LU TP 94-5
May 1994

Delta 2.0 – A Program for Finding Dependencies in Tables of Data

Hong Pi[1] and Carsten Peterson[2]

Department of Theoretical Physics, University of Lund,
Sölvegatan 14 A, S-223 62 Lund, Sweden

Computer Physics Communications, **83**, 293 (1994)

Abstract:

A package written in C for identifying variable dependencies in tables of data is presented. The key ingredient is the δ -test, which establishes dependency structures by exploiting the properties of continuous functions using conditional probabilities formed out of the data. The method estimates most relevant variables, embedding dimensions and noise levels. The program, which is self-contained, also includes optional graphical output.

[1] pihong@thep.lu.se
[2] carsten@thep.lu.se

PROGRAM SUMMARY

Title of Program: `Delta 2.0`

Catalogue number:

Program obtainable from: anonymous ftp at `thep.lu.se` in directory `pub/LundPrograms` – `delta.tar.Z` and `delta.readme` respectively.

Computer for which the programme is designed: DEC Alpha, DECstation, SUN, NeXT, VAX, IBM, Hewlett-Packard, and others with a C compiler

Computer: DEC Alpha 3000; installation: Department of Theoretical Physics, University of Lund, Lund, Sweden

Operating system: DEC OSF 1.3

Program language used: ANSI C

High speed storage required: $\approx$ 170k words for a data file of 4000 lines

No. of bits in a word: 32

Peripherals used: terminal for input, terminal or printer for output

No. of lines in combined program and test deck: 2950 excluding the `Xplot` graphic package

CPC Library subroutines used: none

Keywords: data analysis, dependencies, correlations, nonlinear systems, embedding dimension, artificial neural network

Nature of physical problem

Analysis of experimental data for determining dependencies among the measured variables and establishing noise levels. This is a frequently occurring task in natural sciences. Standard methods for these problems are typically limited to linear dependencies like using correlation matrices. Many problems in physics are nonlinear by nature and hence require analysis methods that are not limited to linear approximations.

Method of solution

The core of the algorithm is based on the δ-test [1], which establishes dependency structures by exploiting the properties of continuous functions. The method, which in contrast to standard correlation methods is not confined to linear dependencies, forms conditional probabilities from data tables, which are then extrapolated to the infinite resolution limit. From these limit values one reads off relative dependencies, noise levels and embedding dimensions. The method is very useful when it comes to single out most relevant input variables for artificial neural network processing. Also, in this case the approach can be used to track residual dependencies of the output errors. The degree of nonlinearity is estimated by comparing the δ-test noise reading with the variance of the residuals of the linear multiple regression model.

Restriction of complexity of the problem

The only restriction of the complexity for an application is set by available memory and CPU time. A table of M variables with N measurements each requires a storage of $M \times N$ double precision numbers. The corresponding CPU time grows like $M^2 N$. For a problem with $M = 5$ (one independent and 4 dependent variables) and $N = 1000$ this amounts to 35 CPU seconds on a DEC Alpha 3000/300.

For users equipped with `X11` some of the informative results can be displayed with interactive graphics.

LONG WRITE-UP

1 Introduction

A system is often modeled by analyzing records of certain system variables. Such modeling could range from parametric approaches to non-parametric ones like Artificial Neural Networks (ANN). The success of such models relies heavily upon identifying the underlying structure in the input space – it is advantageous to know in advance which inputs are most relevant, the embedding dimension in the case of a time series, noise level, etc. Existing methods for doing this are based either on linear regression, which limits the analysis to linear dependencies, or on trial-and-error procedures. The δ-test [1], to be briefly described below, aims at determining any dependency, be it linear or nonlinear, assuming an underlying continuous function.

2 The δ-test

Assume that we have sequences of measurements on a *dependent* variable z_0 and a set of *independent* variables z_1, z_2,...z_m. These measurements can correspond to multivariate time series, or to a univariate time series, in which case z_k should be understood as a time-lagged variable of z_0: $z_k(t) = z_0(t-k)$. The central question is whether there exist functional dependencies of the form

$$z_0 = f(z_1, z_2, ..., z_m) + r \tag{1}$$

where r represents an indeterminable part, which originates either from insufficient dimensionality of the measurements or from real noise. There is of course no crisp borderline between these two interpretations of r.

We approach the problem by constructing conditional probabilities in embedding spaces of various dimensions d. The data can be represented as a series of N points $\mathbf{z}(i)$ in a $(d+1)$-dimensional space $(d = 0, 1, 2, ...)$

$$\mathbf{z}(i) = (z_0(i), z_1(i), .., z_k(i), ., z_d(i)). \tag{2}$$

In terms of distances $l_k(i,j)$ between the k:th components of two vectors $\mathbf{z}(i)$ and $\mathbf{z}(j)$

$$l_k(i,j) = |z_k(i) - z_k(j)|, \quad k = 0, 1, ..., d \tag{3}$$

one can construct the conditional probabilities

$$P_d(\epsilon|\vec{\delta}) \equiv P(l_0 \leq \epsilon | \vec{l} \leq \vec{\delta}) = \frac{n(l_0 \leq \epsilon, \vec{l} \leq \vec{\delta})}{n(\vec{l} \leq \vec{\delta})}, \tag{4}$$

where ϵ and δ are positive numbers and $n(\vec{l} \leq \vec{\delta})$ and $n(l_0 \leq \epsilon, \vec{l} \leq \vec{\delta})$ are the number of vector pairs satisfying the corresponding distance constraints[3]. The probability structures of the dependent variables with respect to the independent ones carry the following important information [1]:

[3]The notation $\vec{l} \leq \vec{\delta}$ is short for $\{(l_1 \leq \delta), (l_2 \leq \delta), ..., (l_d \leq \delta)\}$.

