



**WSG-RR 9/96**

**IFNN Manual:**

**Integrated Framework for Neural Network  
and Conventional Modelling**

*Dimitri Volkov, Adrian Trapletti and Manfred M. Fischer*

Institut für Wirtschafts-  
und Sozialgeographie

**Wirtschaftsuniversität  
Wien**

Department of Economic  
and Social Geography

**Vienna University of  
Economics and Business  
Administration**

**WSG-RR 9/96**

**IFNN Manual:**

**Integrated Framework for Neural Network  
and Conventional Modelling**

***Dimitri Volkov, Adrian Trapletti and Manfred M. Fischer***

**Abteilung für Theoretische und Angewandte Wirtschafts- und Sozialgeographie  
Institut für Wirtschafts- und Sozialgeographie  
Wirtschaftsuniversität Wien**

**Vorstand: o.Univ.Prof. Dr. Manfred M. Fischer  
A - 1090 Wien, Augasse 2-6, Tel. (0222) 313 36 - 4836**

**Redaktion: Mag. Petra Stauer**

**WSG-RR 9/96**

**IFNN Manual:**

**Integrated Framework for Neural Network  
and Conventional Modelling**

***Dimitri Volkov, Adrian Trapletti and Manfred M. Fischer***

**WSG-Research Report 9**

**June 1996**

# IFNN Manual

*Integrated Framework for Neural Network and Conventional  
Modelling*

Dimitri Volkov  
Adrian Trapletti  
Manfred M. Fischer

Department of Economic and Social Geography  
Vienna University of Economics and Business Administration

April 1996

This Manual is for IFNN Version 1.5  
Email address for comments, suggestions and bug reports:  
[adrian@wigeo1.wu-wien.ac.at](mailto:adrian@wigeo1.wu-wien.ac.at)

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>User's Guide</b>	<b>5</b>
2.1	Main Menu . . . . .	5
2.2	Input Module . . . . .	6
2.2.1	Input Data File . . . . .	6
2.2.2	Transformations . . . . .	9
2.2.3	Selection of Training, Validation and Testing Data Sets	13
2.2.4	Save Results . . . . .	18
2.3	Statistical Measures Module . . . . .	18
2.3.1	Standard Statistics . . . . .	19
2.3.2	Graphical Statistics . . . . .	21
2.3.3	Frequency Chart . . . . .	21
2.4	Neural Network Interface Module . . . . .	23
2.4.1	Utilities . . . . .	24
2.4.2	Interactive Mode . . . . .	27
2.4.3	Batch Mode . . . . .	28
2.5	Analysis Module . . . . .	29
2.6	Utilities . . . . .	32
2.7	Setup Module . . . . .	33
2.8	Quit . . . . .	35
<b>3</b>	<b>System Programming Guide</b>	<b>36</b>
3.1	Environment Variables . . . . .	36
3.2	Content of the IFNN Distribution . . . . .	36
3.3	IFNN Module Description . . . . .	36
3.3.1	Main programs . . . . .	36
3.3.2	Service subroutines . . . . .	37
3.3.3	Lower end service subroutines . . . . .	38
3.3.4	Headers . . . . .	38
3.3.5	Temporary files . . . . .	38
3.3.6	Other files . . . . .	38
3.3.7	Directories . . . . .	39
3.4	Precision Setting . . . . .	39
3.5	Interaction with External Programs . . . . .	39
3.6	Building IFNN and System Requirements . . . . .	39
3.7	Implementing Future Extensions . . . . .	40
	<b>Appendix</b>	<b>42</b>

<b>A IFNN Messages</b>	<b>42</b>
A.1 Error messages . . . . .	42
A.2 Information messages . . . . .	43
A.3 Messages from the X Windows shell . . . . .	44
A.4 Fatal error messages . . . . .	44
<b>B Example of an Input Data File</b>	<b>45</b>
<b>C References</b>	<b>46</b>

# 1 Introduction

The program package **IFNN (Integrated Framework for Neural Network Modelling)** is a special operational environment supporting the process of neural network modelling, mainly in the application domain areas of

- **spatial interaction modelling (SI),**
- **time series analysis (TS),**
- **pattern recognition (PR) and**
- **traveling salesman problems (TV).**

The program runs under UNIX and X-Windows with the OPEN LOOK graphical user interface. It combines a graphical input/output user interface with an interface to *NeuralWorks Professional II/PLUS* (NeuralWare, 1993) and provides robust and efficient algorithms supporting the process of neural network modelling. The package was implemented in C and has been tested in research situations on a local area network of Sun SPARC workstations at the Department of Economic and Social Geography of the Vienna University of Economics and Business Administration.

The computing environment consists of the following major process modules:

**Input Module:** Reads, checks, preprocesses and formats the input data obtained from the above domain areas and finally separates the data into training, validation and testing sets.

**Statistical Measures Module:** Calculates standard statistics and provides an interface to external graphical representation programs used for graphical statistics.

**Neural Network Interface Module:** Provides an interface and some useful tools to work with *NeuralWorks Professional II/PLUS*, which allows the user to create, learn and test suitable neural network architectures in each domain of application.

**Analysis Module:** Provides tools for analyzing the simulation results from the previous module with suitable statistical performance measures.

The IFNN modules reflect the basic process of developing a neural network application which consists of the following major steps:

1. **Data collection and preprocessing.**
2. **Separation of the data** into training and testing sets or into training, validation and testing sets (for cross-validation purposes).

3. **Choice of the model architecture.**
4. **Training and testing the chosen model.**
5. Repeating steps 1 to 4 as required.

The structure of the manual is as follows. Section 2 is written as a user's guide, containing some background information and information on how to use IFNN in the context of an application. Section 3 serves to provide information for the system programmer, who likes to understand the IFNN implementation from scratch. The appendix contains a list of all error messages from IFNN, an example of an input data file and the reference list.

In this documentation **typewriter** style is used for user and IFNN input and output, syntax specification, commands and file contents. *Italic* shape is used for options to commands and for names (e.g. file, directory, widget names). The **Boldface** series is used to emphasize text. Syntax specification is given in EBNF (Extended Backus-Naur-Form).



## 2 User's Guide

### 2.1 Main Menu

To start IFNN, type `ifnn` at the UNIX shell prompt, and press the *Return* key. Then the `mainmenu` (see figure 1) is opened on the screen.

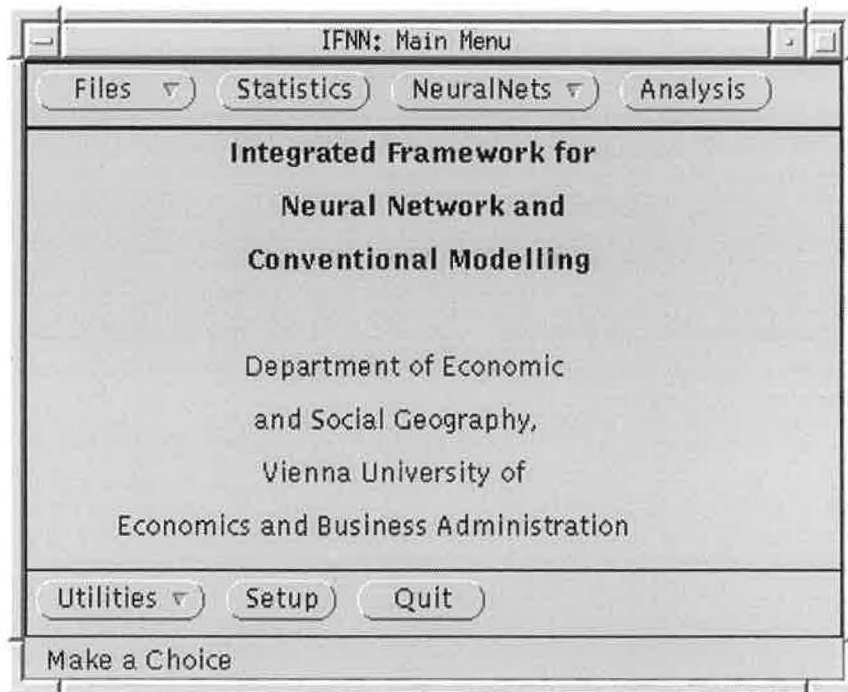



Figure 1: Main menu

You start the major process modules by clicking on the *Files*, *Statistics*, *NeuralNets* or *Analysis* button. To open the window for initializing the seed values for the build-in random number generator, use the *Utilities* button. The configuration window for the environment variables of IFNN is opened by clicking on the *Setup* button. To exit from IFNN use the *Quit* button and confirm.

The IFNN windows usually have two buttons to go back to the previous level of the dialogue: *OK* with executing the desired actions and *Cancel* without taking any actions.

The help utility is available at any time of the dialogue and provides a description about the next working steps or some user errors. You start help either by using the *Help* or *F1* key of the keyboard. An example of the *Help* window is shown in figure 2.

The general use of the mouse buttons is as follows: The right mouse button is used for buttons concerned with popup menus like ,

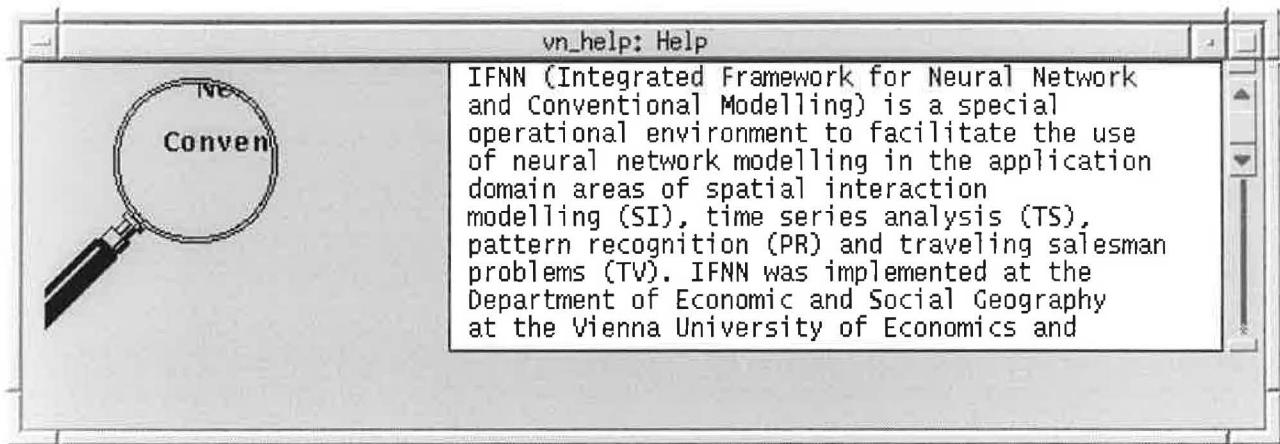


Figure 2: Example of the *Help* window

the left mouse button for conventional buttons like **Statistics**, and the middle mouse button doesn't have any function.

## 2.2 Input Module

The popup menu associated with the button *Files* contains the following items:

- *INPUT DATA FILE*
- *TRANSFORMATIONS*
- *TRAIN/VAL/TEST*
- *SAVE RESULTS*

### 2.2.1 Input Data File

You specify the type, the name and the format of the input data in the *Input Data File* window (see figure 3).

The popup menu *Application Domain* denotes the type of the input data and contains the following items:

- SI:** Spatial Interaction,
- PR:** Pattern Recognition,
- TS:** Time Series,
- TV:** Traveling Salesman.

