Correlation Statistics of Gaussian Noise

Jason Castiglione, Dr. Sergei Klimenko, and Dr. Guenakh Mitselmakher
Department of Physics, University of Florida

Abstract

In order to detect correlation of Gaussian signals, linear, rank, and sign correlation are developed. Magnitude of correlation coefficients, significance, and two-sided significance are computed and compared for each test. Programs are written as macro files, in c++, that are run in ROOT. Efficiencies of correlation tests are calculated resulting in a relative length equation, and methods of accurately relating results from different tests. Wavelet Analysis is mildly introduced as a method of localizing correlation in frequency bands.

Table of Contents

1.Introduction

Gravitational wave detection is an exacting science and unforgiving in calculations. Precision is fundamental and requires analyzing data with supreme accuracy. Presently in the United States, the Laser Interferometer Gravitational-Wave Observatory (LIGO) is at the forefront of gravitational wave detection. There are also laser interferometers entitled VIRGO (French & Italian), TAMA (Japan), and GEO (German & British) spread throughout the world. There will be several gravitational wave interferometers active in the United States. Two of these interferometers are located in Washington, and one is located in Louisiana. These locations are separated by approximately 3000 km, a distance that will allow any detected signal to be verified by at least three detectors. This method will help prevent local noise from being misinterpreted as a gravitational wave. It is readily understood the ease at which one can extract false results when the magnitude of gravitational waves is taken into account.

The gravitational force is the weakest of fundamental forces, and can be compared to the electrical force for an idea of this weakness. If one compares these forces for two protons, the gravitational force is approximately 10^{-36} times smaller than the electrical force[1]. Now that the relative magnitude of the gravitational force can be approximated, it is beneficial to calculate the change in path length that is anticipated from the interferometers with a path length of 4km. To give an approximate idea of the magnitude of the space-time contraction that LIGO will need to detect, it is possible to use a very basic equation:

$$\Delta L = h \times L \tag{1}$$

In Eq. (1), h is the fractional change in length, or strain, and L is equal to the length of the path to be measured. When h and L are multiplied they result in the length contraction that can be detected [1]. The fractional change in length, h, is calculated by Eq. (2):

[1] It is also noted that this method is comparing static forces and not radiation.

$$h = \frac{G\ddot{Q}}{c^4 D} \tag{2}$$

In Eq. (2)[1], h is computed by using the quadrupole moment, $\ddot{Q}$. The quadrupole moment is best understood as the deviation of the gravitational force from a spherical symmetry and is calculated by expanding the mass distribution into multipole moments. It is noted in *Gravitation* [2], that there is no mass dipole gravitational radiation. This condition results from the second derivative of the mass dipole moment equaling zero, because of the law of conservation of momentum. D is the distance from the source of the gravitational waves. The fractional length change, h, can only be calculated with numerous approximations. Nominally, one can use the Virgo cluster as the location of a source that is approximately a distance of 21 Mpc[2] with an error of 1 Mpc [6]. The most likely sources will be rotating black holes, coalescing binary stars, or gravitational background radiation. The energy expected from one of these sources is on the order of one solar mass. The resulting path change is expected to be less then 10^{-17} m [1].

The minute perturbances that must be detected from enormous data sets necessitates unbelievably accurate and robust statistical computing procedures. Some of the main software needs are data compression, time series analysis, and correlation methods. LIGO plans to receive data at a rate of 15 MB/s with a total yield of one petabyte per year. Data compression software is needed to store this enormous amount of data in the least space possible. Time series analysis will use wavelet analysis to localize frequency bursts and Fourier transforms to analyze the frequency content of the signal. Correlation software will be used to verify results among other locations. The correlation software will consist of parametric and non-parametric methods of correlation procedures. This software will also be

[2] 1 Mpc= 3.12x 10^{22} meters

used on the power supplies to isolate noise from 60 Hz lines, and harmonics, to calculate any trends in power fluctuation, and for correlation analysis of any detected signals.

This project develops correlation testing for Gaussian signals and noise. I will describe the characteristics of results that will be expected for detection of patterns in LIGO data. Three different correlation tests will be compared, with an ultimate goal of finding the most efficient test. Wavelet analysis will also be developed mildly in order to localize correlation in the wavelet domain. All the programming and data simulation will be performed in ROOT. ROOT is an object oriented c++ program and contains many classes with useful data processing tools, and a command line c++ interpreter.

The main software for correlation testing was written over the summer and is mainly derived from *Numerical Recipes* [3]. Programs for wavelet analysis are provided by the Wavelet Analysis Tool (WAT)[3]. I have written programs in ROOT using c++ that serve several functions. The main program is entitled **mira.C** and computes linear, rank, and sign statistics. To properly test **mira.C**, the program **aver.C** was written. This program simulates Gaussian signals embedded in Gaussian Noise and does 200 averages for every signal to noise ratio (SNR)[4] tested. An additional program, **wtor.C**, was written to facilitate testing for correlation in different frequency ranges. This program decomposes signals using wavelet transforms. Upon decomposition, **wtor.C** performs correlation testing for each scale, or frequency layer. This form of correlation testing will prove to be extremely important for localizing correlation.

[3] WAT is being developed at the University of Florida by Dr. Klimenko and Dr. Sazonov.

2.Correlation Tests

2.1 Purpose

A prime concern in calculating the correlation between two data sets is statistical accuracy. This concern necessitates the use of proven techniques and the ability to classify their relative errors and effectiveness. Rank, linear, and sign correlation testing are developed in this project, along with the calculation of various statistical properties. When correlation testing is performed, three results for each method are produced. These results are correlation coefficients, significance, and double-sided significance. Before discussing individual tests, the characteristics of these results will be described.

Correlation coefficients reveal the degree of correlation or anti-correlation between two data sets. They are valued from -1 to 1. A positive value indicates correlation, with 1 representing complete correlation, while a value close to zero indicates no correlation. Negative values represent anti-correlation, and -1 is indicative of complete anti-correlation. Correlation coefficients are not very useful unless a significance level is associated with them.

Significance determines the confidence in which one might have over resulting coefficients. The significance ranges from zero to ∞. Values that are under two are questionable, and anything over three is nominally a valuable correlation. The significance is calculated for correlation coefficients by Eq.(3):

$$t = r\sqrt{\frac{N-2}{1-r^2}} \tag{3}$$

In Eq. (3), N is the length of the data set, and r is the correlation coefficient. Eq. (3) is only accurate for rank and sign correlation, when N is large, greater than 100. Eq. (3) is only valid for linear correlation when N is large and the distribution is Gaussian. Upon viewing this equation, the

[4] SNR is calculated for Gaussian Distributions as the ratio of signal root mean square (RMS) to noise RMS.

dependence of the significance on the data set length is readily apparent. This is very important because in some cases it is possible to control the length of the data sets.

Another important measure of significance is the two-sided significance level. This test helps confirm significance. For this test, values which are exceedingly close to zero are representative of a powerful correlation coefficient. Two-sided significance level (SL) is calculated for a given correlation coefficient with a known probability distribution function(PDF) by Eq.(4):

$$SL = \int_{r}^{\infty} f(x)dx + \int_{-\infty}^{-r} f(x)dx \tag{4}$$

In Eq.(4), r is the correlation coefficient, and $f(x)$ is the PDF of the r statistic variable.

2.2 Linear Correlation Coefficients

The linear correlation coefficient (LCC) is widely used to measure relations between two data sets with known PDF's. Linear correlation is classified as a parametric test, meaning that it is dependent on the characteristics of the data sets being analyzed. The LCC is calculated by Eq. (5):

$$r = \frac{\sum_i (x_i - \bar{x}) \times (y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2} \times \sqrt{(y_i - \bar{y})^2}} \tag{5}$$

In this equation, the difference of each data member, x_i *or* y_i, from the mean, $\bar{x}$ *or* $\bar{y}$, of the data set is summed, and then multiplied by the same sum of differences from the other data set. These results are then normalized.

2.3 Rank Correlation Coefficients

Rank correlation (RC) is a very effective method of detecting similarities between data sets, because of the fact that it is a non-parametric test. A non-parametric test is defined as one that is

independent from the characteristics of the data. The test is based on assigning every data member an integer ranking, R_i or S_i, amongst the others, and then inputting these ranks into Eq. (6):

$$r = \frac{\sum_i (R_i - \overline{R}) \times (S_i - \overline{S})}{\sqrt{\sum_i (R_i - \overline{R})} \sqrt{\sum_i (S_i - \overline{S})}} \tag{6}$$

Eq. (6) is the same as Eq. (5) except the ranks, and the mean of the resulting list of ranks are inputted in this formula. The mean, $\overline{R}$ or $\overline{S}$, of the ranks is obviously calculated by (N+1)/2. A minor disadvantage of using RC is the loss of information about the original data set, although the advantages of using RC greatly outweigh the disadvantages. RC for a whole produces correlation coefficients that are 95% of LCC's, and the resulting significances posses the same ratio. An important fact about using RC is that the distribution of the numbers is always known, which is due to drawing the numbers from a distribution of 1 to N. When the distribution is known, one will have a firmer foundation to select between the null hypothesis (data sets are correlated) and the alternative hypothesis (data sets are not correlated).