1. For a completely random time series there is no dependency and one has

$$P_0(\epsilon) = P_1(\epsilon|\,\vec{\delta}) = \ldots = P_d(\epsilon|\,\vec{\delta}) = \ldots \tag{5}$$

   This identity, which should be understood in a statistical sense, holds for any choice of positive ϵ and δ.

2. If a continuous map exists as in eq. (1) with no intrinsic noise, then for any $\epsilon > 0$ there exists a δ_ϵ such that

$$P_d(\epsilon|\,\vec{\delta}) = 1 \quad \text{for} \quad \delta \leq \delta_\epsilon \quad \text{and} \quad d \geq d_0 \tag{6}$$

   where d_0 represents some minimum dimension which covers all the relevant variables. This is a direct consequence of the definition of function uniform continuity,

3. In the presence of noise r, $P_d(\epsilon|\,\vec{\delta})$ will no longer saturate to 1 as ϵ becomes smaller than the width Δr_{max} of the noise.

The behavior of $P_d(\epsilon|\,\vec{\delta})$ as a function of δ and ϵ for various d are shown schematically in fig. 1. The consequences of randomness (eq. (5)) and complete determination (eq. (6)) provides a yardstick with which to measure the degree of dependency between the variables. Interesting quantities to examine are the maxima

$$P_d(\epsilon) \equiv \max_{\delta > 0} P_d(\epsilon|\,\vec{\delta}) = P_d(\epsilon|\,\vec{\delta})|_{\delta \leq \delta_\epsilon}. \tag{7}$$

How $P_d(\epsilon)$ changes with d and ϵ provides basically all the information we need (see fig. 1b). To quantify the dependency on each of the variables, it is convenient to define a *dependability index*

$$\overline{\lambda}_d = \frac{\int_0^\infty d\epsilon\,(P_d(\epsilon) - P_{d-1}(\epsilon))}{\int_0^\infty d\epsilon\,(1 - P_0(\epsilon))}, \quad d = 1, 2, \ldots \tag{8}$$

In general $1 \geq \overline{\lambda}_d \geq 0$, while $\overline{\lambda}_d = 1$ (or $\sum \overline{\lambda}_d = 1$) signals a completely deterministic relationship and $\overline{\lambda}_d \approx 0$ singles out irrelevant variables[4].

Constructing statistical quantities out of pairs of points is an efficient utilization of available statistics ($N(N-1)/2$ pairs out of N points). Nevertheless, limited statistics can be problematic, especially if noise levels are high. Statistical errors are estimated as

$$\Delta P_d(\epsilon|\,\vec{\delta}) = 2\sqrt{\frac{P_d(1 - P_d)}{n(\vec{l} \leq \vec{\delta})}}. \tag{9}$$

This expression is not entirely adequate for correlated data, but it serves the purpose to signal the onset of statistically unreliable regions. When statistics are at a premium a variable k once identified as irrelevant is set *inactive*, which means that the condition $l_k \leq \delta$ is omitted when computing $P_d(\epsilon|\,\vec{\delta})$ for $d > k$. This *variable elimination* option cuts down the loss of statistics. Another option useful for low statistics data set is the *single variable analysis*, which computes λ_d with only the d:th variable as the conditional variable.

[4] One should keep in mind, however, that these conditions are only necessary but not sufficient.

