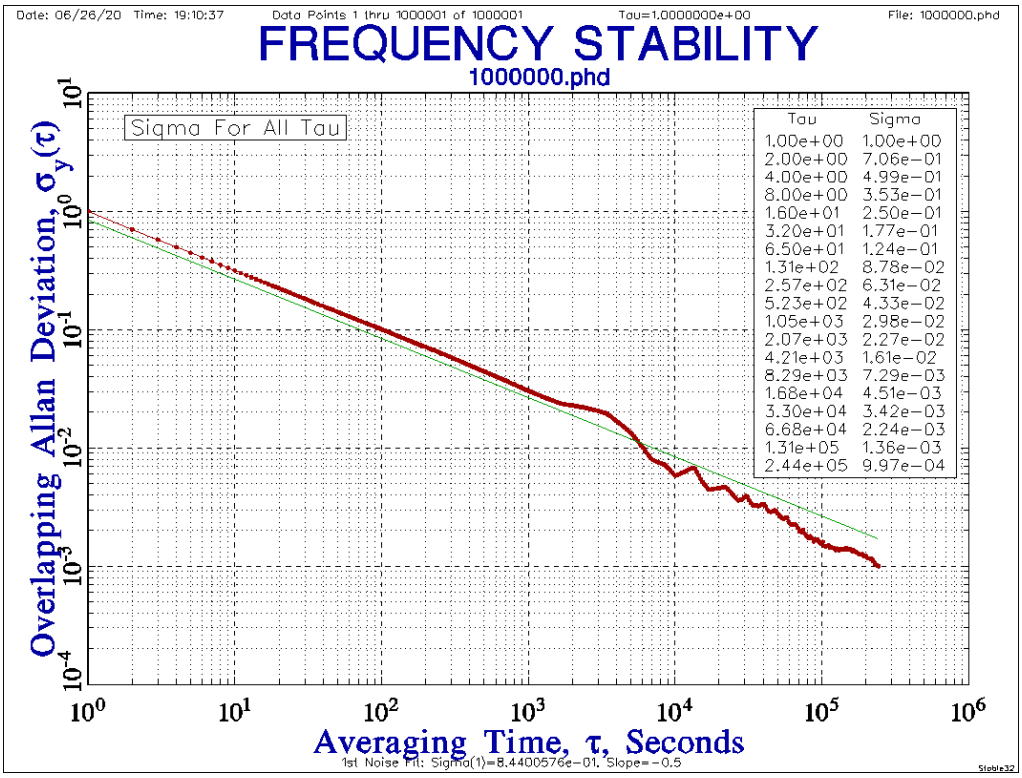

```
> trun(big)
[1] 0.7210418
[1] 1
```