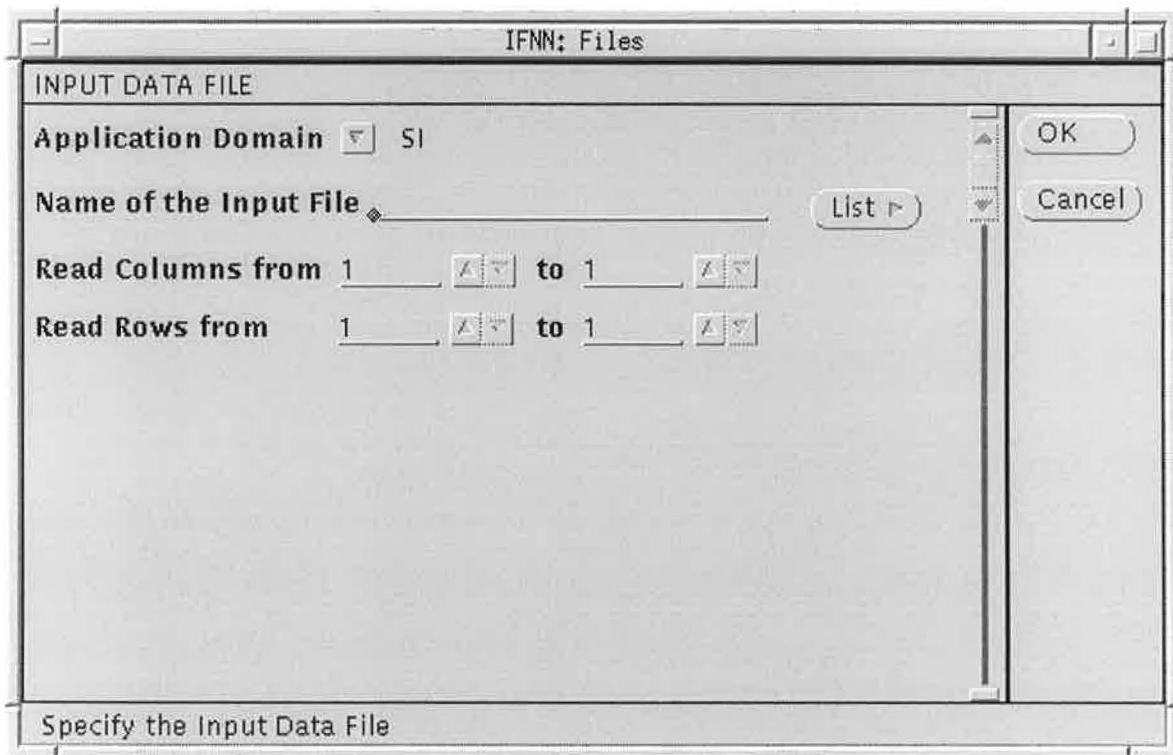


Figure 3: *Input Data File* window (screen 1)

The selected type of the input data is used to construct the future dialogue scenarios of IFNN.

The input data file must be a rectangular matrix of elements in ASCII format. Its syntax is defined by

```
file   = { row }.
row    = ["#"] { column } endofline.
column = ( integer | float | character ).
```

After reading the input data file, IFNN checks for errors in the data (e.g. the data may contain a character in the middle of a numeric field like 0.05467). All rows of the input file beginning with a # symbol (comment) are skipped.

You have also the flexibility to choose only a rectangular submatrix. This is especially useful when you work with a huge dataset and you need only some part of the data. An example of an input data file is presented in appendix B.

Choose the file name either by typing the name with the keyboard or by using the popup menu *List* and selecting the file name from the list of all files located in the current directory or in a subdirectory.

To select only a part of the data file, change the default values of the fields *Read Columns from to* and *Read Rows from to*, equal to the number of columns and rows in the raw input data file, to the desired values. If you click

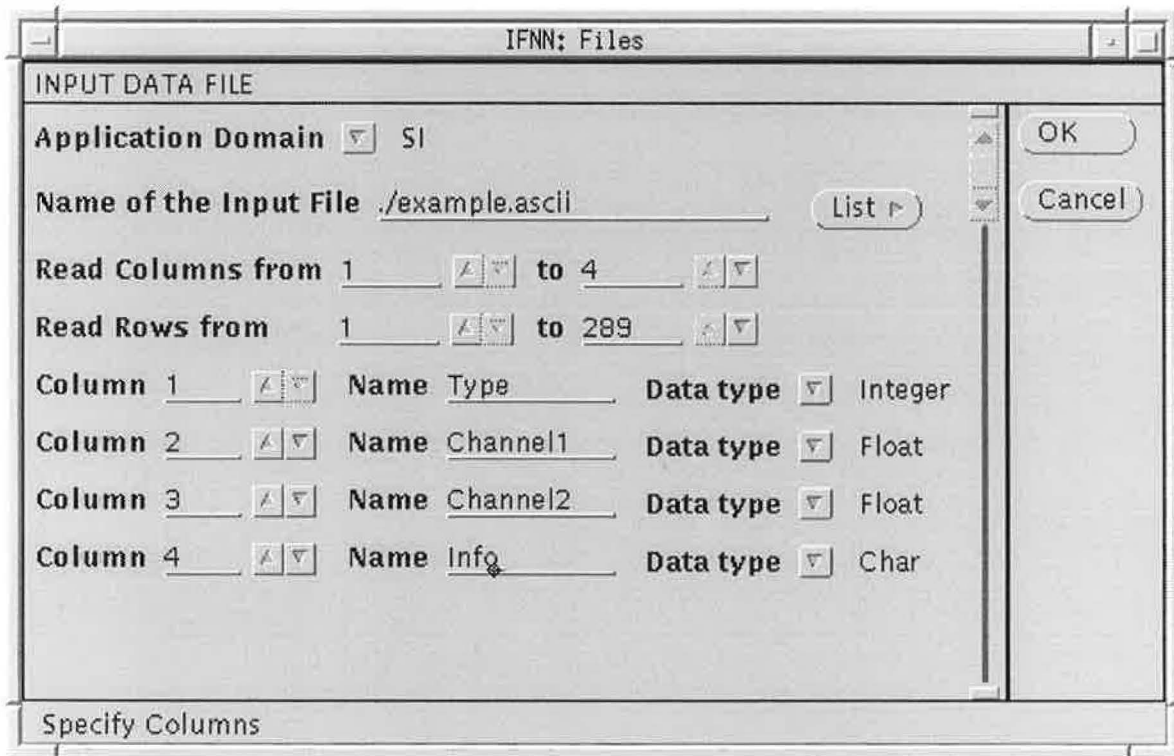


Figure 4: *Input Data File* window(screen 2)

on *OK*, a series of new fields appears in the window, each field describing one column by number, name (identification) and data type (float, integer or character). The window at this level of the dialogue is shown in figure 4. The columns of the input data correspond to the variables, and the rows correspond to the observations of the variables (note that the number of observations should be equal over all variables). After setting all these fields you click again on *OK*, and the program checks all the data on inconsistencies according to the specifications, and if the file is correct, an *OK* window is opened (see figure 5). In this window the overall number of columns and rows, which have been read from the input file, are displayed. To return to the main menu click *OK*.

If IFNN detects inconsistencies, an *ERROR* window is opened (see figure 6). You find a full list of the error codes in appendix A. To respecify the input data in the previous window, use the *No* button. If you click on the *Yes* button, the *Correct Errors* window is opened, where you can correct the errors. The location of the first error is usually indicated by a special

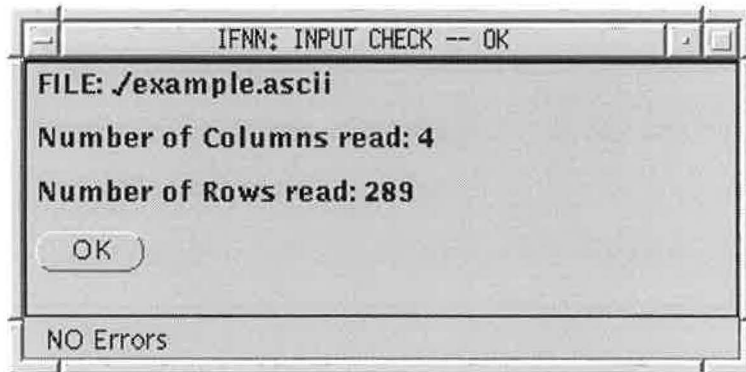


Figure 5: *OK* window

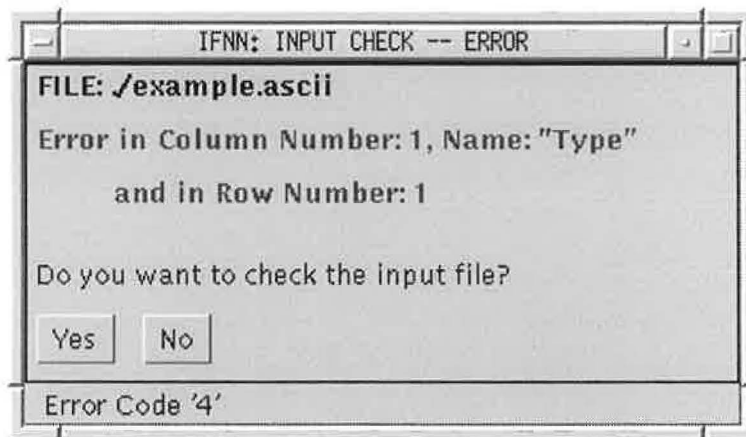


Figure 6: *ERROR* window

mark. After correcting the errors in the input data file you can either save all changes with the *Save* button, or cancel all changes and stay in the *Correct Errors* window with the *Cancel* button, or return to the *Input Data File* window with the *Return* button.

### 2.2.2 Transformations

An important step in the process of neural network modelling is to transform the raw data into a neural network appropriate format. IFNN provides different transformation techniques relevant for the main application domain areas. You can also apply a sequence of different transformations. An example of the *Transformations* window for time series data is shown in figure 7. To transform a variable, you select this variable on the left side panel with the left mouse button. On the right side you choose the desired transformation from the shown list of transformations. The bottom panel shows the already executed transformations. To start a transformation use

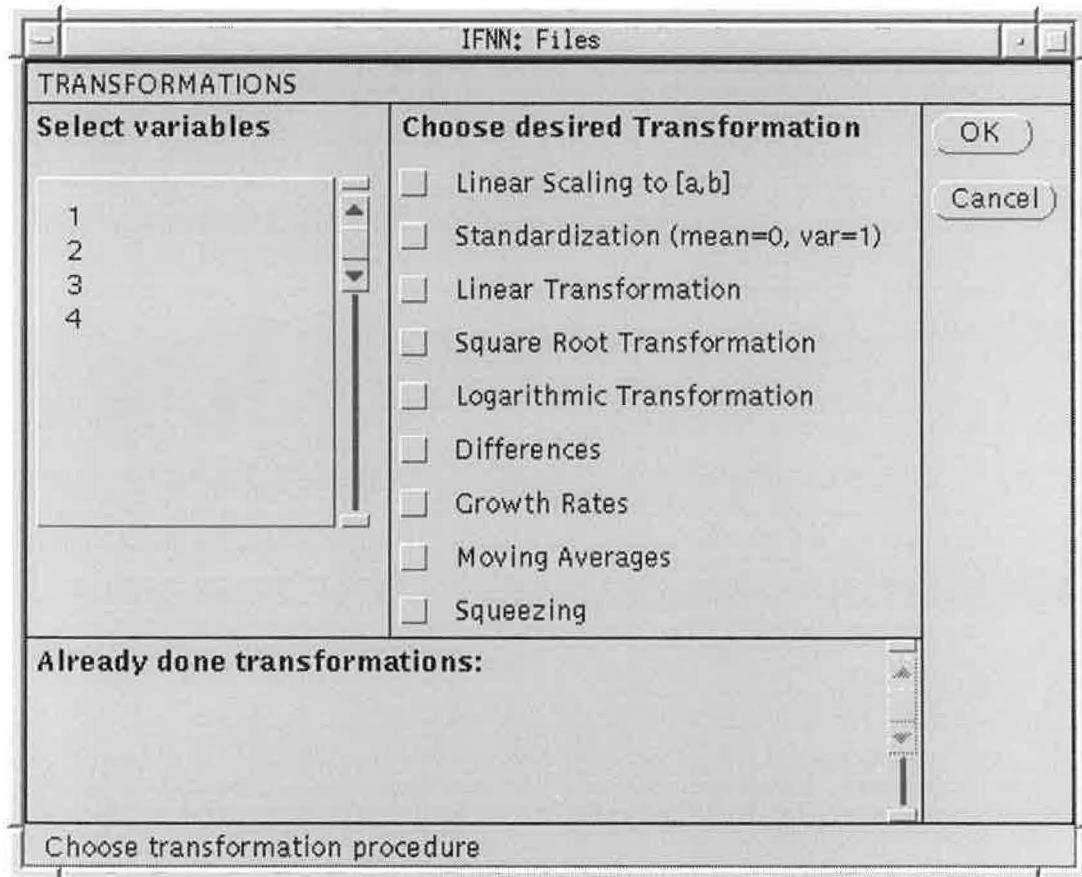


Figure 7: *Transformations* window

the *OK* button, and to go back to the previous window use the *Cancel* button.

In the sequel a definition of all provided transformations is given. The following notation is used to define the transformations:  $k = 1, \dots, K$  denotes the index for rows (observations), and  $l = 1, \dots, L$  the index for columns (variables).  $X_{kl}$  denotes the  $k$ -th observation of the original  $l$ -th variable, and  $Y_{kl}$  the  $k$ -th observation of the transformed  $l$ -th variable.