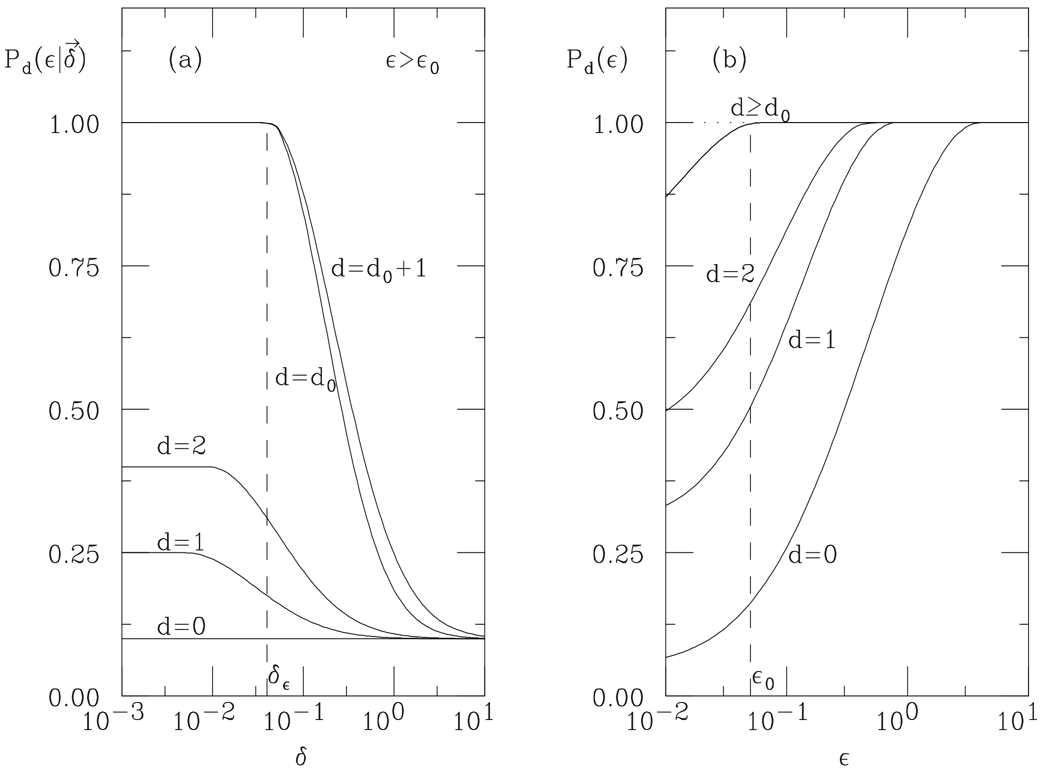

Figure 1: **(a)**. $P_d(\epsilon|\,\vec{\delta})$ as a function of δ for fixed ϵ. Saturation to 1 would be observed if the width of noise is less than ϵ. **(b)**. Behaviour of the maxima $P_d(\epsilon) \equiv \max_{\delta>0} P_d(\epsilon|\,\vec{\delta})$ as a function of ϵ. The region saturating to 1 would be pushed toward smaller ϵ if the d:th conditional variable is relevant. The point ϵ_0 at which the saturation deviates from 1 can be approximately identified as the width of the noise $\Delta r_{max} \sim \epsilon_0$.

3 Nonlinear versus Linear Dependencies

Eq. (1) is reduced to the usual multiple regression (MR) model if f is restricted to be linear,

$$\hat{z}_0 = a_0 + \sum_{i=1}^{m} z_i. \tag{10}$$

The term r is then the residual of the MR predictor $\hat{z}_0$. The coefficient of determination, defined as the ratio of the *explained variation* $\langle(\hat{z}_0 - \overline{\hat{z}}_0)^2\rangle$ over the true variation $\langle(z_0 - \overline{z}_0)^2\rangle$, is given in the MR model as

$$D_{MR} = \frac{\sum_{i=1} a_i \langle z_0, z_i\rangle}{\sigma^2_{z_0}} \tag{11}$$

where $\langle z_0, z_i \rangle$ is the covariance and σ_{z_0} is the standard deviation of the dependent variable. The standard deviation of the residual r is related to the coefficient of determination by

$$\frac{\sigma_r}{\sigma_{z_0}} = \sqrt{1 - D_{MR}}. \tag{12}$$

It is convenient to transform all the variables to have zero mean and unit standard deviation. In what follows and in the numerical implementations, we regard z_i as the transformed variables.

In an optimized nonlinear model, the residual r should resemble white noise. A quantitative estimation of its variance can be obtained in the δ-test by considering the behavior of $P_d(\epsilon)$ in fig. 1b. At the region $\epsilon \sim \epsilon_0$ where $P_d(\epsilon)$ starts to drop off from one, we expect

$$P_d(\epsilon) \approx 1 - Prob(\Delta r > \epsilon) = Prob(\Delta r \leq \epsilon). \tag{13}$$

If the noise is a flat distribution extending from $-R$ to R with standard deviation $\sigma_r = R/\sqrt{3}$, we obtain

$$Prob(\Delta r \leq \epsilon) = \frac{\epsilon}{R}\left(1 - \frac{1}{4}\frac{\epsilon}{R}\right). \tag{14}$$

Correspondingly for Gaussian distributed noise,

$$Prob(\Delta r \leq \epsilon) = \frac{1}{\sqrt{2\pi}} \int_{-\epsilon/\sqrt{2}\sigma_r}^{\epsilon/\sqrt{2}\sigma_r} e^{-\frac{u^2}{2}}\, du \tag{15}$$

It is possible to fit $P_d(\epsilon)$ to eq. (14) and (15) in order to determine whether the noise distribution is uniform, Gaussian, or neither. We use eq. (14) to obtain a conservative estimate on σ_r as

$$\sigma_r = \frac{\epsilon_0}{2\sqrt{3}(1 - \sqrt{1 - P_d(\epsilon_0)})} \tag{16}$$

($\sigma_r \leq 1$ should be imposed), which leads to an estimate on the coefficient of determination

$$D_\delta = 1 - \sigma_r^2. \tag{17}$$

If dependencies are predominantly linear, D_δ should approximate D_{MR}. The amount of D_δ being larger than D_{MR} reflects the degree of nonlinearity in the data.

4 Variable Ordering

While the ordering of the variables will not affect the dependency on the set of variables as a whole, it affects the value of the dependability index on individual variables. There might also occur situations where the variables have very different distributions, and a "wrong" ordering of a particular variable before the others can severely cut the available statistics and render the test unable to detect dependencies on the subsequent variables. In all these cases one should run the δ-test on a number of different variable orderings in order to gain a better picture of the dependency structure.

The lack of statistics often becomes evident when a large number ($\gtrsim 10$) of variables are to be analysed. The following heuristics may be helpful to reduce the number of reorderings needed and to improve the statistics.

1. With an arbitrary ordering run the δ-test in the SVA (single variable analysis) mode. The resulting $\overline{\lambda}_d$ is a measure of certain nonlinear correlation between the dependent and the d:th variable.

2. Divide the variables into subgroups according to the size of the $\overline{\lambda}_d$ (SVA). The variables in each subgroup, having similar value of $\overline{\lambda}_d$ (SVA) or other characteristics, are likely to be mutually correlated.

3. Turn off SVA and run δ-test on each of the subgroup of variables (a few reordering may again be worth experimenting with), the variables having insignificant $\overline{\lambda}_d$ may be regarded as irrelevant and removed from further consideration.

4. The remaining variables may then be ordered according to descending values of $\overline{\lambda}_d$ (SVA). Run δ-test again and eliminate *irrelevant* variables having diminishing $\overline{\lambda}_d$. One should however inspect the figures to make sure that the zero dependability index is not simply because the very lack of statistics. The variable elimination is not sustained if figures of the type fig. 1a show unacceptable statistics in the plateau region.

5. In general one should not allow a front variable to cut statistics to the extent that the analysis of the subsequent variable becomes impossible. Whenever this happens reorder this variable to be the very last. This may enable one or two irrelevant variables to slip through, but it avoids the elimination of relevant variables due to low statistics.