2.4 Sign Correlation Coefficients

Sign correlation (SC) is classified as robust, which is defined as "insensitive to small departures from the idealized assumptions for which the estimator is optimized" [3]. SC is considered parametric, because the weak dependence on one parameter, the median. The SC coefficient (SCC) is calculated in three steps. The first step requires choosing a median, and then converting each number in a data set to either 1 or -1 depending upon whether that number is greater or lesser than the median. After step one, every member of one data set, S_i, is multiplied by the corresponding member, T_i, of the other data set. The final step, sums the array of 1's and -1's, from the last step, and divides the resulting number by the length of the data sets, N. This is represented by Eq. (7):

$$r = \frac{1}{N} \sum_i S_i \times T_i \tag{7}$$

SC is the least sensitive test to correlation between data sets, because of the information lost when performing the test. When studied in direct comparison to the other tests, it is possible to predict how SC will act. The magnitude of the SCC, is predominantly 64% of the magnitude of the LCC, and this percentage directly applies to the significance. Although sc results in lower coefficients, it is desirable, because the relative ease at which it can be computed.

2.5 Wavelet Analysis

Previously, methods of computing correlation coefficients were presented. The amount of information that can be inferred from a correlation test on time series data can be greatly enhanced when the data are decomposed using orthogonal functions. This decomposition is entitled a wavelet transform, and the most basic transform is a Haar Transform(HT). For the scope of this paper, the HT will be performed and explained using lifting steps[4].

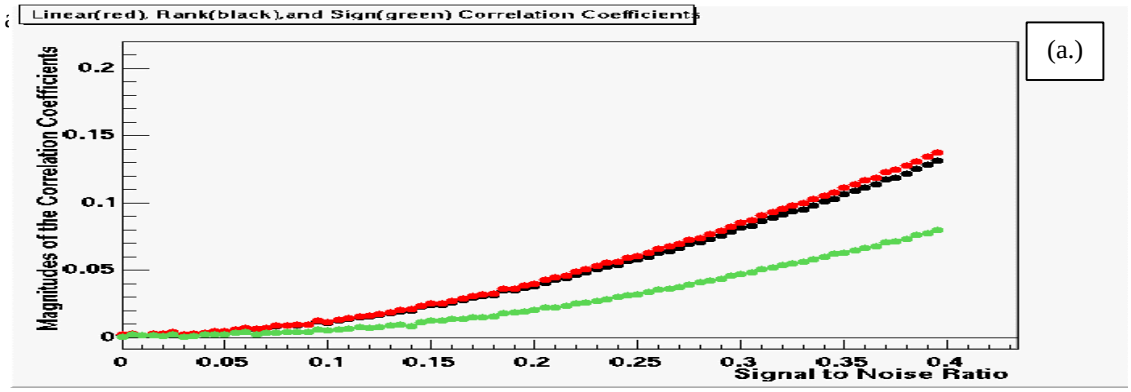
The three main steps of the HT are split, predict, and update. The split stage consists of separating a data set in two, by placing odd-indexed samples in one set and placing even-indexed samples into another set. The predict step consists of predicting the odd samples by using the even samples. There are many methods of prediction, like interpolation and splines. The HT uses the even samples without altering them. Next, the accuracy of the prediction is calculated by subtracting the even samples from the odd resulting in an array of detail coefficients. The update step averages corresponding even and odd indexed samples, resulting in an array of smooth coefficients. These three steps make up the first level of a HT. The second level consists of repeating the same three steps on the smooth coefficients from the previous HT. The number of decompositions that can be performed depends on the dyadic length of the signal. A signal of length 2^j can be represented by j levels. High frequency information is stored in the beginning levels. As the levels increase, the information pertains to lower frequencies.

During the summer, I performed wavelet analysis on data sets that consisted of 65536 samples of random Gaussian signals and noise. I decomposed the data sets for 12 levels and then calculated

correlation statistics for each level. SNR was varied from 0.05 to 0.5. Increased correlation in any wavelet level was minute, and could be attributed to statistical fluctuations. Lack of varying correlation further verified the effectiveness of the random number generators in use. In addition to analyzing Gaussian signals, sine signals were added to the Gaussian noise to effectively map out responsiveness of wavelet levels for frequency response. Sine signals added to Gaussian noise resulted in correlation coefficients that were approximately four times greater in the corresponding wavelet level.

2.6 Results of Correlation Tests

The correlation tests presented in this paper have been tested extensively with Gaussian signals and noise. These testing procedures resulted in certain characteristics of each correlation procedure. The results aforementioned were derived from performing simulations in ROOT. The main simulation program is **aver.C**. **Aver.C** repeatedly does correlation testing for different SNR's. At each SNR, it performed 200 averages to account for any statistical fluctuations. Tests were performed from 0.005 to 0.4 SNR. Fig. (1) presents results for a data length of 16384.



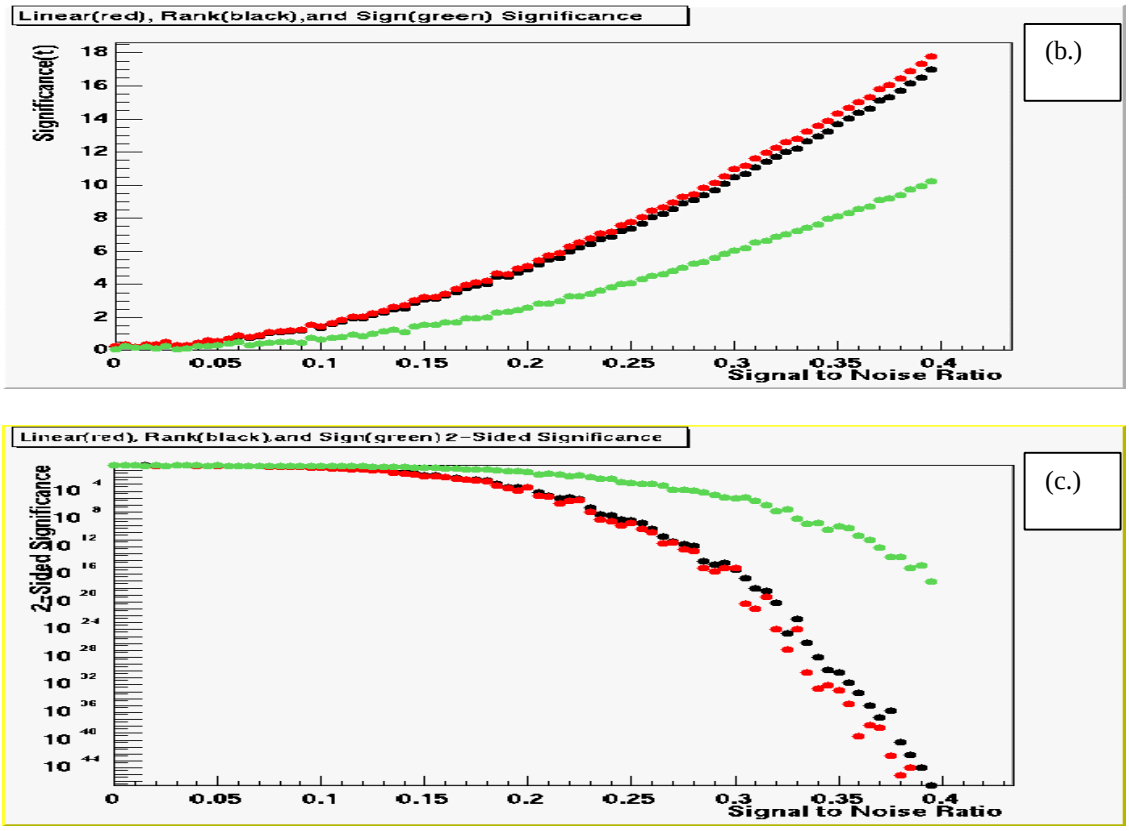


FIGURE 1 *Correlation Statistics as a function of SNR for RC(black), SC(green), and LC(red)*

Aver.C also includes a routine for correlation testing at various data length sizes while holding the SNR constant. This routine is entitled **naver**, and performs 200 averages for each data set length from 1,000 to 15,000. Fig. (2) contains the graphs for these results at a SNR of 0.2. **Naver** produces some interesting results. Fig.(2a) clearly shows increasing data lengths has no effect on the correlation coefficient, as long as the signal is present in additional data points. Figs.(2b&2c) clearly show positive effects of data lengthening, which are elevating significance in the correlation coefficients.