1. **Linear Scaling to  $[a, b]$ :**

$$Y_{kl} = \frac{(b - a)X_{kl} - b m_l + a M_l}{M_l - m_l},$$

where  $m_l = \min_k(X_{kl})$ ,  $M_l = \max_k(X_{kl})$ , and the values of  $a$  and  $b$  can be:

- $a = 0.0, b = 1.0,$
- $a = 0.1, b = 0.9,$
- $a = -1.0, b = 1.0,$
- $a = -0.9, b = 0.9,$
- user defined  $a$  and  $b.$

2. **Standardization:**

$$Y_{kl} = \frac{X_{kl} - \mu_l}{\sigma_l},$$

where  $\mu_l = \frac{1}{K} \sum_k X_{kl}$  is the mean and  $\sigma_l = \sqrt{\frac{1}{K-1} \sum_k (X_{kl} - \mu_l)^2}$  is the standard deviation of the  $l$ -th variable. As a side effect of this transformation, IFNN saves the global scaling parameters  $g_l = \frac{1}{\sigma_l}$  and  $i_l = -\frac{\mu_l}{\sigma_l}$  for each variable. You can use this parameters in the *Linear Transformation*.

3. **Linear Transformation** (see figure 8):

$$Y_{kl} = g_l X_{kl} + i_l,$$

where  $g_l$  and  $i_l$  denote the global scaling parameters. These are set by IFNN in a previous transformation, and can be used to linearly scale the data set with the scaling parameters from a previous transformation. To select some of the saved values, click on the *Saved Values* box, mark the desired values with the mouse (right button), and click on the *Get g* or *Get i* button. The selected values appear at the bottom pannel of the window. If you select more than one value, each value is used by IFNN to scale the same variable as in the previous transformation. You can also specify your own values by using the *Other Values* box and the  $g=$  and  $i=$  fields.

4. **Square Root Transformation:**

$$Y_{kl} = \sqrt{X_{kl}}.$$

5. **Logarithmic Transformation:**

$$Y_{kl} = \ln(X_{kl}),$$

where  $\ln$  denotes the natural logarithm.

6. **Differences** (time series specific transformation):

$$Y_{kl} = X_{kl} - X_{(k-\Delta k)l}, \quad k = \Delta k + 1, \dots, K,$$

where you specify  $\Delta k$  in a separate window.

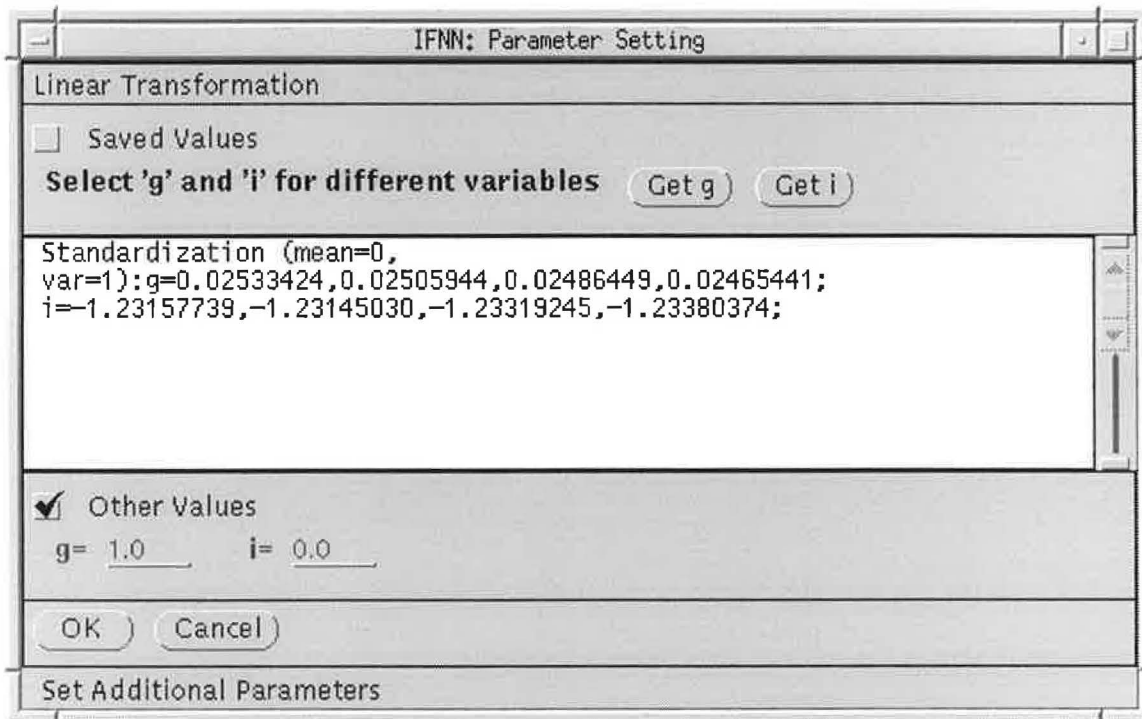


Figure 8: *Linear Transformation* window

7. **Growth Rates** (time series specific transformation):

$$Y_{kl} = \ln(X_{kl}) - \ln(X_{(k-1)l}), \quad k = 2, \dots, K.$$

8. **Moving Averages** (time series specific transformation):

$$Y_{kl} = \begin{cases} \frac{1}{a}[X_{(k-\frac{a-1}{2})l} + \dots + X_{kl} + \dots + X_{(k+\frac{a-1}{2})l}] & \text{if } a \text{ is odd,} \\ \frac{1}{a}[X_{(k-\frac{a}{2})l} + \dots + X_{kl} + \dots + X_{(k+\frac{a}{2}-1)l}] & \text{otherwise,} \end{cases}$$

$$k = \begin{cases} \frac{a-1}{2} + 1, \dots, K - \frac{a-1}{2} & \text{if } a \text{ is odd,} \\ \frac{a}{2} + 1, \dots, K - \frac{a}{2} + 1 & \text{otherwise,} \end{cases}$$

where you specify  $a$  in a separate window.

9. **Squeezing**:

$$Y_{kl} = b \ln(X_{kl} + c) + a,$$

where you specify  $a$ ,  $b$  and  $c$  in a separate window, and the condition  $X_{kl} + c > 0$  must be satisfied.

The concrete content of the list of transformations depends on the setting, which you can change in the setup module (see section 2.7) for each type of the data. The default list is:



- For SI: 1), 2), 3), 4), 5), 9).
- For PR: 1), 2), 3), 4), 5), 9).
- For TS: 1), 2), 3), 4), 5), 6), 7), 8), 9).
- For TV: 1), 2), 3), 4), 5), 9).

After each transformation without errors an *OK* window is displayed. In the case of an error an *ERROR* window is displayed, where the transformation causing the error is marked. A list of the error codes is given in appendix A.

### 2.2.3 Selection of Training, Validation and Testing Data Sets

In principle there are two major strategies to tackle the overfitting problem: the cross-validation and the pruning strategies. In the first case we need to separate the data set into three subsets: the training, validation and testing data set. In the second case we need to separate the data set into training and testing data sets only.

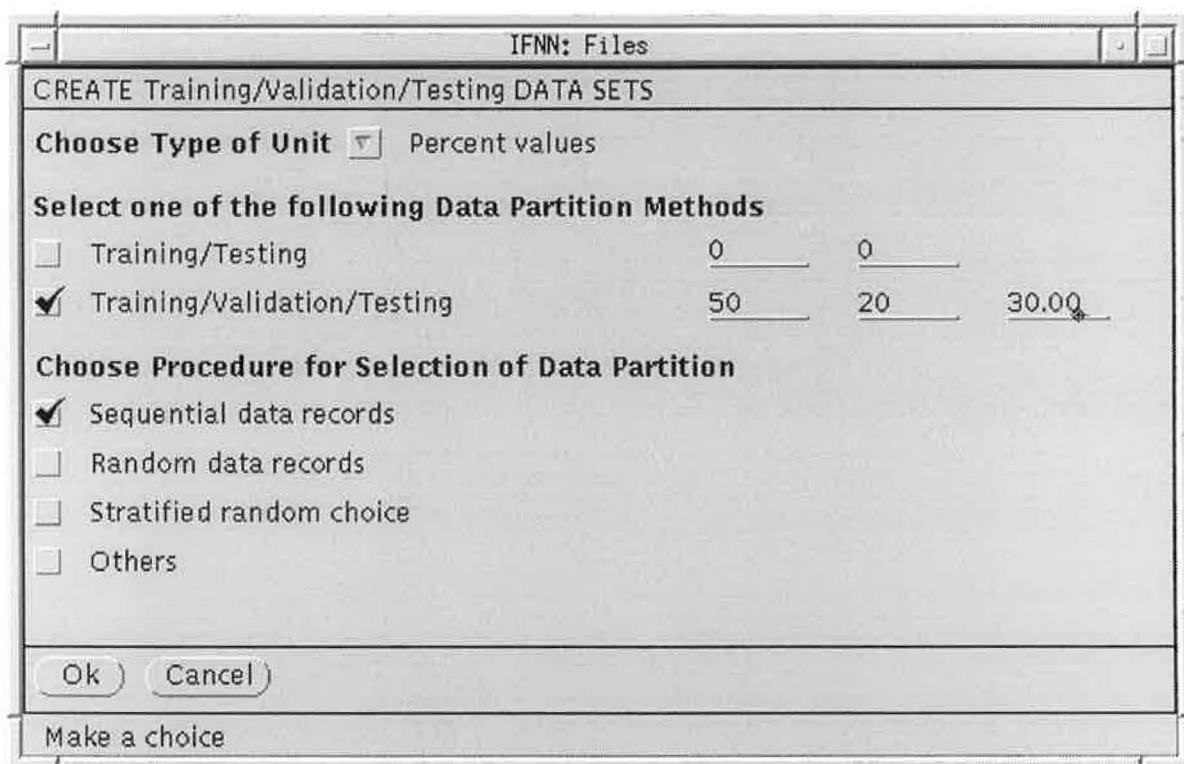


Figure 9: Create training, validation and testing data sets

The user interface for training and testing (or training, validation and testing) data set selection is shown in figure 9. To create the training,

validation and testing data sets, you specify the type of unit for the size of the data (i.e. absolute or percent values), the size of the data itself, and the procedure for the data partition. The current data is used as input and the resulting data sets are saved as files. By convention, IFNN uses `fn "_train.nna"`, `fn "_valid.nna"` and `fn "_test.nna"` as the filenames for the training, validation and testing data sets, where `fn` denotes the filename from the original input data file. To specify the size of the data sets you can use percentage or absolute values (i.e. the number of data records). If you hit the *Return* key instead of typing a number in the last of the size fields, IFNN will complete the specification, i.e. sum up to 100% or to the total number of data records.

As procedures for the data partitioning, you can choose between the following methods:

- **Sequential data records** reads and writes sequentially record by record from the current data to the set files. For example in figure 9, the first 50% of records are written to the training, the next 20% of records to the validation, and the remaining 30% of records to the testing data files in sequential order from the current data.
- **Random data records** means that the algorithm randomly (without replacement) picks a certain percentage of the entire data set for the training, validation and testing data files (e.g. 50% for the training, 20% for the validation and 30% for the testing files).
- **Stratified random choice** is suggested for supervised classification tasks (e.g. remote sensing applications). You have to specify a priori the number of classes and the number of records in each class in an additional window. If the total number of records for all classes in the specification is not equal to the total number of data records, IFNN displays an error message. For example, if you select 60% and 40% for the training and testing files, and 4 classes with 40, 30, 25 and 5 records (total of 100 records), then the random (without replacement) selection of the data records by IFNN for the set files is

Classes	Count	Train	Test
class 1	40	24	16
class 2	30	18	12
class 3	25	15	10
class 4	5	3	2
Total	100	60	40

For **time series data sets (TS)**, another window template (see figure 10) is used by IFNN for the training, validation and testing data set specification.