5 Restrictions

The delta test is a statistical method and it relies on accumulating good statistics. Obviously the method will fail if the data are very few. The minimum number of N depends on the individual problem. Generally the program should not be used for $N \lesssim 100$. The upper limit of N is subject to the available space for memory allocation and also the user's tolerance on CPU consumption.

6 Program Installation

Most of the information in this section can be found in `delta.readme`.

The entire code is packed in `delta.tar.Z`. The program, which is self-contained, can be compiled with two options:

- An X version which generates useful interactive graphics. This option requires a `X11` system.
- A "plain" version which produces no graphics.

The installation takes the steps as follows.

6.1 Unpacking

The ftp transmission of `delta.tar.Z` must have been done in the binary mode. The `UNIX` utilities `zcat` or `uncompress` and `tar` must be available for unpacking:

```
% zcat delta.tar.Z | tar xvf -
% cd delta
```

6.2 Compiling with `Xplot`

`Xplot` is an X-Window based graphic package written by Anders Nilsson. `Delta 2.0` uses `Xplot` to produce interactive graphics of the type in fig. 1.

6.2.1 Installation of `Xplot`

The executable codes of `Xplot` should be placed in a permanent directory. Edit the directory path in `Imakefile.Xplot` to specify where `Xplot` will be stored, and then do the following

```
% cp Imakefile.Xplot Imakefile
% xmkmf
% make
```

6.2.2 Compilation of Delta

If the previous step is successful, an executable `Xplot` should have been produced in the given directory. This same directory path must then be written in `Imakefile.delta`. Edit the directory path for `Xplot` in `Imakefile.delta` and perform

```
% cp Imakefile.delta Imakefile
% xmkmf
% make
```

A successful compiling will produce an executable `delta`. At this stage one may wish to test the installation of `Xplot` by (assuming your C compiler is referred to by `cc`)

```
% cc Xexample.c xplot.o
% a.out
```

If this fails to generate some graphics, `Xplot` does not function on your platform and you will have to either run delta with the -x option or recompile the program without Xplot.

6.3 Compiling without Xplot

If a X11 system is not available or if for some reason step 2 does not work Delta 2.0 has to be compiled without the Xplot graphics.

Set XWXPLOT to 0 in Imakefile.delta, and then do

```
% cp Imakefile.delta Imakefile
% xmkmf
% make
```

6.4 The Alternative to Imake

If xmkmf is unavailable or if it fails in the above procedures, ignore the Imakefiles and use Makefile.orig instead. Copy Makefile.orig to Makefile and read it carefully, follow the instructions to make changes if necessary, and then do

```
% make clean
% make Xplot
% make deltax      (for the X version), or
% make delta      (for the plain version)
```

6.5 Clean up Files

Only the two files delta and delta.txt are needed for running the program. All others can be removed. This can be done by

```
% make -f Makefile.orig cleanall
```

7 Executing the Program

The program is executed by the command

```
% delta -[options] input_filename
```

where the options are

-set Creates an input file and assists in setting up the major input parameters. Instructions on formatting the data file are also given.
-run Runs δ-test if the input file has already been set.
-pro Processes the existing δ-test outputs (for graphic or numerical display) only, assuming a δ-test run has been done.
-x Used in combination with the options above to suppress the interactive X-plots. (This option exists in the X version only.)

Sample usages are shown in section 8.

8 Sample Applications

Executing the `Delta 2.0` involves: (1) preparing the raw data file, (2) setting up the input parameters and (3) interpreting the results. Rather than describing these steps in general, we demonstrate the procedure by applying the program to two problems: the **logistic map** and a **random series**. These problems represent two extremes with respect to being deterministic versus completely random. Also shown are the graphics output available for systems with `X11`.

8.1 Logistic Map

The Data File

We generate data from the logistic map $x_t = 4x_t(1 - x_{t-1})$. The data are stored in `logist.dat`. The format of the data file can be rather arbitrary with the general rule that the data must be organized in "lines" and "columns". A "line" is an entry terminated by the newline character (a carriage return). The data entries on each line must be separated by spaces. These data entries form "columns" with each column representing one variable. The first few lines of `logist.dat` look as follows:

```
     x_t          x_t-1         x_t - x_t-1
1  0.10492      0.26957E-01   0.77965E-01
2  0.37566      0.10492        0.27073
3  0.93815      0.37566        0.56250
4  0.23208      0.93815       -0.70607
5  0.71288      0.23208        0.48080
6  0.81872      0.71288        0.10584
7  0.59366      0.81872       -0.22507
8  0.96491      0.59366        0.37125
9  0.13543      0.96491       -0.82949
.    .             .              .
.    .             .              .
```

The Input File

There are problem specific parameters that must be set before execution. These parameters are stored in an input file, which is named `logist.inp` in this example.

Two options can be used to run delta:

1. If the input file does not exist, use
   `% delta -set logist.inp`
   This option prompts the user for the mandatory inputs and generates the named input file.
2. If the input file exists and one only wishes to make minor changes to the parameters, edit the input file and run delta using
   `% delta -run logist.inp`

The `-x` option can be used in combination with either one of the above to suppress the `Xplot` graphics.