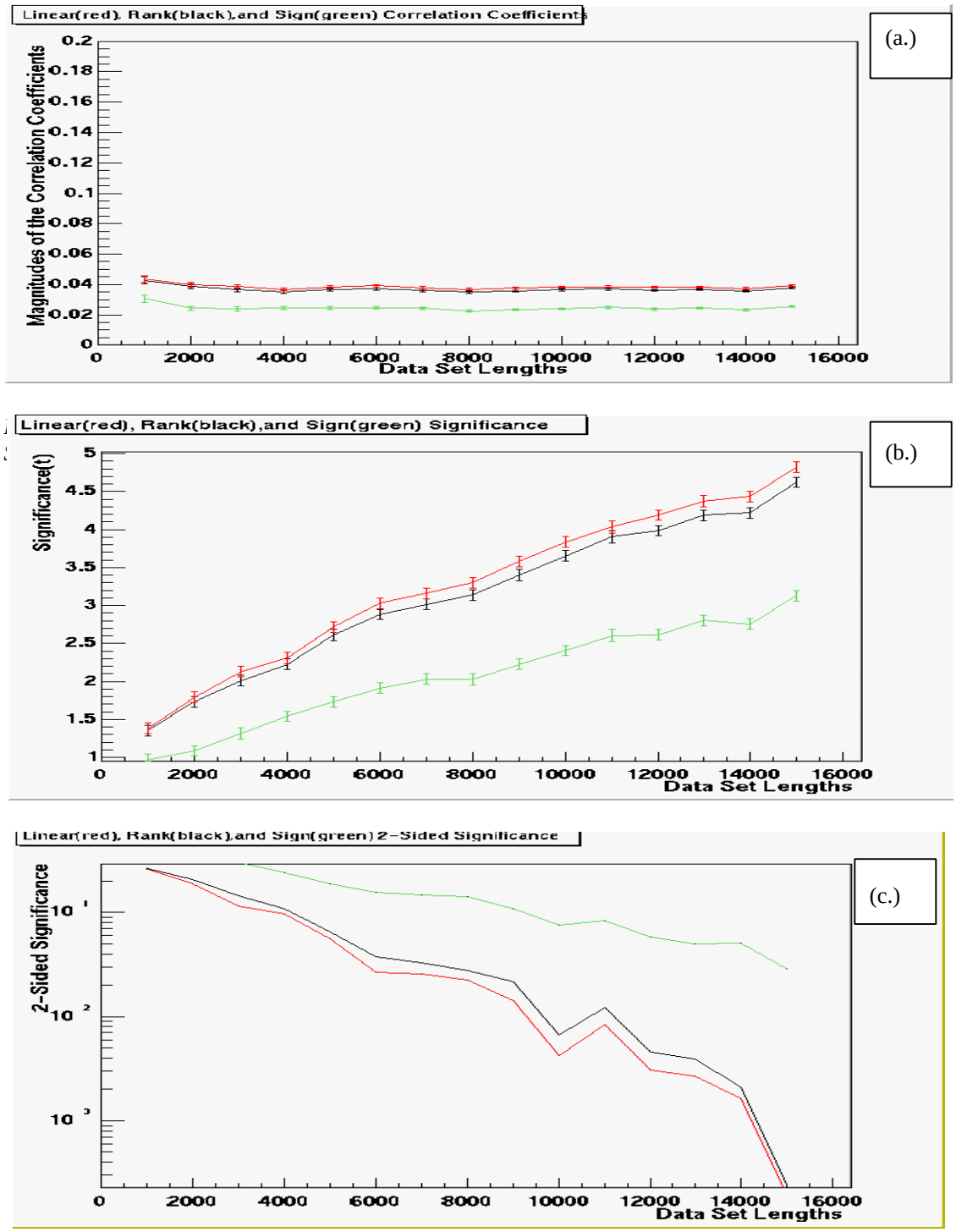


FIGURE 2 *LC(red), RC(black), and SC(green) correlation statistics with a constant SNR=0.2*

A noticeable effect in the preceding results is consistent lagging of the SC and RC to the LC. LC predominantly will produce results with higher correlation statistics in comparison to SC and RC.

The interesting relation is that each correlation test will repeatedly perform the same in comparison to the others. Fig. (3) presents this relation for correlation coefficients comparing SC and RC to LC.

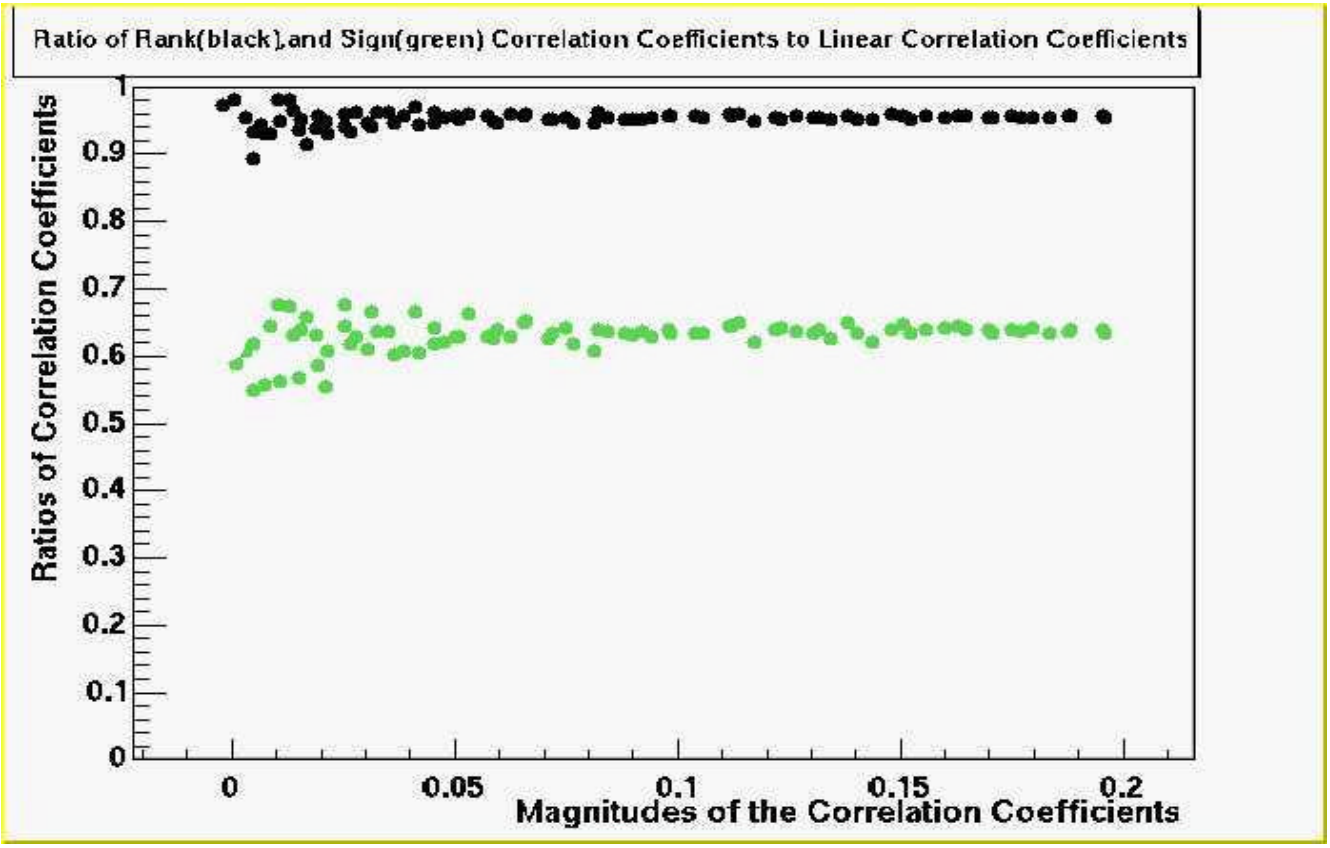


FIGURE 3 *Ratios of SCC (green) and RCC (black) to LCC*

Fig. (3) shows important information about relating correlation tests. If correlation tests perform consistently in the same manner, it is possible to formulate an equation that will allow one to transform results between tests. This equation shall be entitled the relative length equation (RLE), and it will calculate the ratio of the data set lengths for two different tests that will produce the same significance. Eq. (8) will only work if the correlation tests provide coefficients that are a constant percentage of each other, and the signal is present in the additional data.

$$\frac{N'}{N} = \frac{\frac{T^2}{p^2 \times r^2} - T^2 + 2}{\frac{T^2}{r^2} - T^2 + 2} \approx \frac{1}{p^2} \tag{8}$$

In Eq. (8), *p* is the ratio of correlation coefficients, *r* is the coefficient in the denominator of the ratio, and *T* is the significance. The approximation is only accurate for a significance greater than 2. Fig. (4) plots Eq.(8) for significance levels from 0-10 in 0.5 increments. Fig. (4) compares sign and linear correlation. As coefficients approach one, the ratio of data sets increases rapidly. Fig. (4)