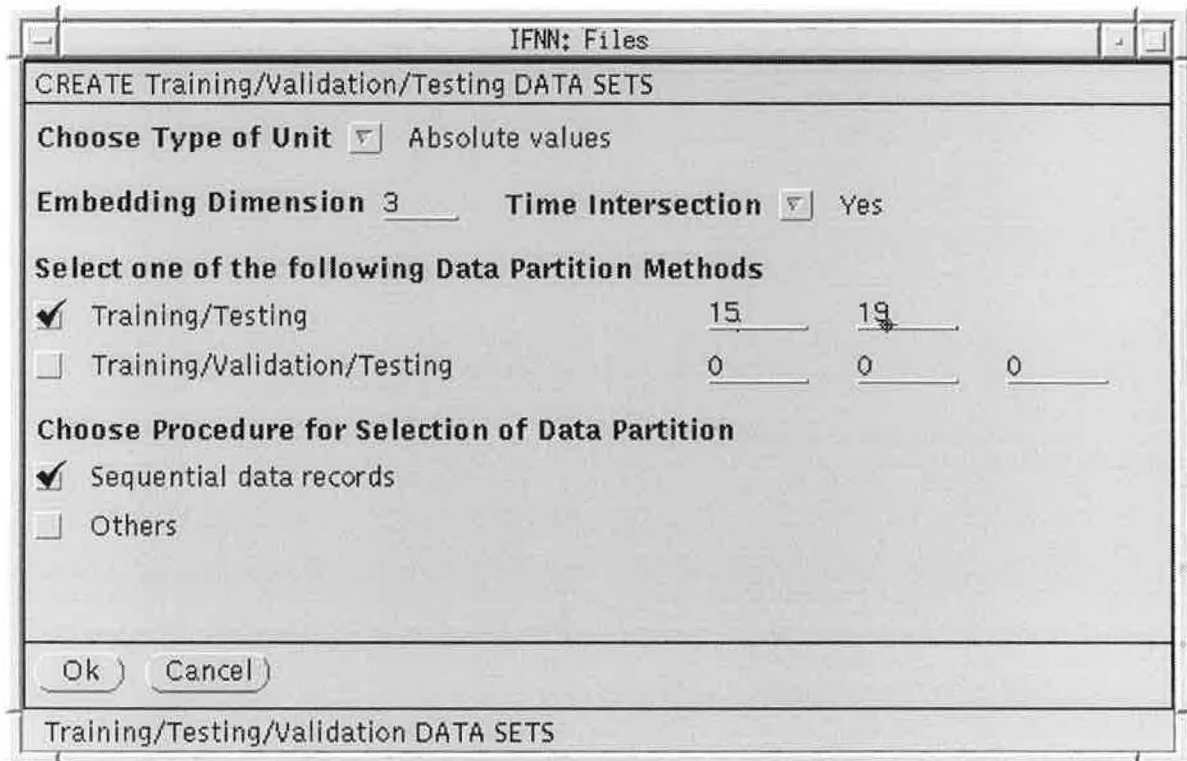


Figure 10: Create training, validation and testing data sets for time series

A time series typically consists of a set of observations on a variable,  $y$ , taken at equally spaced intervals over time. A series of  $T$  observations will be denoted by  $y_1, \dots, y_T$ . Modelling a time series can be described formally as follows: Find a function  $\mathbf{F} : \mathcal{R}^p \rightarrow \mathcal{R}^q$  such as to obtain an estimate of the output-vector  $(y_t, y_{t+1}, \dots, y_{t+q-1})$  in the  $q$ -dimensional output space, given the input-vector  $(y_{t-1}, y_{t-2}, \dots, y_{t-p})$  in the  $p$ -dimensional input space:

$$(\hat{y}_t, \hat{y}_{t+1}, \dots, \hat{y}_{t+q-1}) = \mathbf{F}(y_{t-1}, y_{t-2}, \dots, y_{t-p}).$$

The procedure of building the input- and output vectors is called embedding, and the number  $p + q$  denotes the embedding dimension. Time intersection means using overlapping intervals at the borders between training, validation and testing data sets.

Using the specifications of figure 10 would result in the following input and output files:

1. input file *exampleTS.ascii*:

5  
11

16  
23  
36  
58  
29  
20  
10  
8  
3  
0  
0  
2  
11  
27  
47  
63  
60  
39  
28  
26  
22  
11  
21  
40  
78  
122  
103  
73  
47  
35  
11  
5  
16  
34

2. output file *exampleTS\_train.nna*:

5.000000 11.000000 16.000000  
11.000000 16.000000 23.000000  
16.000000 23.000000 36.000000  
23.000000 36.000000 58.000000  
36.000000 58.000000 29.000000  
58.000000 29.000000 20.000000  
29.000000 20.000000 10.000000

```
20.000000 10.000000 8.000000
10.000000 8.000000 3.000000
8.000000 3.000000 0.000000
3.000000 0.000000 0.000000
0.000000 0.000000 2.000000
0.000000 2.000000 11.000000
2.000000 11.000000 27.000000
11.000000 27.000000 47.000000
```

3. output file *exampleTS\_test.nna*:

```
27.000000 47.000000 63.000000
47.000000 63.000000 60.000000
63.000000 60.000000 39.000000
60.000000 39.000000 28.000000
39.000000 28.000000 26.000000
28.000000 26.000000 22.000000
26.000000 22.000000 11.000000
22.000000 11.000000 21.000000
11.000000 21.000000 40.000000
21.000000 40.000000 78.000000
40.000000 78.000000 122.000000
78.000000 122.000000 103.000000
122.000000 103.000000 73.000000
103.000000 73.000000 47.000000
73.000000 47.000000 35.000000
47.000000 35.000000 11.000000
35.000000 11.000000 5.000000
11.000000 5.000000 16.000000
5.000000 16.000000 34.000000
```

As procedures for the data partitioning of time series, you can choose between the following methods:

- ***Sequential data records*** reads and writes sequentially record by record from the current data to the set files as described in the example above.
- ***Others*** is used to call an external program, which transforms and partitions the data. The window for this option is shown in figure 11. You specify a file name, a command and some options in the fields *Write Data to File*, *Call Command* and *Options*. IFNN saves the current state of the internal data (e.g. if you transformed the data, IFNN takes the transformed data) to the specified file in the current working directory. Before calling the external program IFNN creates the *\*.nna*

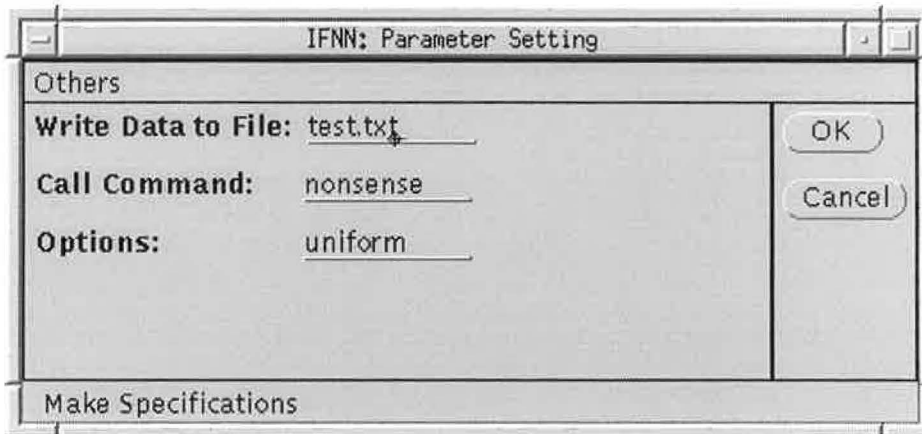


Figure 11: *Others* option

files as above explained for the *Sequential data records*. Then IFNN executes the UNIX command line

```
command fn ["PV"|"AV"] n1 n2 n3 "ED"n "DRI"["Y"|"N"] option,
```

where `command` is the content of the field *Call Command*, `fn` the content of the field *Write Data to File*, and `option` the content of the field *Options*. If you specified percentage values, "PV" is taken, else "AV". `n1`, `n2` and `n3` are the specified sizes of the training, validation and testing data sets. `n` is the embedding dimension, and "Y" is for time intersection, else IFNN takes "N".

#### 2.2.4 Save Results

You can save the whole internal data set, e.g. obtained after transformations, to a single file by using the item *SAVE RESULTS* of the input module. You have to specify a file name and click on *OK*. If the file already exists, a warning message is displayed, and you have to confirm or cancel the operation.

### 2.3 Statistical Measures Module

The statistical measures module provides standard statistics and an interface to external graphical representation programs used for graphical statistics. You can choose the desired type of statistics in the window shown in figure 12. You start the procedures for the different type of statistics with the buttons *Standard Statistics*, *Graphical Statistics* and *Frequency*.

**Standard statistics** provides some useful statistics like the mean and the variance of a variable, or the covariance between two variables and so on. The definitions of these measures are given below.

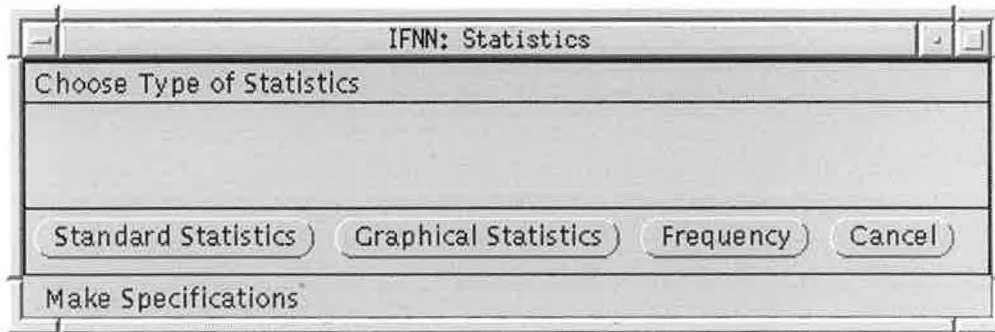


Figure 12: Choosing type of statistics

**Graphical statistics** is an interface to an external graphical representation program. IFNN supports GNUPLOT and ACE/gr. You can define a default for the graphical representation program in the setup module (see section 2.7). It allows you to create graphical representations of the data.

The **frequency procedure** provides the calculation and the graphical representation of the empirical distribution of the data, also called a histogram.

### 2.3.1 Standard Statistics

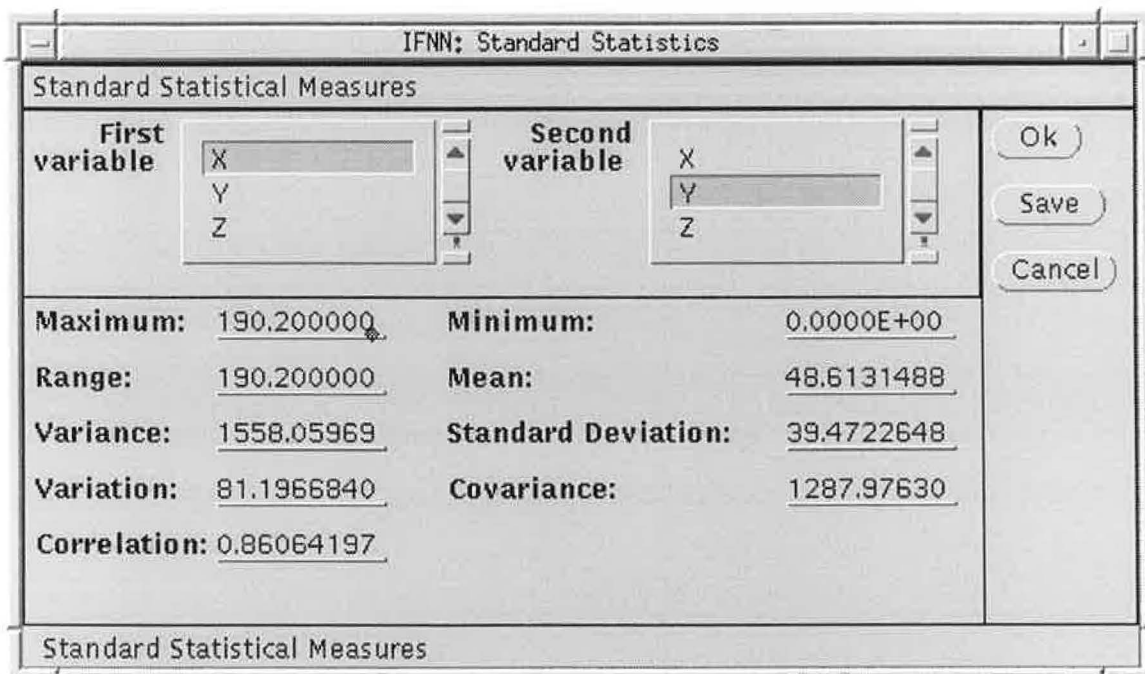


Figure 13: *Standard Statistics* window

IFNN uses the variable of the field *First variable* (see figure 13), in the sequel denoted by  $X$ , to calculate the following statistics, where  $k = 1, \dots, K$  denotes the index for rows, i.e. observations, and  $X_k$  the  $k$ -th observation of  $X$ :

- **Maximum:**

$$M(X) = \max_k(X_k).$$

- **Minimum:**

$$m(X) = \min_k(X_k).$$

- **Range:**

$$r(X) = M(X) - m(X).$$

- **Mean:**

$$\mu(X) = \frac{1}{K} \sum_k X_k.$$

- **Variance:**

$$\sigma^2(X) = \frac{1}{K-1} \sum_k (X_k - \mu(X))^2.$$

- **Standard Deviation:**

$$\sigma(X) = \sqrt{\sigma^2(X)}.$$

- **Variation:**

$$v(X) = \frac{\sigma(X)}{\mu(X)} 100.$$

The variable of the field *Second variable*, denoted by  $Y$ , is used to calculate the following statistics:

- **Covariance:**

$$\Sigma(X, Y) = \frac{1}{K-1} \sum_k (X_k - \mu(X))(Y_k - \mu(Y)).$$

- **Correlation:**

$$\rho(X, Y) = \frac{\Sigma(X, Y)}{\sigma(X)\sigma(Y)}.$$



For time series (TS) data two additional statistics are available:

- **Autocovariance:**

$$\gamma(\tau) = \frac{1}{K} \sum_{k=\max(1,1-\tau)}^{\min(K,K-\tau)} (X_{k+\tau} - \mu(X))(X_k - \mu(X)).$$

- **Autocorrelation:**

$$r(\tau) = \frac{\gamma(\tau)}{\gamma(0)}.$$

If you click on *OK*, IFNN computes the statistics. If IFNN detects a division by zero, the appropriate field is set to **NaN** for **Not a Number**.