The file `logist.inp` generated by the `-set` option looks as follows.

```
datafile   logist.dat  --name of the data file
nskip      1  --# of header lines to be skipped in datafile
nline      1000  --# of data lines to fetch in (if 0, take all)
order[5]  200  201  202  203  204
/////  Don't forget to set the number of elements of order in []!! ///
/////  Normally only the above parameters need to be set by user ///
speriod    0  --1/0, if on, exclude periodic (repetitive) patterns from P(0|0).
sva        0  --1/0, single variable analysis mode on/off
velim      0  --1/0, variable elimination mode on/off
lammin    0.020000  --A variable is eliminated if Lambda<lammin and velim=1
mdout      51  --Binary output of P_d(eps|delta) suppressed for d>mdout
epsplot   0.500000  --The epsilon for which P_d(eps|delta) are to be plotted
minstat    50  --Minimum statistics for computed P_d to be acceptable
bineps     30  --# of bins in log10(epsilon)
bindelt    30  --# of bins in log10(delta)
epsmin    0.005000  --The lowest epsilon bin
epsmax    4.000000  --The highest epsilon bin
deltmin    0.000050  --The lowest delta bin
deltmax    4.000000  --The highest delta bin
//// (Lines beginning with // are comment lines.) ////
******** Adjust parameters above this line ***********
***The following parameters will automatically be set by
***Delta if order[] has been set properly.
ncol       2  --# of columns in datafile
niva       4  --# of independent variables
```

Only the first four parameters `datafile, nskip, nline, order[]` are mandatory. Among the others, apart from the occasional use of `velim` (*variable elimination option*) and `sva` (*single variable analysis*), there is in general no need for adjustment.

The array `order[`n`]` specifies the ordering of the variables, where n must be specified to reflect the total number of variables. `order[0]` always gives the dependent variable, and the others represent independent variables. A variable is indexed by its "column number" and "time lag" in the data file:

`order[] = (column number)`$\times$`100 + (time lag).`

In our example, all the variables refer to column 2, which is x_t. `order[1] = 201` means the column-2 variable with 1 time lag, and so on. Thus the variables specified by `order[]` in this example mean $\{x_t, x_{t-1}, x_{t-2}, x_{t-3}, x_{t-4}\}$.

The Output

`Delta 2.0` generates output to the terminal, to a set of output files, and to graphics displaying X-Windows. The graphs are shown in fig. 2 and fig. 3.

The following is the terminal output, which gives an explanation on the notations. The lines such as `[ P_2(200|201,202) ]` keep track on the set of variables involved in computing $P_d(\epsilon|\delta)$.

```
                   >>>> Delta Test, Version 2.0(X) <<<<


 Delta writes to the following files:
   logist.bin  --Binary output for the full set of P_d(eps|delta)
   logist.ded  --ANSI output for a limited set of P_d(eps|delta)
   logist.dee  --P(eps), Lambda(eps), and others as a function of epsilon
   logist.del  --The Lambda indices (a copy of the screen output)
   logist.log  --A log file that collects miscellaneous items
 Notations:
   Lambda_d: The integrated dependability index for the d-th variable.
   Err_Lamb: Estimated error on Lambda_d.
   Sum_Lamb: The sum of the Lambda_i, i=1,2,...d.
   Err_sum:  Estimated error on the sum.
   LinCorr:  The simple linear correlation.
   CD_Lin:   The Coefficient of Determination for (linear) multiple regression.
   CD_Delt:  The Delta-test estimated Coefficient of Determination achievable
        by nonlinear models.

  1000 lines of data fetched from logist.dat.

 d Lambda_d Err_Lamb Sum_Lamb Err_sum  LinCorr   CD_Lin   CD_Delt
[ P_1(200|201) ]
 1  1.0000  0.0030   1.00000  0.00301   0.0271  0.00073  1.00000
[ P_2(200|201,202) ]
```