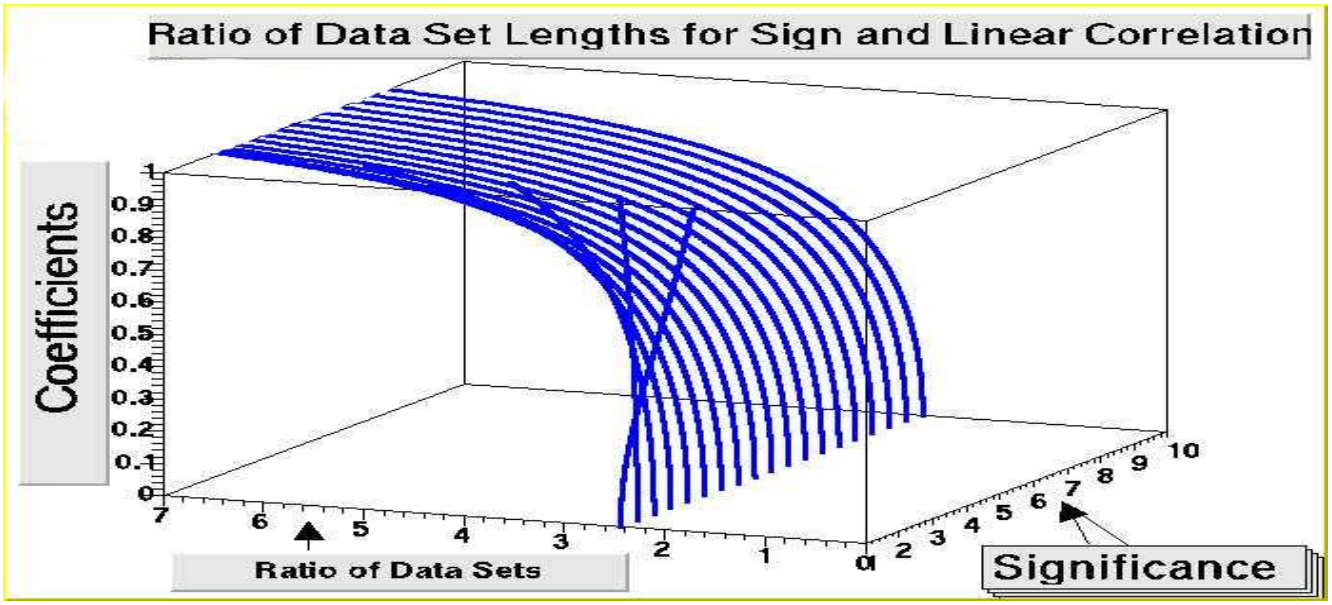


FIGURE 4 *Relationship between SC and LC Data Set Length for varying significance and coefficients.*

shows that setting 3 as the cutoff for significance levels, SC will require processing anywhere from 2.4 to 7 times the amount of data as LC. RC exhibits the same graph when compared to LC, except it is shifted to approximately 1.1 instead of 2.4. The lengthening of data sets will definitely be a possibility in enhancing less powerful correlation tests, although it will not be successful for all signals. In order for data set lengthening to be effective, the signal must be present throughout the whole data set. If the gravitational wave signal is localized in 1 sec of data, and the signal's length is increased to 2 sec of data, the opposite effect will happen, and results will prove to be unsatisfactory.

The correlation tests in this paper have been extensively tested with Gaussian signals and noise. The analysis of data has been presented, and now we shall classify benefits of each test in table **I.** Table **I** compares each test according to type, data type, computational efficiency, efficiency, and the ratio of data set lengths needed to produce the same significance. Type refers to whether or not the test

is dependent on any variables. Data type addresses the dependence of some tests on the distribution of the data. Computational efficiency describes the complexity of the computer program for calculating the statistics of each correlation test. Efficiency references the ratio of significance in comparing the results of each test. The last column displays the numbers necessary to multiply the data set length by to achieve similar significances.

Correlation Test	Test Type	Data Type	Computational Efficiency	Efficiency	$\frac{N'}{N}$ T=2, r=0.2
Linear	Parametric	Known Distribution	Medium	1	1
Rank	Non-Parametric	Any	Slow	95%	1.11
Sign	Robust	Any with known median	Fast	64%	2.47

TABLE I. *Comparison of Correlation Tests*

Linear correlation(LC) has one major disadvantage. When finding correlation between data sets with LC, it is necessary to know before hand the PDF of the data. In the context of this report, all correlation testing was done with Gaussian distributions. When this is the case, LC will perform the best. The computational complexity of LC is described as medium, because of the ease at which the SCC is computed. All other tests are compared to LC, because LC will output higher significance, correlation coefficients with a higher magnitude, and will require shorter data lengths for detection of the same correlation.

RC's greatest disadvantage is the complexity of the program. RC requires a sorting algorithm[5] to rank data, and must perform the sort twice for each test. Presently, this does not pose a threat because the program will compute the three correlation statistics in approximately one second for data lengths of 100,000. RC is definitely the most flexible test, and requires no knowledge about the PDF of the data being analyzed. In comparison to LC, RC results in coefficients, and significance

levels that are approximately 95% of LC results. The data set length can be lengthened by 1.11 times the LC data set length to achieve the same results.

In comparison, SC is the least sensitive correlation test. This is because a great deal of information is lost about the original data sets. Two positive aspects of SC are the ease at which it can be computed and its independence from the distribution of the original data set. The only necessary parameter to compute SC is the median of the data sets. SC consistently produces coefficients and significances that are 65% of LC results. SC will achieve the same results when the data sets are lengthened by approximately 2.4 times the length of the LC data sets.

3.Summary

Gravitational wave detection has gone through many incarnations since the prediction of gravitational radiation in the general theory of relativity. Several methods of detection have been explored and have proven too insensitive to be fruitful. Presently, LIGO is on the borderline of whether or not it will meet the criteria for detecting gravitational waves. I am confident that with the equipment and extraordinary scientists presently working on all aspects of LIGO that if gravitational waves are not detected, there will be a solid foundation for further gravitational interferometers.

One of the many important aspects of LIGO that will be a determinant in LIGO's success is data analysis. In the scope of this paper, methods of correlation analysis were developed for Gaussian signals. These methods were compared and relations were developed to properly transform results between correlation tests. Sign correlation will prove to be a valuable method in correlation testing of raw data due to its computational simplicity. Cross checking of results with other correlation tests on filtered data will also be necessary. Sign and rank correlation are the best methods for present data

processing needs because of their robustness, and independence from the probability distribution function of the data.

4.Acknowledgements

I would like to thank the National Science Foundation for its support in undergraduate research, which I believe is very important. I would mainly like to thank Drs. Guenakh Mitselmakher and Sergei Klimenko for their guidance, support, and knowledge of experimental physics. I am also thankful to Drs. Kevin Ingersent and Alan Dorsey for making the summer research program a valuable step towards becoming a physicist. Last of all, I would like to acknowledge Drs. Darin Acosta, Richard Cavanaugh, Steven Detweiler, and Jorge Rodriguez for fielding various important questions.

5.References

[1]D. Sigg, LIGO-P980007-00-D, **Gravitational Waves**, proceedings of TASI 98, LIGO Hanford Observatory, Richland, WA, 1998 (unpublished).

[2] C. W. Misner, K. S. Thorpe, and J.A. Wheeler, *Gravitation* (W. H. Freeman & Company, New York, 1973).

[3] W. H. Press, S.A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in Fortran* (Cambridge University Press, Cambridge, 1994).

[4]W. Sweldens, and P. Schroder. *Building Your Own Wavelets at Home*. Technical Report 1995:5, Industrial Mathematics Initiative, Department of Mathematics, University of South Carolina, 1995.

[5]R. Sedgewick, *Algorithms in C++* (Addison Wesley, Menlo Park, California, 1998).

[6]G. A. Tammann, A. Sandage, and B. Reindl. *The Distance of the Virgo Cluster*.19th Texas Symposium on Relativistic Astrophysics and Cosmology, Paris, 1998.

6.APPENDIX

This appendix includes the macro files that were written in c++ for ROOT. The programs are dependent on classes in ROOT and WAT. The sections that are commented out, /* */, were compiled and not run as macros.