In order to save your results, you can use the *Save* button. IFNN opens a window for the filename specification and writes the statistics to a file.

### 2.3.2 Graphical Statistics

You can use the graphical statistics procedure for a more representative analysis of the data by graphical tools. This procedure provides an interface to an external graphical program, first preparing the data for the external program and second starting the external program with the prepared data. IFNN supports GNUPLOT and ACE/gr. You can define a default for the graphical representation program in the setup module (see section 2.7).

In the fields *X* and *Y* (see figure 14) the variables for the x- and y-axis of the plot are selected. You specify the range for the x- and y-axis in the *Upper* and *Lower* fields. The default values are the current minimum and maximum values. You can choose between linear and logarithmic scaling for both axis in the field *Scale*.

If the data has only one column, like a univariate time series, IFNN provides an additional column, called *<COUNTER>*. This *<COUNTER>* column runs from 1 to the number of rows of the data.

The list *Choose a suitable symbol to represent the plot* contains different symbols used by the external program to plot the graphic. If you select *Yes* in the field *Open a new window* a new window is opened for the graphic, whereas in the case of *No* all old windows are first closed and then a new window is opened.

The fields *Title*, *X Axis* and *Y Axis* specify the labels for the title and the axis. An example of a graphical statistic produced by ACE/gr is shown in figure 15.

### 2.3.3 Frequency Chart

To estimate the empirical probability density function in terms of a histogram, you can use the frequency chart facility of IFNN. A histogram

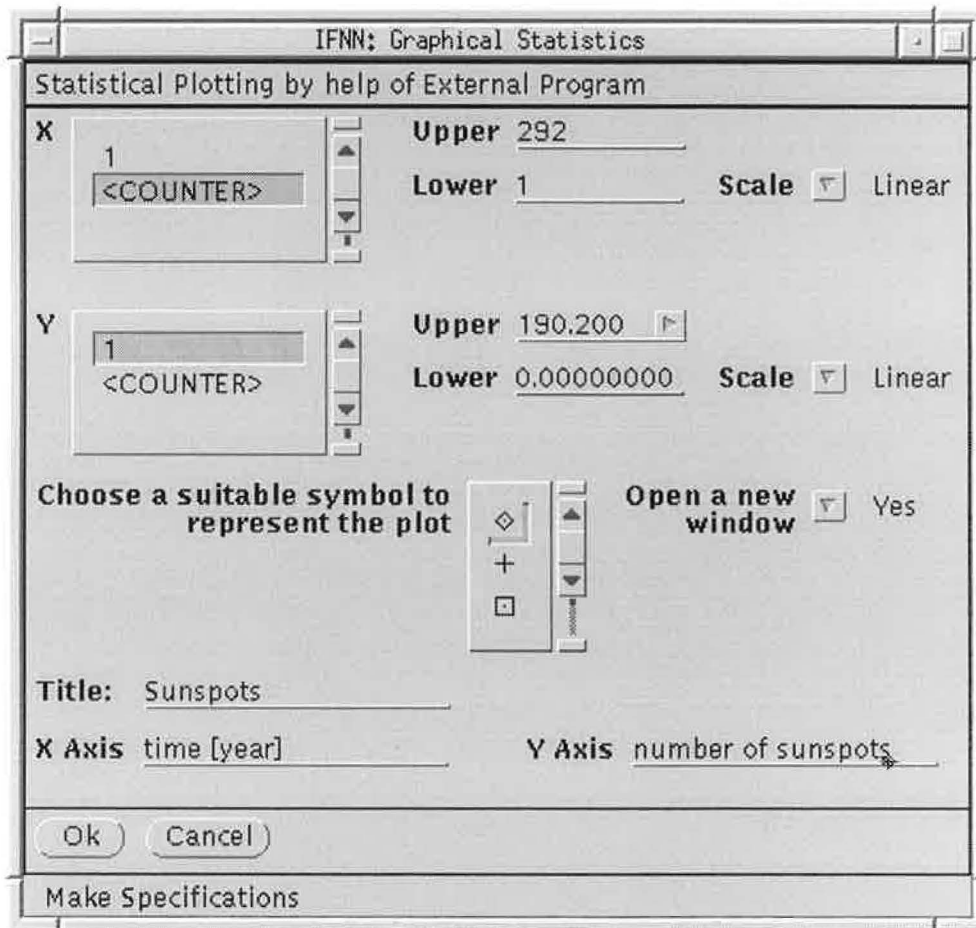


Figure 14: *Graphical Statistics* window

shows the number of observations of a variable that fall in each of a number of intervals. The specification window for the frequency charts is shown in figure 16.

The field *Specify variables* is used to select the variables for the histogram. You can select one or more variables. Use *Specify class interval* and *Number of classes* to specify the x-axis. The range of the x-axis is from zero to the length of the class interval times the number of classes. To continue click *OK* and you can choose between two different types of histogram:

- **Curve diagram** (see figure 17): Select *NO* in the field *Bar Chart Choose pattern* and a line in the field *Choose a line type*.
- **Box diagram** (see figure 18): Select the box in the field *Bar Chart Choose pattern* and *NO* in the field *Choose a line type*. IFNN shows in this case only the first variable.

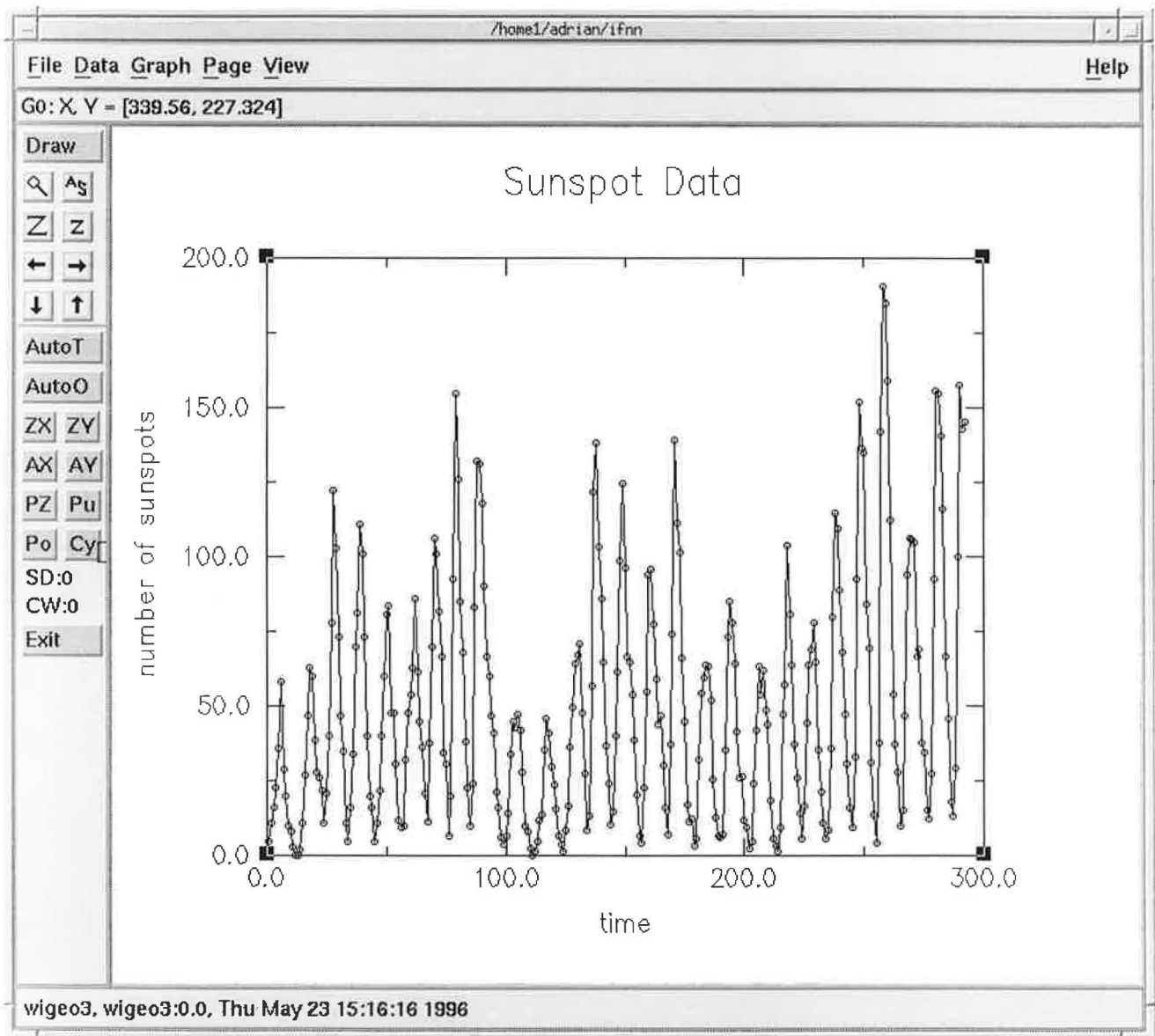


Figure 15: Example of a graphical statistic

## 2.4 Neural Network Interface Module

The neural network interface module is the main module of IFNN and provides an interface and some useful tools to work with *NeuralWorks Professional II/PLUS*, which allows the user to create, learn and test suitable neural network architectures in each domain of application. The popup menu associated with the button *NeuralNets* contains the following items:

- *NeuralWorks* ▷

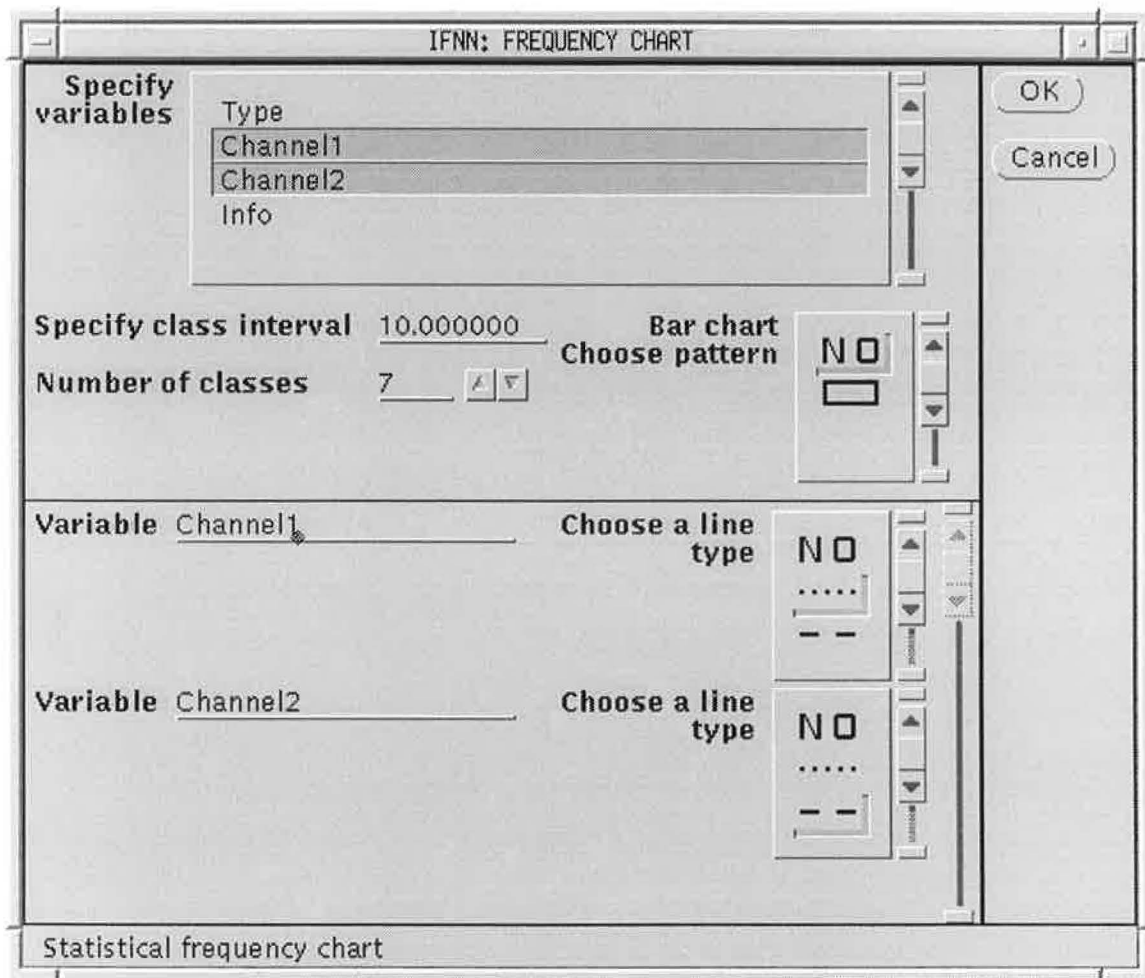


Figure 16: *FREQUENCY CHART* window

- *Utilities* ▷
  - \* *Weight Initialization*
  - \* *Show Weights*
- *Interactive Mode*
- *Batch Mode*

### 2.4.1 Utilities

The *Utilities* of the neural network interface module are the *Weight Initialization* and the *Show Weights* procedure.

The ***Weight Initialization*** procedure requires the specification of a *NeuralWorks* weight file with the extension *.nnw* and a weight initialization

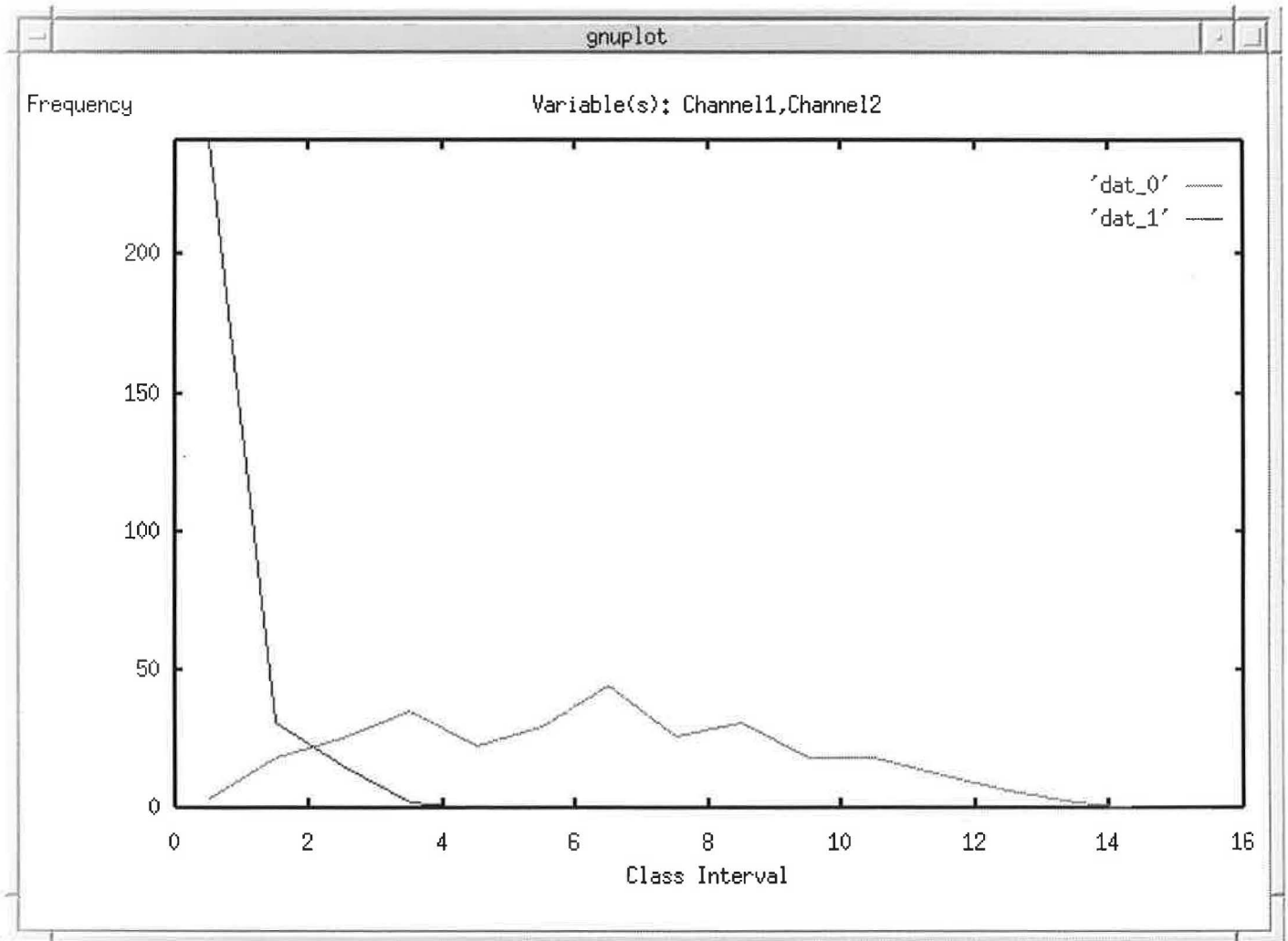


Figure 17: Example of a curve diagram

policy. This version provides the *Fan-In Dependent Uniform Distributed* weight initialization policy, which works as follows:

1. You choose a weight range  $\omega$ .
2. The individual weights are initialized by IFNN randomly by means of a random number generator from a uniform distribution over the interval  $[-\frac{\omega}{i}, +\frac{\omega}{i}]$ , where  $i$  is the number of connections coming into a unit (fan-in connections). Exceptions are the bias weights, which are set to 0.0.
3. IFNN saves the *.nnw* file with the new weights.

The *Show Weights* procedure takes as input two *NeuralWorks .nnw*

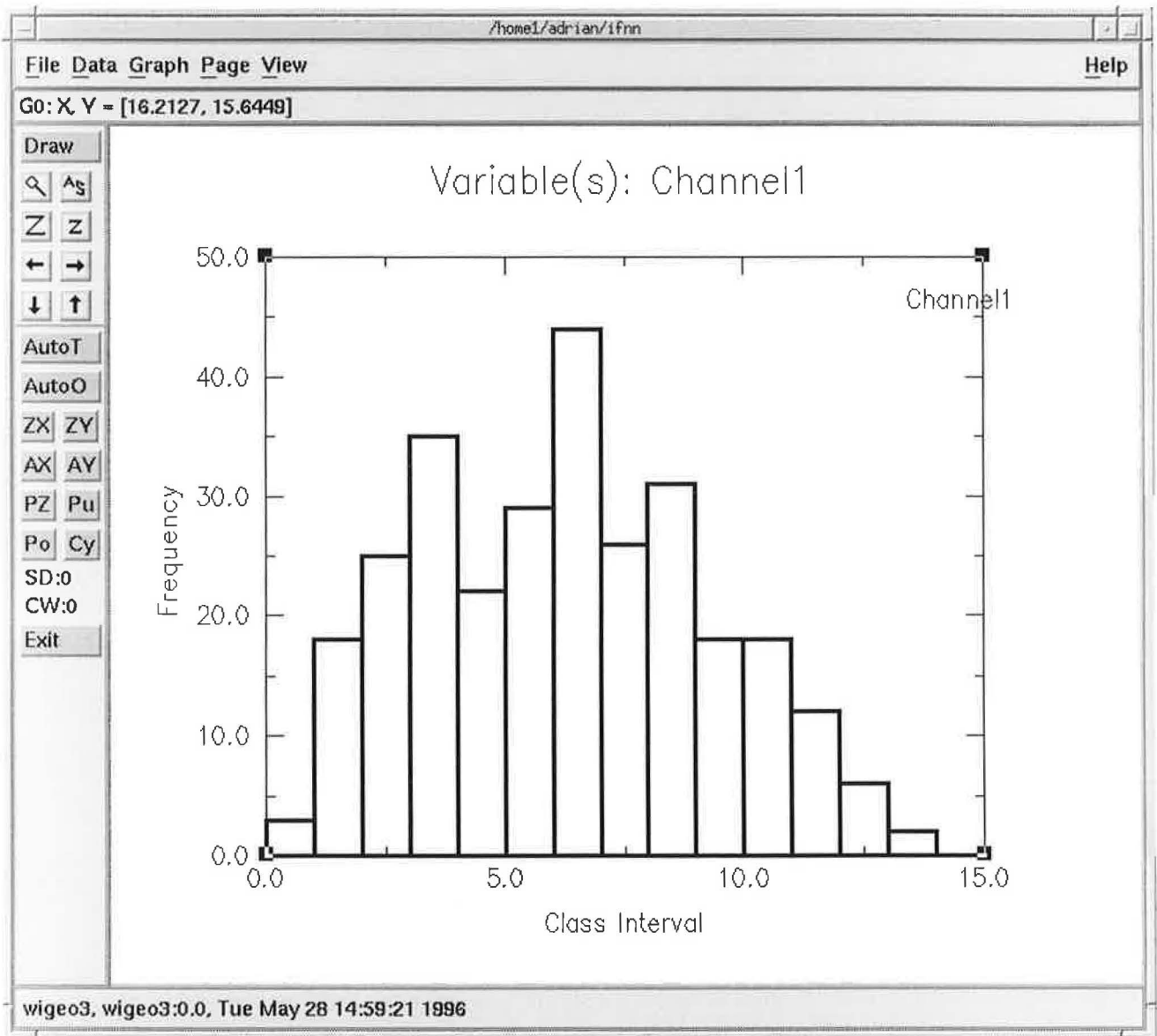


Figure 18: Example of a box diagram

weight files and produces as output a file, containing a representation of the two weight files in a tabular form. An example of such a file is the following:

```
! Start file <./init.nnw> Final file <./end.nnw>
```

```
Weights from Bias Unit
```

	Start	Final
to Hidden1 Unit1	0.0000	0.0000

```

to Hidden1 Unit2    0.0000  0.0000
to Hidden1 Unit3    0.0000  0.0000

```

```

Weights from Input Unit I1=1      Unit I1=2      Unit I1=3
                        Start  Final      Start  Final      Start  Final
to Hidden1 Unit1    0.1302  0.1302   -0.0075 -0.0075   -0.1891 -0.1891
to Hidden1 Unit2    0.1926  0.1926   -0.0946 -0.0946   -0.1878 -0.1878
to Hidden1 Unit3    0.0617  0.0617   -0.1989 -0.1989    0.0725  0.0725

```

```

Weights from Bias Unit
                        Start  Final
to Hidden2 Unit1    0.0000  0.0000
to Hidden2 Unit2    0.0000  0.0000
to Hidden2 Unit3    0.0000  0.0000

```

```

Weights from Hidden1 Unit H1=1      Unit H1=2      Unit H1=3
                        Start  Final      Start  Final      Start  Final
to Hidden2 Unit1    0.0317  0.0317    0.1118  0.1118   -0.2111 -0.2111
to Hidden2 Unit2   -0.1999 -0.1999    0.1645  0.1645    0.2176  0.2176
to Hidden2 Unit3    0.2389  0.2389    0.1935  0.1935    0.2176  0.2176

```

```

Weights from Bias Unit
                        Start  Final
to Hidden3 Unit1    0.0000  0.0000
to Hidden3 Unit2    0.0000  0.0000

```

```

Weights from Hidden2 Unit H1=1      Unit H1=2      Unit H1=3
                        Start  Final      Start  Final      Start  Final
to Hidden3 Unit1    0.2051  0.2051   -0.0858 -0.0858    0.2129  0.2129
to Hidden3 Unit2   -0.0520 -0.0520    0.2899  0.2899   -0.2001 -0.2001

```

```

Weights from Bias Unit
                        Start  Final
to Output Unit1    0.0000  0.0000

```

```

Weights from Hidden3 Unit H1=1      Unit H1=2
                        Start  Final      Start  Final
to Output Unit1    0.3998  0.3998   -0.4062 -0.4062

```

## 2.4.2 Interactive Mode

You can use this mode to automatically call *NeuralWorks* from IFNN. You have to specify the host and the directory of *NeuralWorks*, click *OK* and IFNN starts *NeuralWorks* as a new process.