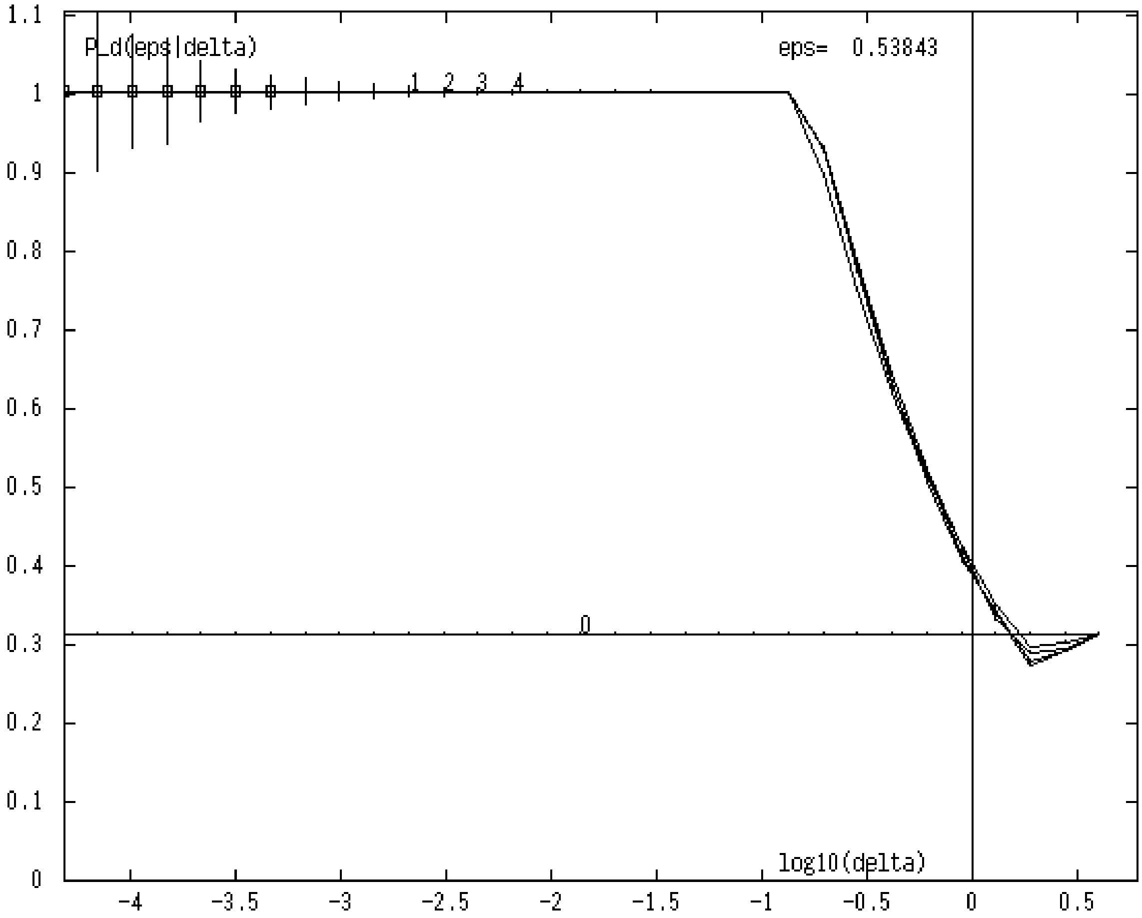

Figure 2: $P_d(\epsilon|\delta)$ as a function of $\log\delta$ for a fixed ϵ for the logistic map. The numbers marked on the curves are d. The low statistics points marked by squares have not been used for the purpose of identifying the maximum P_d.

```
 2  0.0000  0.0062   1.00000  0.00692   0.0275  0.00145  1.00000
[ P_3(200|201,202,203) ]
 3  0.0000  0.0091   1.00000  0.01142   0.0294  0.00223  1.00000
[ P_4(200|201,202,203,204) ]
 4 -0.0000  0.0097   1.00000  0.01499  -0.0427  0.00425  1.00000
```

The dependability index λ_d is 1 for $d = 1$ and 0 for the rest, clearly indicates that x_t depends on x_{t-1} only. The sum of λ_d equals to 1, indicating there is no noise in the function dependence. The same can also be said from the fact that `CD_Delt = 1` - a perfect determination coefficient. The fact that the coefficient of determination `CD_Lin` from the linear regression is much smaller than `CD_Delt` signals that the dependency is predominantly nonlinear.

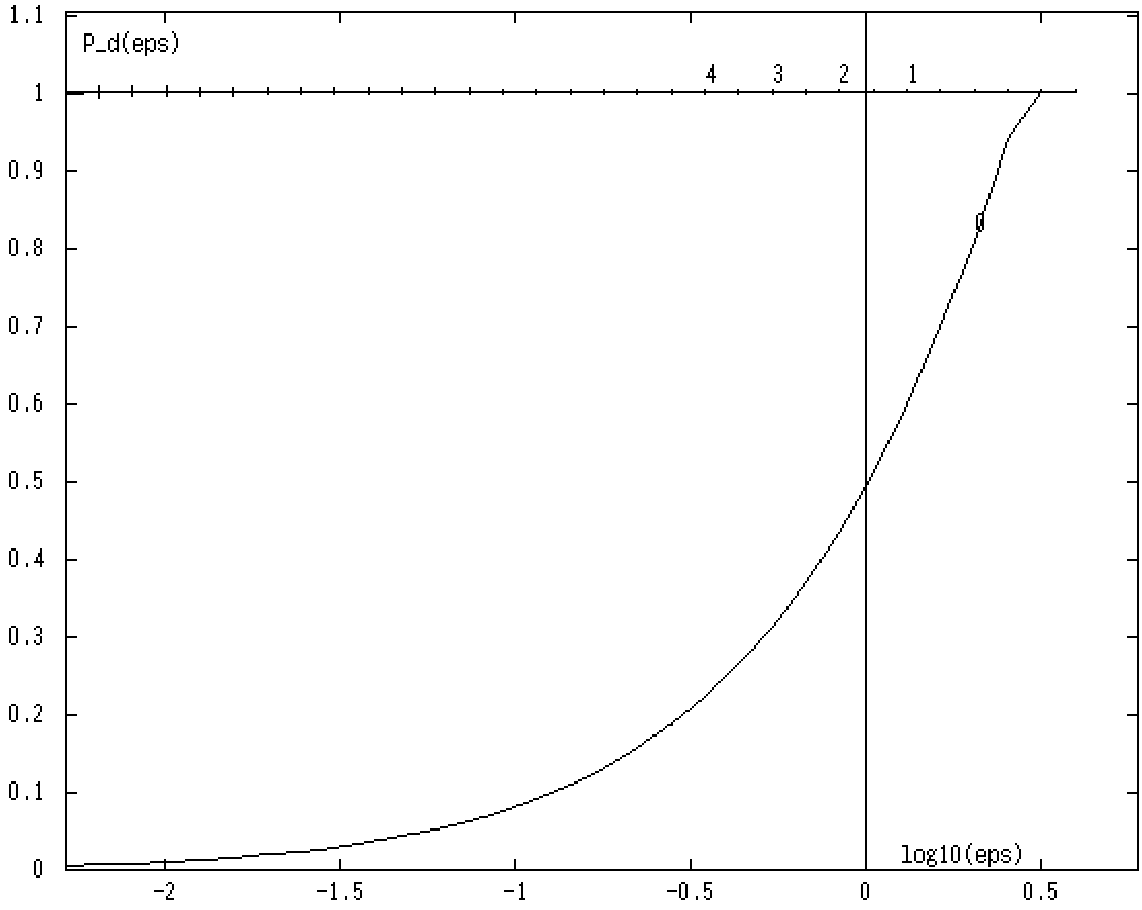

Figure 3: $P_d(\epsilon)$ (maximum of $P_d(\epsilon|\delta)$) as a function of $\log \epsilon$ for the logistic map. The variable index d is marked on the curves.