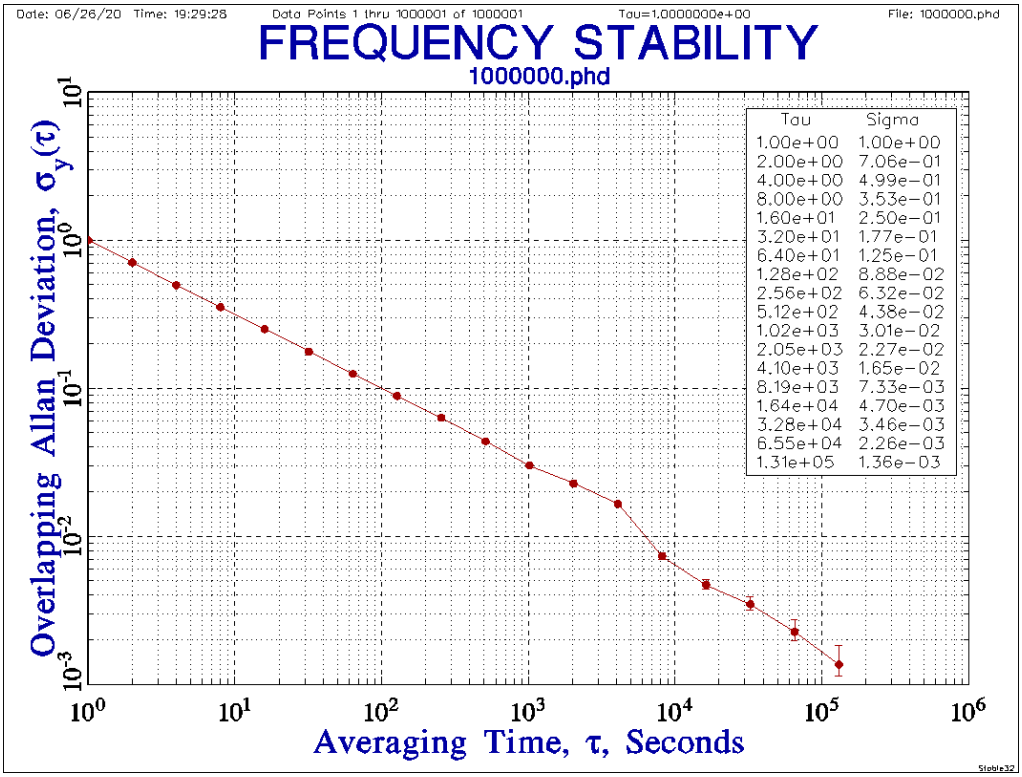
The corresponding overlapping Allan deviation plot, shown below, is, of course, similar but covers a smaller tau range. It takes only about 7 seconds to compute in Stable32, so would be the analysis method of choice unless results were needed at large tau. If they were, the fast ThêoBR analysis is quite practical, especially compared with extending the data run.

The ADEV slopes show the noise to be W FM in the short term, becoming close to F FM and then F PM in the medium term, and tending toward F FM in the long term.

Thêo1 is known to be less prone to having oscillatory variations, but those are about the same as for ADEV in this run, so they are probably artifacts in the simulated data.

The same 1-sigma error bars in the octave ADEV plot are corresponding larger than those for the ThêoBR run, especially since those are for F FM rather than W FM noise.


Date: 06/26/20 Time: 19:10:37
Data Points 1 thru 1000001 of 1000001
Tau=1.0000000e+00
File: 1000000.phd
FREQUENCY STABILITY
1000000.phd
Sigma For All Tau
Tau Sigma
1.00e+00 1.00e+00
2.00e+00 7.06e-01
4.00e+00 4.99e-01
8.00e+00 3.53e-01
1.60e+01 2.50e-01
3.20e+01 1.77e-01
6.50e+01 1.24e-01
1.31e+02 8.78e-02
2.57e+02 6.31e-02
5.23e+02 4.33e-02
1.05e+03 2.98e-02
2.07e+03 2.27e-02
4.21e+03 1.61e-02
8.29e+03 7.29e-03
1.68e+04 4.51e-03
3.30e+04 3.42e-03
6.68e+04 2.24e-03
1.31e+05 1.36e-03
2.44e+05 9.97e-04
Overlapping Allan Deviation, σy(τ)
Averaging Time, τ, Seconds
1st Noise Fit: Sigma(1)=8.4400576e-01, Slope=-0.5
Stable32


Date: 06/26/20 Time: 19:29:28
Data Points 1 thru 1000001 of 1000001
Tau=1.0000000e+00
File: 1000000.phd
FREQUENCY STABILITY
1000000.phd
Tau Sigma
1.00e+00 1.00e+00
2.00e+00 7.06e-01
4.00e+00 4.99e-01
8.00e+00 3.53e-01
1.60e+01 2.50e-01
3.20e+01 1.77e-01
6.40e+01 1.25e-01
1.28e+02 8.88e-02
2.56e+02 6.32e-02
5.12e+02 4.38e-02
1.02e+03 3.01e-02
2.05e+03 2.27e-02
4.10e+03 1.65e-02
8.19e+03 7.33e-03
1.64e+04 4.70e-03
3.28e+04 3.46e-03
6.55e+04 2.26e-03
1.31e+05 1.36e-03
Overlapping Allan Deviation, σy(τ)
Averaging Time, τ, Seconds
Stable32