### 2.4.3 Batch Mode

The *Batch Mode* provides useful tools to automate the work with *NeuralWorks*. You can prepare some scripts, i.e. jobs, through a user interface, and *NeuralWorks* will automatically run these jobs for you. The user interface is shown in figure 19, and is used to specify each of the jobs. The specification

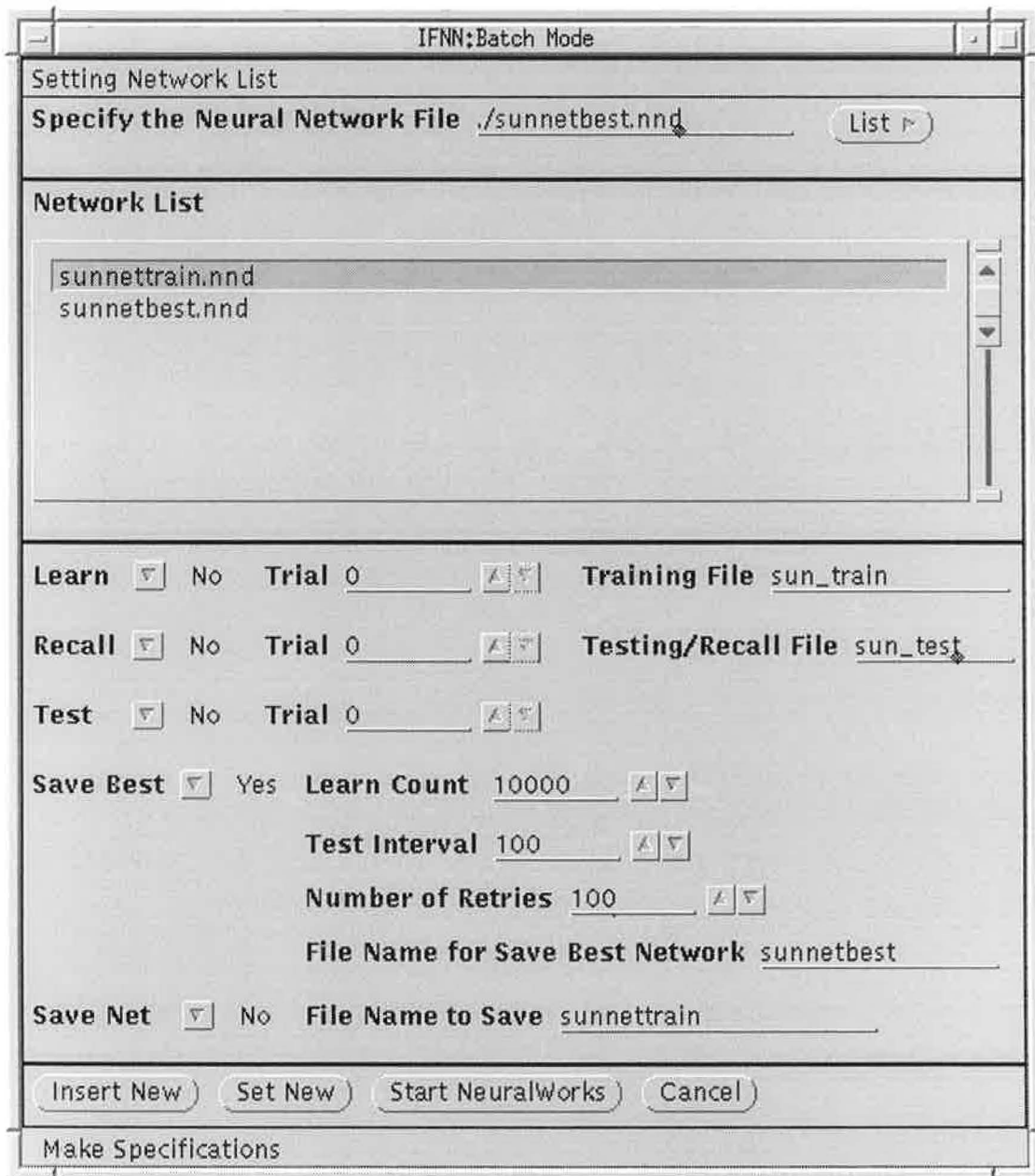


Figure 19: *Batch Mode* user interface



of a job requires the following steps:

1. Fill out *Specify the Neural Network File*. *NeuralWorks* network files have the extension *.nnd*.
2. Click on *Insert New*. IFNN inserts the network file into the *Network List*. The default values of the fields *Training File* and *Testing/Recall File* come from the *.nnd* network file. The *Trial* fields are set by IFNN according to the number of records of the training and testing/recall file. Of course, you have the flexibility to use your own specifications.
3. Specify the job itself by using the fields *Learn*, *Recall*, *Test* and so on in the middle pannel of the window. The actions taken by IFNN respectively by *NeuralWorks* are the same as using the similar commands in an interactive session with *NeuralWorks* (For a definition of the interactive and batch commands of *NeuralWorks* see NeuralWare, 1993).
4. Click on *Set New*.

You can then specify the next job in the same manner, and after the specification of all jobs, you click on *Start NeuralWorks*, and *NeuralWorks* automatically computes these jobs.

## 2.5 Analysis Module

You can evaluate the simulation results from the previous module with suitable statistical performance measures in the analysis module. Therefore you have to create instruments with *NeuralWorks*, which log the observed and the estimated values of a neural network during training (see NeuralWare, 1993). The extension for the *NeuralWorks* log files is *.nnp* and the name of the log file should include the string *dout* and *nout* for the observed (**d**esired) values log file and the estimated (**n**eural) values log file, respectively. Figure 20 shows the user interface of the analysis module.

In the field *File mask* you set the common part in the name of the *.nnp* log files. For example, **A**Log is a mask for the files *ALogdout.nnp* (desired), *ALognout1.nnp* (estimated trial 1) and *ALognout2.nnp* (estimated trial 2). If the field *File mask* is empty, all files with *dout* and *nout* substrings in the name are selected. IFNN searches for the files in the directory *Directory*. The exact UNIX search string is **mask ( "dout" | "nout" ) number ".nnp"**. To calculate some of the performance measures you have to specify an additional file in the *File* field.

After clicking on *OK*, the list of all possible plots is shown in a new window. You select then one or more plots, specify the epoch size when producing the log files, and choose between linear and logarithmic scaling of

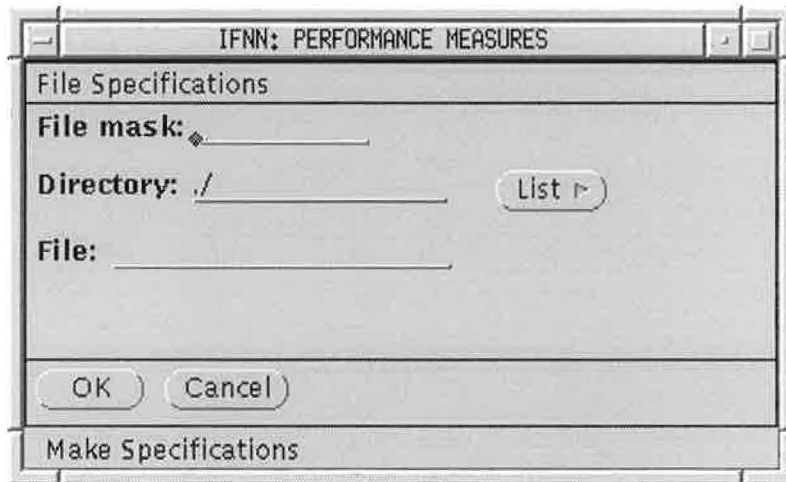


Figure 20: User interface of the analysis module

the axis. After clicking again on *OK*, IFNN opens the windows, each containing a specified plot (by the help of an external graphical representation program, which might be chosen in the setup module).

IFNN provides different types of plots: There are two types of performance measures which might be chosen. The first is ARV (Average Relative Variance), which may be calculated in two different ways, leading to the measures  $ARV(1)$  and  $ARV(2)$ :

- $ARV(1)$  is calculated as

$$ARV(1) = \frac{\sum_{Y_k \in S} (Y_k - \hat{Y}_k)^2}{\sum_{Y_k \in S} (Y_k - \bar{Y}_S)^2},$$

where  $S$  denotes the chosen epoch set of observed values  $\{Y_k\}$ ,  $\hat{Y}_k$  the estimated value of the neural network for  $Y_k$ , and  $\bar{Y}_S = \frac{1}{|S|} \sum_{Y_k \in S} Y_k$ .

- $ARV(2)$  is calculated as

$$ARV(2) = \frac{1}{\hat{\sigma}^2} \frac{1}{|S|} \sum_{Y_k \in S} (Y_k - \hat{Y}_k)^2,$$

where  $\hat{\sigma}^2$  is an estimator of the variance of the data and is calculated as the empirical variance of the first column of the file specified in the *File* field. This is usually a *NeuralWorks .nna* file. If no file specification has been made, then  $\hat{\sigma}^2$  is set to 1.

The second type of performance measure is  $R^2$ , which may be calculated again in two different ways, leading to  $R^2(1)$  and  $R^2(2)$ :

- $R^2(1)$  is calculated as

$$R^2(1) = \frac{\sum_{Y_k \in S} (\hat{Y}_k - \bar{\hat{Y}}_S)^2}{\sum_{Y_k \in S} (Y_k - \bar{Y}_S)^2},$$

where  $\bar{\hat{Y}}_S = \frac{1}{|S|} \sum_{Y_k \in S} \hat{Y}_k$ .