Mira.C

```
#include <math>
/*
#define NR_END 1
#include <math>
#define ITMAX 100
#define EPS 3.0e-7
#define FPMIN 1.0e-30
#define NRANSI



static double sqrarg;
#define SQR(a) ((sqrarg=(a)) == 0.0 ? 0.0 : sqrarg*sqrarg)
void nrerror(char error_text[])
/* error handler */
{
        fprintf(stderr,"Bad computer! error...\n");
        fprintf(stderr,"%s\n",error_text);
        fprintf(stderr,"...now exiting to system...\n");
        exit(1);
}



double betacf(double a, double b, double x)
{
        double qap, qam, qab, em, tem, d;
        double bz, bm = 1.0, bp, bpp;
        double az = 1.0, am = 1.0, ap, app, aold;
        int m;


        qab = a + b;
        qap = a + 1.0;
        qam = a - 1.0;
        bz = 1.0 - qab * x / qap;
        for (m = 1; m <= ITMAX; m++) {
                em = (double) m;
                tem = em + em;
                d = em * (b - em) * x / ((qam + tem) * (a + tem));
                ap = az + d * am;
                bp = bz + d * bm;
                d = -(a + em) * (qab + em) * x / ((qap + tem) * (a + tem));
                app = ap + d * az;
```

```
                bpp = bp + d * bz;
                aold = az;
                am = ap / bpp;
                bm = bp / bpp;
                az = app / bpp;
                bz = 1.0;
                if (fabs(az - aold) < (EPS * fabs(az))) return az;
        }
        nrerror("a or b too big, or ITMAX too small in betacf");
}



double erfcc(double x)
{
        double t, z, ans;

        z = fabs(x);
        t = 1.0 / (1.0 + 0.5 * z);
        ans = t * exp(-z * z - 1.26551223 + t * (1.00002368 +
                                t * (0.37409196 + t * (0.09678418 +
                                t * (-0.18628806 + t * (0.27886807 +
                                t * (-1.13520398 + t * (1.48851587 +
                                t * (-0.82215223 + t * 0.17087277)))))))));
        return x >= 0.0 ? ans : 2.0 - ans;
}


double gammln(double xx)

{
        double x, y, tmp, ser;
        static double cof[6] = {76.18009172947146, -86.50532032941677,
                24.01409824083091, -1.231739572450155,
                0.1208650973866179e-2, -0.5395239384953e-5};
        int j;

        y = x = xx;
        tmp = x + 5.5;
        tmp -= (x + 0.5) * log(tmp);
        ser = 1.000000000190015;
        for (j = 0; j <= 5; j++) ser += cof[j] / ++y;
        return -tmp + log(2.5066282746310005 * ser / x);
}




double betai(double a, double b, double x)
{
  double bt;

        if (x < 0.0 || x > 1.0) nrerror("Bad x in routine betai");
        if (x == 0.0 || x == 1.0) bt = 0.0;
        else
                bt = exp(gammln(a + b) - gammln(a) - gammln(b) + a * log(x) +
                                b * log(1.0 - x));
```

```
        if (x < (a + 1.0) / (a + b + 2.0))
                return bt * betacf(a, b, x) / a;
        else
                return 1.0 - bt * betacf(b, a, 1.0 - x) / b;
}



inline void gcf(double *gammcf, double a, double x, double *gln)
{

        int i;
        double an,b,c,d,del,h;

        *gln=gammln(a);
        b=x+1.0-a;
        c=1.0/FPMIN;
        d=1.0/b;
        h=d;
        for (i=1;i<=ITMAX;i++) {
                an = -i*(i-a);
                b += 2.0;
                d=an*d+b;
                if (fabs(d) < FPMIN) d=FPMIN;
                c=b+an/c;
                if (fabs(c) < FPMIN) c=FPMIN;
                d=1.0/d;
                del=d*c;
                h *= del;
                if (fabs(del-1.0) < EPS) break;
        }
        if (i > ITMAX) nrerror("a too large, ITMAX too small in gcf");
        *gammcf=exp(-x+a*log(x)-(*gln))*h;
}


inline void gser(double *gamser, double a, double x, double *gln)
{

        int n;
        double sum,del,ap;

        *gln=gammln(a);
        if (x <= 0.0) {
                if (x < 0.0) nrerror("x less than 0 in routine gser");
                *gamser=0.0;
                return;
        } else {
                ap=a;
                del=sum=1.0/a;
                for (n=1;n<=ITMAX;n++) {
                        ++ap;
                        del *= x/ap;
                        sum += del;
                        if (fabs(del) < fabs(sum)*EPS) {
                                *gamser=sum*exp(-x+a*log(x)-(*gln));
                                return;
                        }
```

```
                }
                nrerror("a too large, ITMAX too small in routine gser");
                return;
        }
}




double gammq(double a, double x)
{

        double gamser,gammcf,gln;

        if (x < 0.0 || a <= 0.0) nrerror("Invalid arguments in routine gammq");
        if (x < (a+1.0)) {
                gser(&gamser,a,x,&gln);
                return 1.0-gamser;
        } else {
                gcf(&gammcf,a,x,&gln);
                return gammcf;
        }
}




double gammp(double a, double x)
{

        double gamser,gammcf,gln;

        if (x < 0.0 || a <= 0.0) nrerror("Invalid arguments in routine gammp");
        if (x < (a+1.0)) {
                gser(&gamser,a,x,&gln);
                return gamser;
        } else {
                gcf(&gammcf,a,x,&gln);
                return 1.0-gammcf;
        }
}
*/
//This file assigns ranks to an array and outputs the sums of F^3-F where f=
number of ties for a certain number.

inline void  crank(WaveData &w,double wiop)
{

   int jo=0,ji=0,jt=0,p=w.N;
   double ter,rank;
   wiop=0.0;
   while (jo<p-1)
   {
      if(w.data[jo+1]!=w.data[jo])
      {
       w.data[jo]=jo+1;
       ++jo;
```

```
      }
      else
      {
       for (jt=jo+1;jt<p && w.data[jt]==w.data[jo];jt++);
       rank=0.5*(jo+jt+1);
       for (ji=jo;ji<jt;ji++) w.data[ji]=rank;
       ter=jt-jo;
       wiop += ter**3-ter;
       jo=jt;
      }
   }
   if (jo==p-1)w.data[jo]=p;
}
//This is the implementation of quicksort for sorting two lists at the same time.
/*This is provided in quor.so . It stays in to be able to port to other systems
static const int MH = 15;
inline void ex(double &A, double &B,double &C,double &D)
    {
       double t = A; A = B; B = t;
       double v = C; C = D; D = v;
    }
inline void cx(double &A, double &B,double &C,double &D)
{
   if (B < A)
   {
      ex(A, B,C,D);
   }
}
static int partition(double x[],double y[], int i, int j)
{
   int k = j;
   double v = x[k],w=y[k];
   i--;
   while (i < j)
   {
      while (x[++i] < v);
      while (--j>i && x[j]>v);
      if (i < j)
      {
       ex(x[i],x[j],y[i],y[j]);
      }
   }
   x[k] = x[i]; y[k]=y[i];
   x[i] = v; y[i]=w;
   return i;
}
void ins(double a[],double b[], int l, int r)
{
   int i;
     for (i = r; i > l; i--)
      cx(a[i-1], a[i],b[i-1],b[i]);
     for (i = l+2; i <= r; i++)
     {
      int j = i;
      double y = b[i];
      double v = a[i];
      while (v < a[j-1])
      {
```

```
         a[j] = a[j-1];
         b[j] = b[j-1];
         j--;
       }
       a[j] = v;
       b[j] = y;
      }
}
void qisor(double a[],double b[], int l, int r)
{
    if (r-l <= MH) return;
   ex(a[(l+r)/2], a[r-1],b[(l+r)/2],b[r-1]);
   cx(a[l], a[r-1],b[l],b[r-1]);
   cx(a[l], a[r],b[l],b[r]);
        cx(a[r-1], a[r],b[r-1],b[r]);
      int i = partition(a,b, l+1, r-1);
      qisor(a,b, l, i-1);
      qisor(a,b, i+1, r);
}


void hys(double a[],double b[], int l, int r)
{
   qisor(a,b, l, r);
   ins(a,b, l, r);
}
*/

//This is where the linear correlation coefficient(lrs) is calculated
void linc(WaveData &arr1, WaveData &arr2)
{
   int j,n=arr1.N;
   double me1,me2,rm1,lcs,lts,lrs,rm2,dw=0.0,en,sf=0.0,sg=0.0;

   arr1.getStatistics(me1,rm1);
   arr2.getStatistics(me2,rm2);

   for (j=0;j<n;j++)
   {
     dw += (arr1.data[j]-me1)*(arr2.data[j]-me2);
     sf += (arr1.data[j]-me1)*(arr1.data[j]-me1);
     sg += (arr2.data[j]-me2)*(arr2.data[j]-me2);
     }

   cout<<"rms1="<<sqrt(sf/n)<<"   rms2="<<sqrt(sg/n)<<endl;
   if (sqrt(sf)*sqrt(sg)==0.0)lrs=0.0;
   else lrs=dw/(sqrt(sf)*sqrt(sg));
   lcs = lrs*sqrt((n-2)/(1-lrs*lrs+1e-50));