All the evaluated quantities are saved in files so that information can be re-extracted in the future without having to rerun the δ-test. If one uses the plain version, applying a graphic tool to the numerical tables in these files is also the only way to produce figures of the type fig. 2 and 3. We briefly describe the contents of the output files here.

logist.log contains the means and variances and the correlation matrix of the variables. It also reports the number of identical vector pairs. If this number is large as a fraction of the total pairs one must aware that the data may actually be periodical.

logist.del saves a copy of the terminal output.

logist.dee lists quantities as function of ϵ for each d. These quantities include $P_d(\epsilon)$ and its error estimate, which can be used to plot out fig. 3, $\lambda(\epsilon)$, the Grassberger-Procaccia correlation integral [2], and the dependence index as defined by Savit and Green [3].

logist.ded contains $P_d(\epsilon|\delta)$ versus ϵ and δ for a fixed d. This file is generally not useful unless one wishes to make a 3-D plot of P_d.

logist.bin is a binary record of the entire set of $P_d(\epsilon|\delta)$. Running `delta` with the `-pro` option, one can extract out $P_d(\epsilon|\delta)$ for a particular d and ϵ, the resulting numerical table can be used to plot out fig. 3.

8.2 Random Series

Next we illustrate `Delta 2.0` on a pseudo-random number series. Since the procedure is identical to the previous example we only list the steps.

The Data File

The data file is now named `rand.dat` and the first few lines are the following.

```
     x_t          x_t-1      x_t - x_t-1
1  0.47890E-02 -0.72961       0.73440
2  0.57268      0.47890E-02  0.56789
3  0.40136      0.57268     -0.17131
4  0.71001E-01  0.40136     -0.33036
5 -0.92903      0.71001E-01 -1.0000
.     .            .             .
.     .            .             .
```

The Input File

We denote the input file `rand.inp` and its top section is shown here (the rest of the file is identical to `logist.inp` shown before).

```
datafile   rand.dat  --name of the data file
nskip      0  --# of header lines to be skipped in datafile
nline      4000  --# of data lines to fetch in (if 0, take all)
order[5]  200  300  202  302  204
```

The specification of the `order[]` now involves the 3rd column. Since the column 3 in the data file is just the 1-lag variable x_{t-1}, the specification above is in fact equivalent to `order[] = {200, 201, 202, 203, 204}`.

The Output

Portions of the screen output is shown below.

```
             >>>>   Delta Test, Version 2.0(X)   <<<<

 Delta writes to the following files:
            ...

  4000 lines of data fetched from rand.dat.

 d Lambda_d Err_Lamb Sum_Lamb Err_sum  LinCorr   CD_Lin   CD_Delt
[ P_1(200|201) ]
 1  0.0128  0.0128   0.01277  0.01277   0.0051  0.00003  0.00000
[ P_2(200|201,202) ]
 2 -0.0059  0.0240   0.00692  0.02722  -0.0135  0.00021  0.00000
[ P_3(200|201,202,203) ]
 3  0.0035  0.0275   0.01046  0.03866  -0.0078  0.00027  0.00000
[ P_4(200|201,202,203,204) ]
 4  0.0218  0.0596   0.03231  0.07102  -0.0002  0.00027  0.00000
```

The dependability indices are all small, indicating there is essentially no dependency structure between the variables specified. The coefficients of determination are negligible for either linear or nonlinear models, consistent with the fact that the data is entirely noise.

The two graphs generated by `Delta` are shown in fig. 4 and fig. 5. All curves are very much lying on top of each other, reflecting the independence of the dependent variable on the independent variables.

Acknowledgements

The Göran Gustafsson Foundation for Research in Natural Science and Medicine and the Swedish Board for Industrial and Technical Development (NUTEK) are acknowledged for financial support. We thank A. Nilsson for giving us permission to use his `Xplot` graphic package. We are also grateful to M. Ohlsson and T. Rögnvaldsson for testing the program at various stages of its development.

References

[1] H. Pi and C. Peterson, "Finding the Embedding Dimension and Variable Dependencies in Time Series", *Neural Computation* **6**, 509 (1994).

[2] P. Grassberger and I. Procaccia, "Measuring the Strangeness of Strange Attractor", *Physica* **D9**, 189 (1983).

[3] R. Savit and M. Green, "Time Series and Dependent Variables", *Physica* **D50**, 95 (1991).