   en = 0.5*log((1.0+lrs+1e-50)/(1.0-lrs+1e-50));
   lts = erfcc(fabs((en)*sqrt(n-1.0))/sqrt(2.0));
   cout<<"The linear correlation coefficient = "<<lrs<<"\n"<<endl;
   cout<<"The linear 2-sided signifigance  = "<<lts<<"\n"<<endl;
   cout<<"The linear signifigance  = "<<lcs<<"\n"<<endl;
}
/*
inline void linc(double arr1[],double arr2[],double &lrs,double &lcs, double &lts)
{
   int j,n=arr1.N;
```

```
   double me1=0.0,me2=0.0,rm1,dw,en,sf,sg;
   for (j=0;j<n;j++)
   {
      me1+=arr1.data[j];
      me2+=arr2.data[j];
   }
   rm1=n;
   me1/=rm1;
   me2/=rm1;
   for (j=0;j<n;j++)
   {
     dw += (arr1.data[j]-me1)*(arr2.data[j]-me2);
     sf += (arr1.data[j]-me1)*(arr1.data[j]-me1);
     sg += (arr2.data[j]-me2)*(arr2.data[j]-me2);
     }
   if (sqrt(sf)*sqrt(sg)==0.0)lrs=0.0;
   else lrs=dw/(sqrt(sf)*sqrt(sg));
   lcs = lrs*sqrt((n-2)/(1-lrs*lrs+1e-50));
   en = 0.5*log((1.0+lrs+1e-50)/(1.0-lrs+1e-50));
   lts = erfcc(fabs((en)*sqrt(n-1.0))/sqrt(2.0));
}*/
// Here is where the program for the sign correlation test is
void sigc(WaveData &arr1, WaveData &arr2)
{
   int n,li;
   li=arr1.N/16;
   double en,sts,scs,scc=0.0;
   WaveData r(li),s(li);
   WaveData u(65536);
   u.ReadBinary("look.dat");
   n=arr1.N;
   bitSig(arr1,r);
   bitSig(arr2,s);
   scc=bitSu(r,s,u);
   scs= scc*sqrt((n-2)/(1-scc*scc+1e-50));
   en = 0.5*log((1.0+scc+1e-50)/(1.0-scc+1e-50));
   sts= erfcc(fabs((en)*sqrt(n-1.0))/sqrt(2.0));
   cout<<"The sign coefficient = "<<scc<<"\n"<<endl;
   cout<<"The sign 2-sided signifigance = "<<sts<<"\n"<<endl;
   cout<<"The sign signifigance = "<<scs<<"\n"<<endl;
   WaveData r(),s();
   WaveData u();
}

void sigs(WaveData &arr1,WaveData &arr2)
{
   double en=arr1.N,gt=0.0,scc=0.0,scs=0.0,sts=0.0;
   int li,n=arr1.N,j;


   for (j=0;j<n;j++)
   {
      if(arr1.data[j]>0)arr1.data[j]=1;else arr1.data[j]=-1;
      if(arr2.data[j]>0)arr2.data[j]=1;else arr2.data[j]=-1;
      gt+= arr1.data[j]*arr2.data[j];
   }
   scc=gt/en;
   en=0.0;
```

```
   scs= scc*sqrt((n-2)/(1-scc*scc+1e-50));
   en = 0.5*log((1.0+scc+1e-50)/(1.0-scc+1e-50));
   sts= erfcc(fabs((en)*sqrt(n-1.0))/sqrt(2.0));
   cout<<"The sign coefficient = "<<scc<<"\n"<<endl;
   cout<<"The sign 2-sided signifigance = "<<sts<<"\n"<<endl;
   cout<<"The sign signifigance = "<<scs<<"\n"<<endl;

}



void wavc(WaveData &arr1, WaveData &arr2)
{
   int n=arr1.N;
   double wcc=0.0,wcs=0.0,en=0.0,wts=0.0;
   WaveData bm,qs,qf;
   bm.ReadBinary("look.dat");
   WaveletD wx(8,1);
   WaveletD wy(8,1);
   WaveData* pwx;
   WaveData* pwy;
   wx.t2w(arr1,6);
   wy.t2w(arr2,6);
   pwx=(wx.pWDC);
   pwy=(wy.pWDC);
   qs=bitSign(*pwx);
   qf=bitSign(*pwy);
   wcc=bitSum(qs,qf,bm);
   wcs= wcc*sqrt((n-2)/(1-wcc*wcc+1e-50));
   en = 0.5*log((1.0+wcc+1e-50)/(1.0-wcc+1e-50));
   wts= erfcc(fabs((en)*sqrt(n-1.0))/sqrt(2.0));
   cout<<"The wsign coefficient = "<<wcc<<"\n"<<endl;
   cout<<"The wsign signifigance = "<<wcs<<"\n"<<endl;
   cout<<"The wsign 2-sided signifigance = "<<wts<<"\n"<<endl;
   WaveData bm(),qs(),qf(),pwx(),pwy();
   WaveletD wx();
   WaveletD wy();
}
void wavc(WaveData &arr1, WaveData &arr2, double &wcc,double &wcs,double &wts)
{
   int n=arr1.N;
   double en=0.0;
   WaveData bm,qs,qf;
   WaveData as(n),cs(n);
   as=arr1;cs=arr2;
   bm.ReadBinary("look.dat");
   WaveletD wx(8,1);
   WaveletD wy(8,1);
   WaveData* pwx;
   WaveData* pwy;
   wx.t2w(as,6);
   wy.t2w(cs,6);
   pwx=(wx.pWDC);
   pwy=(wy.pWDC);
   qs=bitSign(*pwx);
   qf=bitSign(*pwy);
   wcc=bitSum(qs,qf,bm);
```

```
   wcs= wcc*sqrt((n-2)/(1-wcc*wcc+1e-50));
   en = 0.5*log((1.0+wcc+1e-50)/(1.0-wcc+1e-50));
   wts= erfcc(fabs((en)*sqrt(n-1.0))/sqrt(2.0));
   as=0.0;cs=0.0;bm=0.0;qs=0.0;qf=0.0;pwx=0.0;pwy=0.0;wx=0.0;wy=0.0;
   WaveData as(),cs();
   WaveData bm(),qs(),qf(),pwx(),pwy();
   WaveletD wx();
   WaveletD wy();
}

//This is the big cheese, which outputs rank correlation data.
/*
//This is the big cheese, which outputs rank correlation data.
inline void  srank( WaveData &arr1, WaveData &arr2,WaveData &u,double bik[])
{
   if (arr1.N != arr2.N){
      cout<<"WaveData lengths are not the same."<<endl;
      return;
   }
   int j,n=arr1.N;
   double
t,sg=0.0,lcs,sf=0.0,fac,en3n,en=0.0,df,scc,dw=0.0,lts,rs,probrs,lrs,tint=1.0e-
20,scs,sts;
   for(j=0;j<9;j++)bik[j]=0.0;
   WaveData wksp1(n),wksp2(n);
   wksp1=arr1;
   wksp2=arr2;
   sigc(arr1,arr2,u,scc,scs,sts);
   linc(arr1,arr2,lrs,lcs,lts);
   quor(wksp1.data,wksp2.data,0,n-1);
   crank(wksp1,sf);
   quor(wksp2.data,wksp1.data,0,n-1);
   crank(wksp2,sg);
   //This is where the sum squared difference of ranks comes in
   for (j=0;j<n;j++){
      dw += (wksp1.data[j]-wksp2.data[j])**2;
   }
   en=n;
   en3n=en*en*en-en;
   fac=(1.0-sf/en3n)*(1.0-sg/en3n);
   rs=(1.0-(6.0/en3n)*(dw+(sf+sg)/12.0))/sqrt(fac);
   fac=(rs+1.0)*(1.0-(rs));
   if (fac > 0.0) {
      t=(rs)*sqrt((en-2.0)/fac);
      df=en-2.0;
      probrs=betai(0.5*df,0.5,df/(df+t*t));
   }else
   { probrs=0.0;}
   bik[0]=lrs;
   bik[1]=lcs;
   bik[2]=lts;
   bik[3]=rs;
   bik[4]=t;
   bik[5]=probrs;
   bik[6]=scc;
   bik[7]=scs;
   bik[8]=sts;
```

```
   wksp1=0.0;
   wksp2=0.0;
   WaveData wksp1(),wksp2();
}*/
//This is the big cheese, which outputs rank correlation data.
void  srank(WaveData &arr1, WaveData &arr2)
{
   gBenchmark->Start("massbb");
   if (arr1.N != arr2.N)
   {
      cout<<"WaveData lengths are not the same."<<endl;
      return;
   }
   int j,n=arr1.N;
   double
vard,t,sg=0.0,lcs,sf=0.0,fac,en3n,en=0.0,df,scc,aved,dw=0.0,zd,lts,probd,rs,probrs
,lrs,tint=1.0e-20,sts,scs,wcs,wts,wcc;
   double *wksp1 = new double[n];
   double *wksp2 = new double[n];

   j=0;
   while(j<n)
   {
      wksp1[j]=arr1.data[j];
      wksp2[j]=arr2.data[j];
      j++;
   }
   WaveData u(65536);
   u.ReadBinary("look.dat");
   sigc(arr1,arr2,u,scc,scs,sts);
   linc(arr1,arr2,lrs,lcs,lts);
   quor(wksp1,wksp2,0,n-1);
   crank(n,wksp1,sf);
   quor(wksp2,wksp1,0,n-1);
   crank(n,wksp2,sg);
   //This is where the sum squared difference of ranks comes in
   j=0;
   while(j<n)
   {
      dw += (wksp1[j]-wksp2[j])**2;
      j++;
   }
   en=n;
   en3n=en*en*en-en;
   //aved=en3n/6.0-(sf+sg)/12.0;
   fac=(1.0-sf/en3n)*(1.0-sg/en3n);
   //vard=((en -1.0)*en*en*SQR(en+1.0)/36.0)*fac;
   //zd=(dw-aved)/sqrt(vard);
   // probd=erfcc(fabs(zd)/1.4142136);
   rs=(1.0-(6.0/en3n)*(dw+(sf+sg)/12.0))/sqrt(fac);
   fac=(rs+1.0)*(1.0-(rs));
   if (fac > 0.0)
   {
      t=(rs)*sqrt((en-2.0)/fac);
      df=en-2.0;
      probrs=betai(0.5*df,0.5,df/(df+t*t));
   }
   else
```

```
   { probrs=0.0;}


   cout<<"_____________________________________"<<endl;
   cout<<"|  Wave  correlation coefficient      = "<<wcc<<endl;
   cout<<"|  Wave signifigance                  = "<<wcs<<endl;
   cout<<"|  Wave 2-sided signifigance          = "<<wts<<endl;
   cout<<"|  Sign correlation coefficient       = "<<scc<<endl;
   cout<<"|  Sign signifigance                  = "<<scs<<endl;
   cout<<"|  Sign 2-sided signifigance          = "<<sts<<endl;
   cout<<"|  Linear correlation coefficient     = "<<lrs<<endl;
   cout<<"|  LCC's signifigance                 = "<<lcs<<endl;
   cout<<"|  LCC's 2-sided signifigance         = "<<lts<<endl;
   cout<<"|  Spearman's correlation coefficient = "<<rs<<endl;
   cout<<"|  Signifigance of non-zero Rs        = "<<t<<endl;
   cout<<"|  2-sided signifigance level of Rs   = "<<probrs<<endl;
   cout<<"|  Number of standard deviations      = "<<zd<<endl;
   cout<<"|  Sum squared difference of ranks    = "<<dw<<endl;
   cout<<"|  Variance of D(above)               = "<<vard<<endl;
   cout<<"|  2-sided signifigance level         = "<<probd<<endl;
   cout<<"|_____________________________________/ "<<"\n"<<endl;
   wksp1=0.0;
   wksp2=0.0;
   delete [] wksp1 ;
   delete [] wksp2 ;
   gBenchmark->Show("massbb");
   Float_t massbb_rt = gBenchmark->GetRealTime("massbb");
   gBenchmark->Stop("massbb");
   gBenchmark->Reset();

}
```