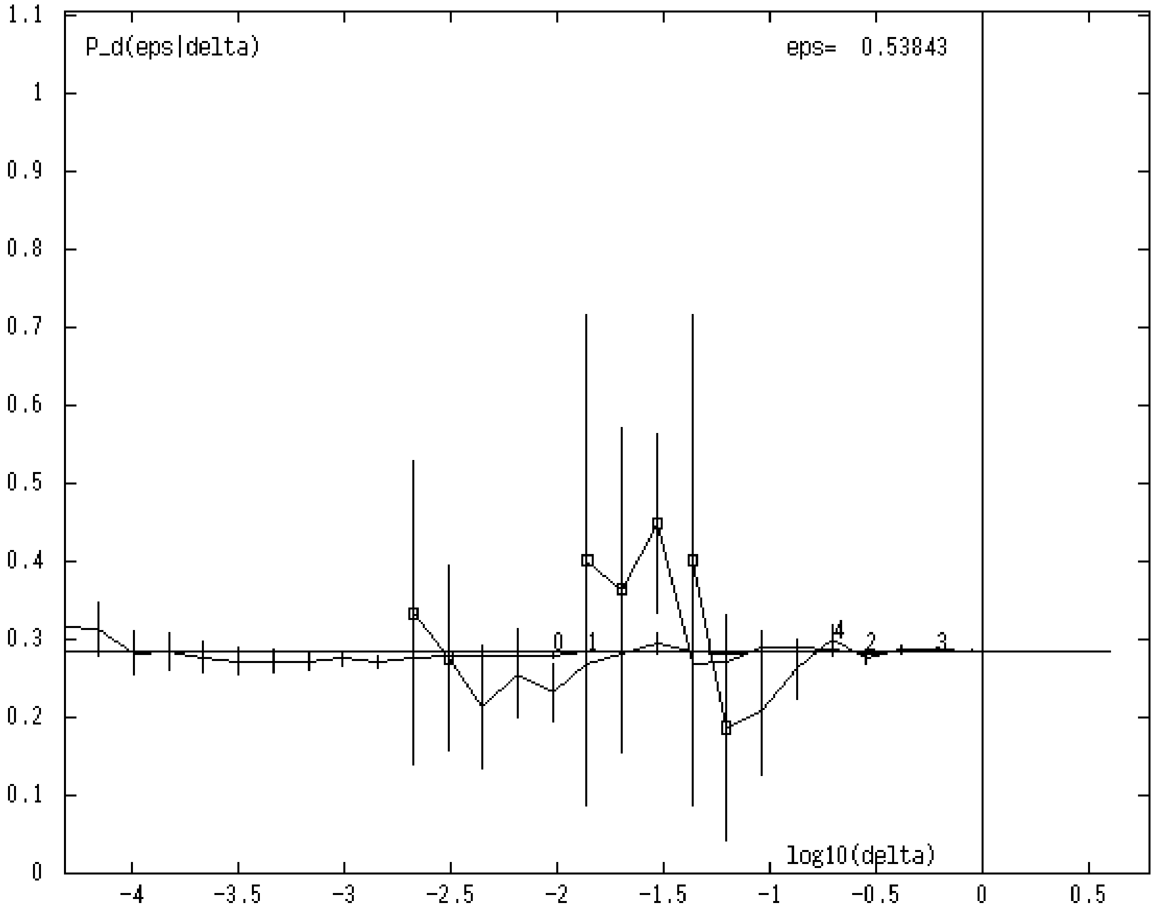

Figure 4: $P_d(\epsilon|\delta)$ as a function of $\log\delta$ for a fixed ϵ for the pseudo-random series. The numbers marked on the curves are d. The squares mark the low statistics points that have been ignored in the program.

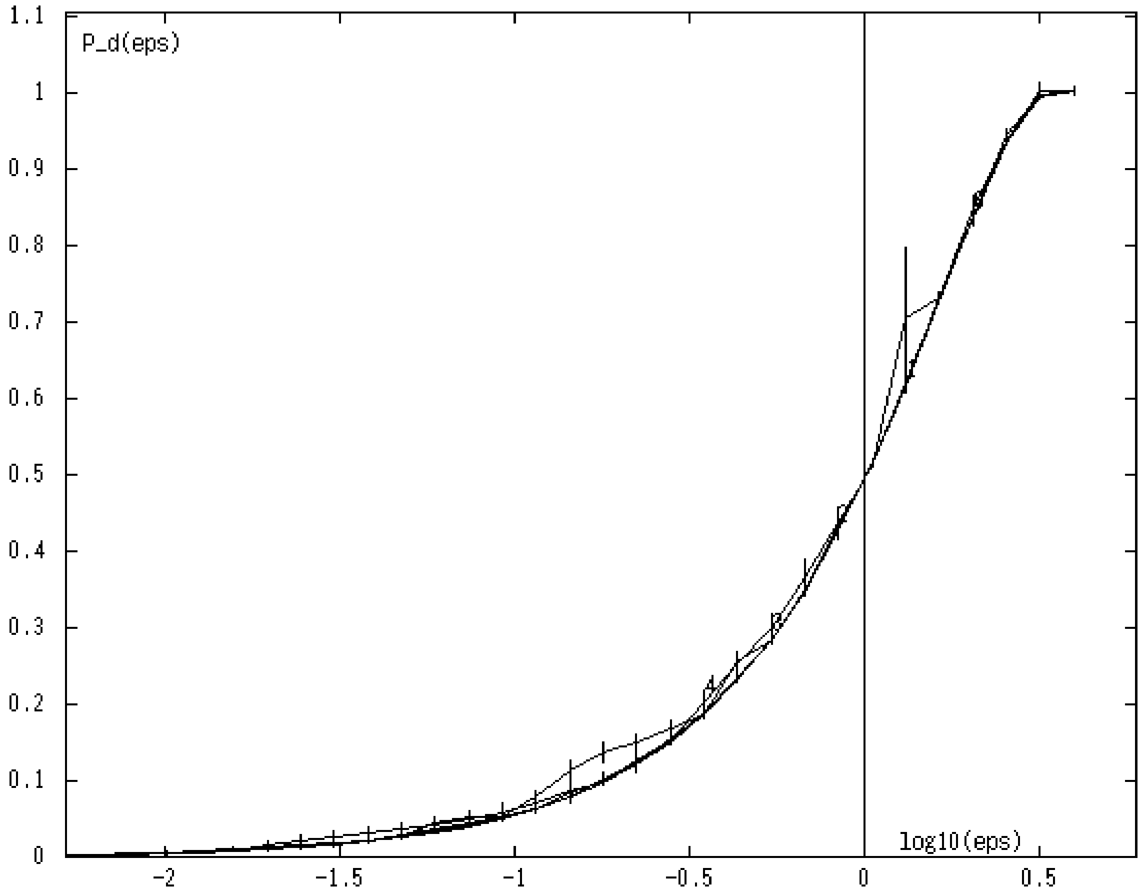

Figure 5: $P_d(\epsilon)$ (maximum of $P_d(\epsilon|\delta)$) as a function of $\log\epsilon$ for the pseudo-random series. The variable index d is marked on the curves.