Aver.C

```
 void ror(WaveData &gre,WaveData &sre,int u,double bink)
        {
           int p,h=gre.N;
           double rms=0.0,ave=0.0;
           gre.getStatistics(ave,rms);
           sre.data[u]=0.0;
           for(p=0;p<h;p++)
           {
            sre.data[u]+=(gre.data[p]-ave)*(gre.data[p]-ave);
           }
           sre.data[u]=sqrt(sre.data[u])/bink;

        }
void naver(int h,double sig,double n,double ho)
{
   gROOT->Reset();
   gBenchmark->Start("massbb");
   int j,i,u,mi,er,p,mo=ho;
   int ki;

   mi= mo*h;
   double noise=20.0,hp=h;
```

```
double bih[9];
WaveData lc(mo),ls(mo),lss(mo),s(mo), te(mo),pro(mo),xi(mo);
WaveData scc(mo),scs(mo),sts(mo);
WaveData lin(mi),sin(mi),rin(mi),slin(mi),ssin(mi),srin(mi);
WaveData selin(mo), sesin(mo), serin(mo),sselin(mo), ssesin(mo), sserin(mo);
WaveData lies(h),sies(h),ries(h), slies(h),ssies(h),sries(h);
WaveData a(100000),b(100000);
double bink;
bink=sqrt(hp*hp-hp);

scc=0.0;
lc =0.0;
ls =0.0;
lss=0.0;
s  =0.0;
te =0.0;
pro=0.0;
scs=0.0;
sts=0.0;
serin=0.0;
for (u=0;u<mo;u++)
 {
    cout<<"The U loop # ="<<u<<endl;
    ki=(u+1)*n;
    a.Resize(ki);
    b.Resize(ki);
    xi.data[u]=ki;
    for(j=0;j<h;j++)
   {
      cout<<j<<endl;

      a=0.;b=0.;
      AddGauss(a,sig);
      //AddSin(a,gp,10.0);
      b=a;
      AddGauss(a,noise);
      AddGauss(b,noise);
      srank(a.data,b.data,bih,ki);
      lin.data[u*h+j] = bih[0];
      rin.data[u*h+j] = bih[3];
      sin.data[u*h+j] = bih[6];
      slin.data[u*h+j] = bih[1];
      srin.data[u*h+j] = bih[4];
      ssin.data[u*h+j] = bih[7];
      lc.data[u] += bih[0];
      ls.data[u] += bih[1];
      lss.data[u] += bih[2];
      s.data[u] += bih[3];
      te.data[u] += bih[4];
      pro.data[u] += bih[5];
      scc.data[u] += bih[6];
      scs.data[u] += bih[7];
      sts.data[u] += bih[8];

   }

    lc.data[u]/=hp;
    ls.data[u]/=hp;
```

```
       lss.data[u]/=hp;
       s.data[u]/=hp;
       te.data[u]/=hp;
       pro.data[u]/=hp;
       scc.data[u]/=hp;
       scs.data[u]/=hp;
       sts.data[u]/=hp;
       for (int p=0;p<h;p++)
       {
        ries.data[p]=rin.data[u*h+p];
        sies.data[p]=sin.data[u*h+p];
        lies.data[p]=lin.data[u*h+p];
        sries.data[p]=srin.data[u*h+p];
        ssies.data[p]=ssin.data[u*h+p];
        slies.data[p]=slin.data[u*h+p];
       }

       ror(ries,serin,u,bink);
       ror(sies,sesin,u,bink);
       ror(lies,selin,u,bink);
       ror(sries,sserin,u,bink);
       ror(slies,sselin,u,bink);
       ror(ssies,ssesin,u,bink);
    }
   sselin.DumpBinary("ntmp.2/selin.dat");
   sserin.DumpBinary("ntmp.2/serin.dat");
   ssesin.DumpBinary("ntmp.2/sesin.dat");
   selin.DumpBinary("ntmp.2/selin.dat");
   serin.DumpBinary("ntmp.2/serin.dat");
   sesin.DumpBinary("ntmp.2/sesin.dat");
   rin.DumpBinary("ntmp.2/rin.dat");
   lin.DumpBinary("ntmp.2/lin.dat");
   sin.DumpBinary("ntmp.2/sin.dat");
   lc.DumpBinary("ntmp.2/lc.dat");
   ls.DumpBinary("ntmp.2/ls.dat");
   lss.DumpBinary("ntmp.2/lss.dat");
   s.DumpBinary("ntmp.2/s.dat");
   te.DumpBinary("ntmp.2/te.dat");
   pro.DumpBinary("ntmp.2/pro.dat");
   scc.DumpBinary("ntmp.2/scc.dat");
   scs.DumpBinary("ntmp.2/scs.dat");
   sts.DumpBinary("ntmp.2/sts.dat");

   WaveData xp(mo);
   xp=0.0;

  TCanvas *c1 = new TCanvas("c1","Linear(red),Rank(black),and Sign(green)
Correlation Coefficients",200,10,700,500);
  TGraph *g = new TGraphErrors(mo,xi.data,s.data,xp.data,serin.data);g-
>SetLineColor(1);
  g->SetTitle(" Linear(red), Rank(black),and Sign(green) Correlation
Coefficients");
  g->Draw();
  g->Draw("AL");
  g->GetXaxis()->SetTitle("Signal to Noise Ratio");
  g->GetYaxis()->SetTitle("Magnitudes of the Correlation Coefficients");
  g->SetMinimum(0.0);
  g->SetMaximum(0.2);
```

```
  g->Draw("AL");
  TGraph *gw = new TGraphErrors(mo,xi.data,lc.data,xp.data,selin.data);gw-
>SetLineColor(2);
  gw->Draw("LP");
  TGraph *hw = new TGraphErrors(mo,xi.data,scc.data,xp.data,sesin.data);hw-
>SetLineColor(8);
  hw->Draw("LP");


  TCanvas *c2 = new TCanvas("c2","Linear(red), Rank(black),and Sign(green)
Significance",200,10,700,500);
  TGraph *f = new TGraphErrors(mo,xi.data,te.data,xp.data,sserin.data);f-
>SetLineColor(1);
  f->SetTitle("Linear(red), Rank(black),and Sign(green) Significance");
  f->Draw();
  f->Draw("AL");
  f->GetXaxis()->SetTitle("Signal to Noise Ratio");
  f->GetYaxis()->SetTitle("Significance(t)");
  f->Draw("AL");
  TGraph *fw = new TGraphErrors(mo,xi.data,ls.data,xp.data,sselin.data);fw-
>SetLineColor(2); fw->Draw("LP");
  TGraph *fu = new TGraphErrors(mo,xi.data,scs.data,xp.data,ssesin.data);fu-
>SetLineColor(8);  fu->Draw("LP");

  TCanvas *c3 = new TCanvas("c3","Linear(red), Rank(black),and Sign(green) 2-Sided
Significance",200,10,700,500);
  TGraph *e = new TGraph(mo,xi.data,pro.data);e->SetLineColor(1);
  e->SetTitle("Linear(red), Rank(black),and Sign(green) 2-Sided Significance");
  e->Draw();
  e->Draw("AL");
  e->GetXaxis()->SetTitle("Signal to Noise Ratio");
  e->GetYaxis()->SetTitle("2-Sided Significance");
  e->Draw("AL");
  TGraph *ew = new TGraph(mo,xi.data,lss.data);ew->SetLineColor(2); ew-
>Draw("LP");c3->SetLogy();
  TGraph *eu = new TGraph(mo,xi.data,sts.data);eu->SetLineColor(8); eu-
>Draw("LP");



  gBenchmark->Show("massbb");
  Float_t massbb_rt = gBenchmark->GetRealTime("massbb");
  gBenchmark->Stop("massbb");
  gBenchmark->Reset();
  WaveData lc(),ls(),lss(),s(), te(),pro();
  WaveData scc(),scs(),sts();
  WaveData lin(),sin(),rin(),slin(),ssin(),srin();
  WaveData selin(), sesin(), serin(),sselin(), ssesin(), sserin();
  WaveData lies(),sies(),ries(), slies(),ssies(),sries();
  WaveData a(),b(),xi(),xp();



}
void naver()
{
   cout<<" _____________________________________________________\n"
       <<"| naver(int h,double sig,double n,double f)      |\n"
```

```
       <<"| h=#of tries to average                          |\n"
       <<"| sig=RMS of signal                               |\n"
       <<"| n=1st signal length                             |\n"
       <<"| f=# of n's to traverse                          |\n"
       <<"|_____________________________________________|\n"<<endl;
}


void aver(int h,double bs,double inc,double mo,double noise)
{
   gROOT->Reset();
   gBenchmark->Start("massbb");
   WaveData ut(65536);
   ut.ReadBinary("look.dat");
   int n=16384;
   int j,i,u,mi,er,p;
   mi= mo*h;
   double gp,bink,hp=h;
   double bih[9];
   WaveData a(n), b(n);
   WaveData lc(mo),ls(mo),lss(mo),s(mo), te(mo),pro(mo);
   WaveData scc(mo),scs(mo),sts(mo);
   WaveData lin(mi),sin(mi),rin(mi),slin(mi),ssin(mi),srin(mi);
   WaveData xi(mo),xp(mo);
   WaveData selin(mo), sesin(mo), serin(mo),sselin(mo), ssesin(mo), sserin(mo);
   WaveData lies(h),sies(h),ries(h), slies(h),ssies(h),sries(h);
   Float_t mass_rt;


scc=0.0;lc=0.0;ls=0.0;lss=0.0;s=0.0;te=0.0;pro=0.0;scs=0.0;sts=0.0;serin=0.0;lin=0
.0;sin=0.0;rin=0.0;slin=0.0;ssin=0.0;srin=0.0;xi=0.0;xp=0.0;lies=0.0;sies=0.0;ries
=0.0;slies=0.0;ssies=0.0;sries=0.0;sserin=0.0;ssesin=0.0;sselin=0.0;sesin=0.0;seli
n=0.0;

   for (u=0;u<mo;u++)
    {
       cout<<"This is U level----"<<u<<endl;
       for(j=0;j<h;j++)
      {

         cout<<j<<endl;
         gBenchmark->Start("mass");
         gp = bs + inc*u;
         AddGaus(a.data,gp,n);
         //AddSin(a,gp,100.0);
         b=a;
         AddGaus(a.data,noise,n);
         AddGaus(b.data,noise,n);
         srank(a.data,b.data,bih,n);

         lin.data[u*h+j] = bih[0];
         rin.data[u*h+j] = bih[3];
         sin.data[u*h+j] = bih[6];
         slin.data[u*h+j] = bih[1];
         srin.data[u*h+j] = bih[4];
         ssin.data[u*h+j] = bih[7];
         lc.data[u] += bih[0];
         ls.data[u] += bih[1];
```

```
        lss.data[u] += bih[2];
        s.data[u] += bih[3];
        te.data[u] += bih[4];
        pro.data[u] += bih[5];
        scc.data[u] += bih[6];
        scs.data[u] += bih[7];
        sts.data[u] += bih[8];
        gBenchmark->Show("mass");
          mass_rt = gBenchmark->GetRealTime("mass");
        gBenchmark->Stop("mass");
        gBenchmark->Reset();
      }
       lc.data[u]/=hp;
       ls.data[u]/=hp;
       lss.data[u]/=hp;
       s.data[u]/=hp;
       te.data[u]/=hp;
       pro.data[u]/=hp;
       scc.data[u]/=hp;
       scs.data[u]/=hp;
       sts.data[u]/=hp;

       ries=0.0;sies=0.0;lies=0.0;sries=0.0;ssies=0.0;slies=0.0;
       for (int p=0;p<h;p++)
       {
        ries.data[p]=rin.data[u*h+p];
        sies.data[p]=sin.data[u*h+p];
        lies.data[p]=lin.data[u*h+p];
        sries.data[p]=srin.data[u*h+p];
        ssies.data[p]=ssin.data[u*h+p];
        slies.data[p]=slin.data[u*h+p];
       }
          if(hp>1)bink=sqrt(hp**2-hp);
       else bink=1.;

       ror(ries,serin,u,bink);
       ror(sies,sesin,u,bink);
       ror(lies,selin,u,bink);
       ror(sries,sserin,u,bink);
       ror(slies,sselin,u,bink);
       ror(ssies,ssesin,u,bink);

}
 sselin.DumpBinary("tmpto4/w16384/selin.dat");
 sserin.DumpBinary("tmpto4/w16384/serin.dat");
 ssesin.DumpBinary("tmpto4/w16384/sesin.dat");
 selin.DumpBinary("tmpto4/w16384/selin.dat");
 serin.DumpBinary("tmpto4/w16384/serin.dat");
 sesin.DumpBinary("tmpto4/w16384/sesin.dat");
 rin.DumpBinary("tmpto4/w16384/rin.dat");
 lin.DumpBinary("tmpto4/w16384/lin.dat");
 sin.DumpBinary("tmpto4/w16384/sin.dat");
 lc.DumpBinary("tmpto4/w16384/lc.dat");
 ls.DumpBinary("tmpto4/w16384/ls.dat");
 lss.DumpBinary("tmpto4/w16384/lss.dat");
 s.DumpBinary("tmpto4/w16384/s.dat");
 te.DumpBinary("tmpto4/w16384/te.dat");
 pro.DumpBinary("tmpto4/w16384/pro.dat");
```

```
   scc.DumpBinary("tmpto4/w16384/scc.dat");
   scs.DumpBinary("tmpto4/w16384/scs.dat");
   sts.DumpBinary("tmpto4/w16384/sts.dat");
  gBenchmark->Show("massbb");
  Float_t massbb_rt = gBenchmark->GetRealTime("massbb");
  gBenchmark->Stop("massbb");
  gBenchmark->Reset();
  lc=0.0;ls=0.0;lss=0.0;s=0.0;te=0.0;pro=0.0;scc=0.0;scs=0.0;sts=0.0;
  lin=0.0;rin=0.0;sin=0.0;slin=0.0;ssin=0.0;srin=0.0;
  selin=0.0;sesin=0.0;serin=0.0;sselin=0.0;ssesin=0.0;sserin=0.0;
  lies=0.0;sies=0.0;ries=0.0;slies=0.0;ssies=0.0;sries=0.0;
  xi=0.0;xp=0.0;ut=0.0;
  WaveData a(),b(),lc(),ls(),lss(),s(), te(),pro();
  WaveData scc(),scs(),sts();
  WaveData lin(),sin(),rin(),slin(),ssin(),srin();
  WaveData selin(), sesin(), serin(),sselin(), ssesin(), sserin();
  WaveData lies(),sies(),ries(), slies(),ssies(),sries();
  WaveData xi(),xp(),ut();

}



void aver()
{
   cout<<" _____________________________________________________\n"
       <<"| aver(int h,double beg,double inc,int mo,double noise) |\n"
       <<"| h=#of tries to average                                |\n"
       <<"| beg= beginning RMS of signal                          |\n"
       <<"| inc=increment size                                    |\n"
       <<"| mo=# of increments                                    |\n"
       <<"| noise= RMS of noise                                   |\n"
       <<"|_____________________________________________________|\n"<<endl;
}
```