- $R^2(2)$  is calculated as

$$R^2(2) = 1 - ARV(1).$$

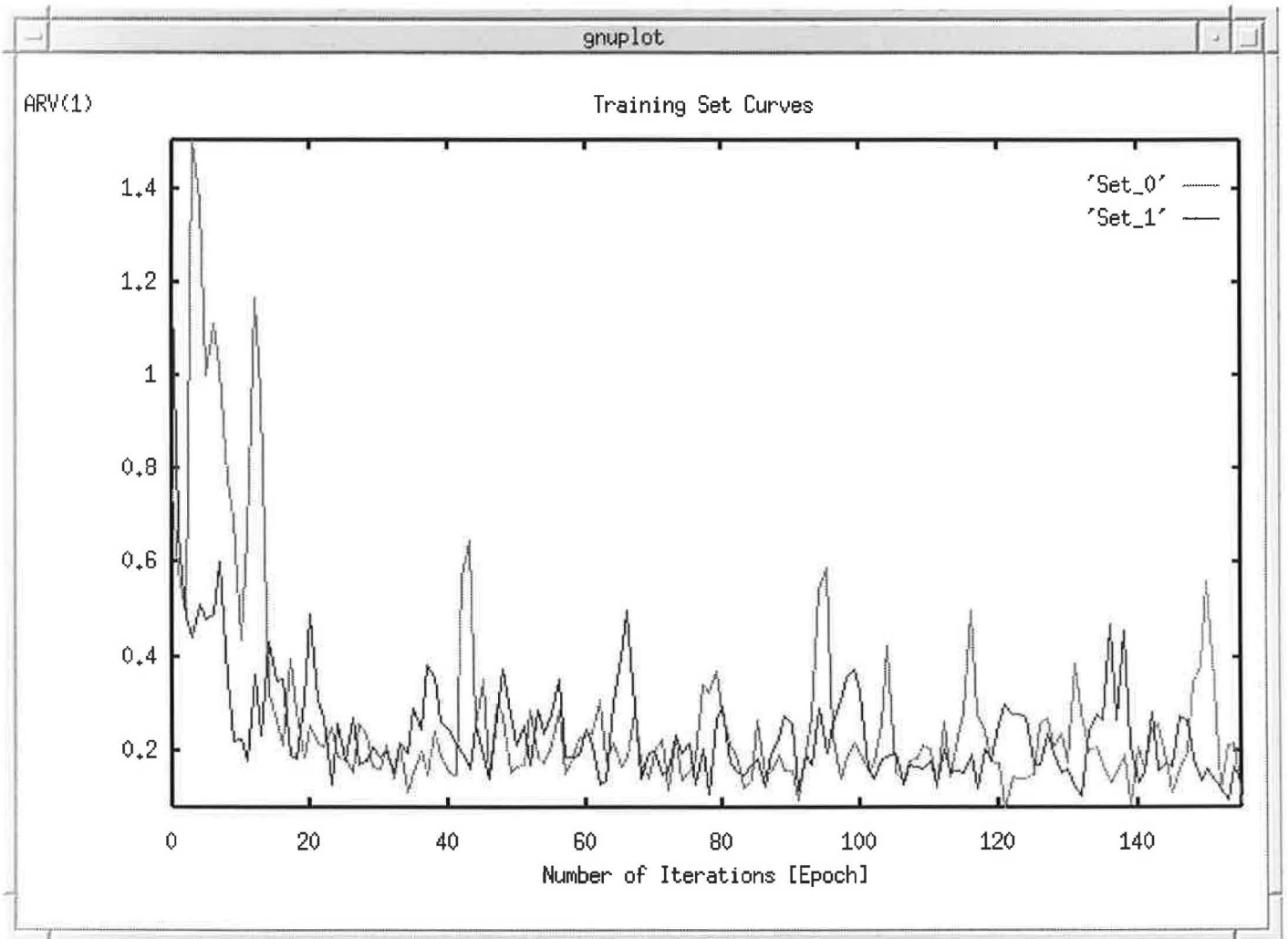


Figure 21: Example of a training set curve

**Note** that training set curves (using the *.nnp* files) are being visualized in form of plots, where the x-axis represents the number of epochs and the y-axis the performance index. An example is shown in figure 21.

One means of further investigating the predictive power (e.g. in the application domain of spatial interaction modeling) is the use of residual analysis. IFNN provides the possibility to compute absolute and relative residuals between observed values  $Y_k$  and estimated values  $\hat{Y}_k$ , and assumes that the file specified in the *File* field (usually a *NeuralWorks .nnr* file) contains in the first column the observed values  $Y_k$  and in the second column the estimated values  $\hat{Y}_k$ :

- **Absolute Residuals** are calculated as

$$R_{\text{abs}}^k = Y_k - \hat{Y}_k.$$

- **Relative Residuals** are calculated as

$$R_{\text{rel}}^k = \frac{Y_k - \hat{Y}_k}{Y_k}.$$

**Note** that the absolute (relative) residuals are graphically displayed in form of a plot (see figure 22), where the x-axis represents the observed values and the y-axis the absolute (relative) residuals.

To evaluate the performance of a neural network during training on a data set, which is independent of the training set, IFNN provides the sequel plot (For a description of the *NeuralWorks* save best mode see NeuralWare, 1993):

- **Save Best:** This plot is calculated from the file specified in the *File* field (usually a *NeuralWorks .sbl* file). The first column of the file should contain the number of training iterations and the second column a measure of the neural network performance on the independent data set during training. The x-axis of the *Save Best* plot represents the number of training iterations and the y-axis the performance measure, provided by a *NeuralWorks* instrument (not the above mentioned  $ARV(1)$ ,  $ARV(2)$ ,  $R^2(1)$  and  $R^2(2)$ ). For an example of a *Save Best* plot see figure 23.

## 2.6 Utilities

The popup menu associated with the button *Utilities* contains the following items:

- *Random Number Seed*

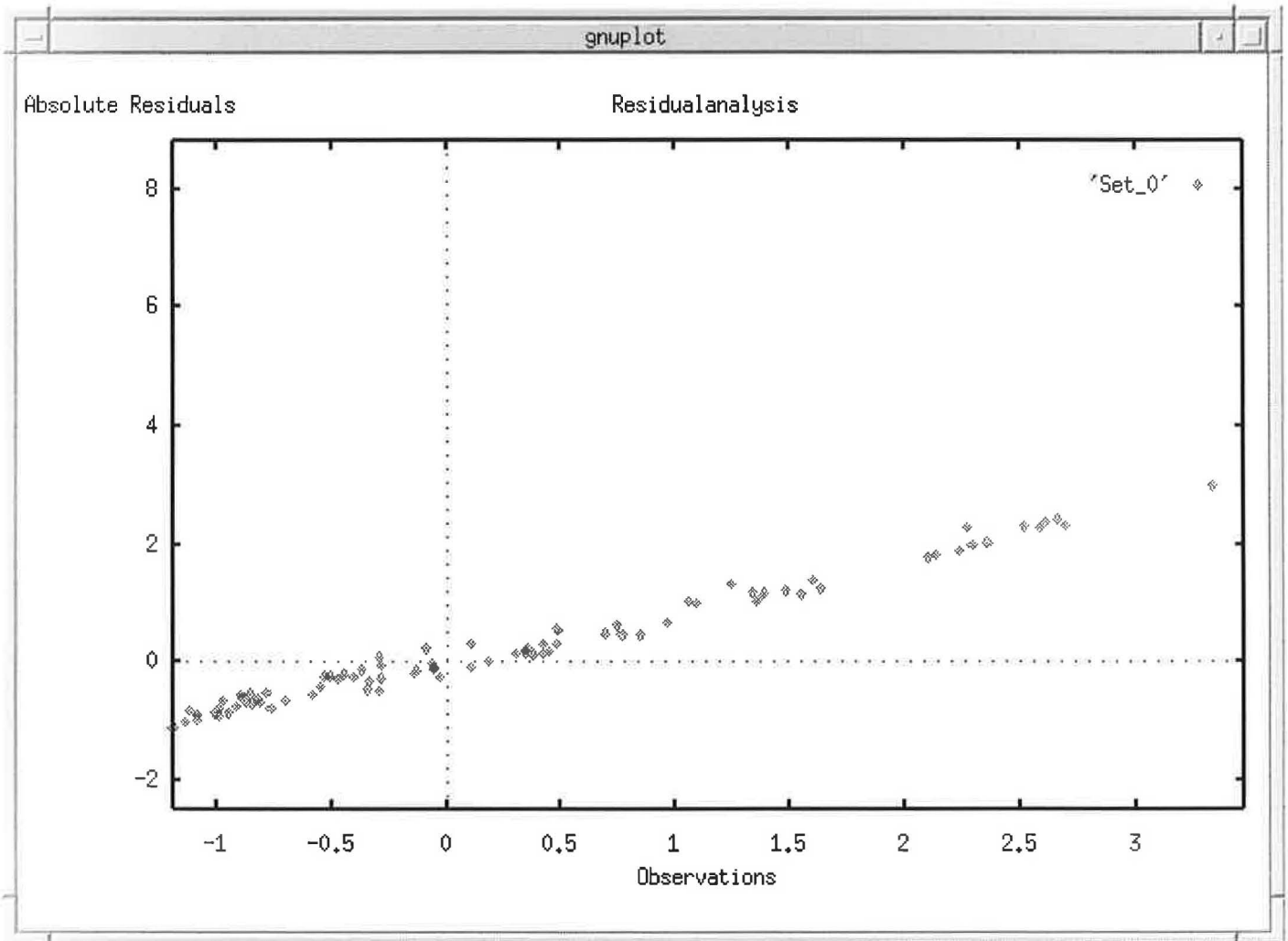


Figure 22: Example of an absolute residuals plot

The random number seed procedure is for initializing the seed values for the build-in random number generator. The random number generator uses two integer values to calculate random numbers. You have to specify these two values in the field *Random seeds*. The random number generator is used by IFNN in different procedures, e.g. in the *Weight Initialization* procedure.

## 2.7 Setup Module

This module is used to set the IFNN system parameters. You can save the whole current setting to the file  $\$HOME/.ifnn-set$  by clicking on the button *Save*. If the file already exists, the old file is renamed to *.ifnn-set .* To load a setting from *.ifnn-set* use the *Load* button and to activate your changes

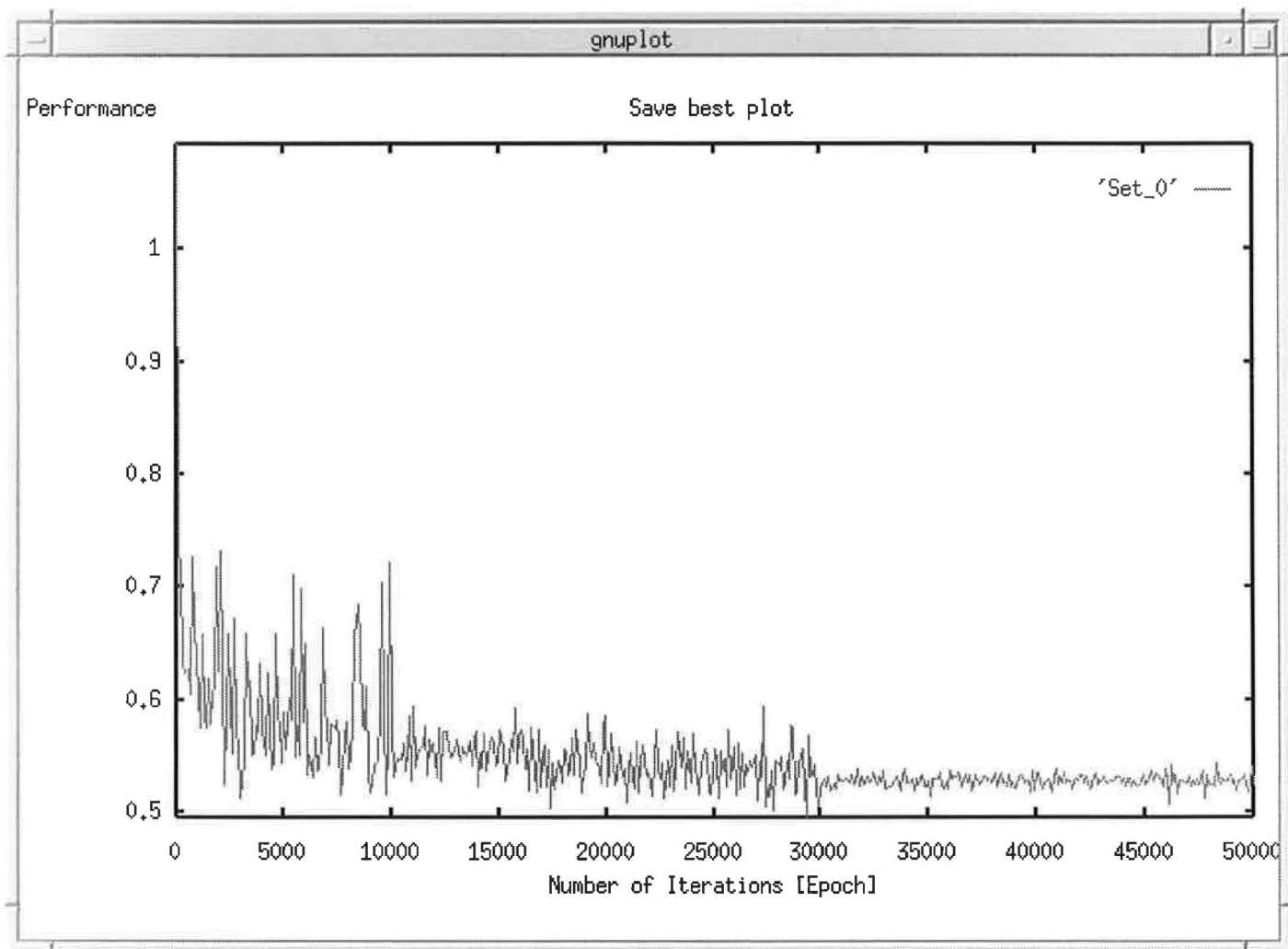


Figure 23: Example of a *Save Best* plot

use *OK*. The *Setup* window is shown in figure 24.

The first popup menu is used to set the actual external graphical representation program, where you can choose between GNUPLOT and ACE/gr.

To specify the default transformations for a specific application domain area use the *Specify default transformations* list and the popup menu *For domain*. This information is used by IFNN in the *Transformations* window (see section 2.2.2).

During the startup, IFNN automatically loads the current setting from the *.ifnn-set* file.

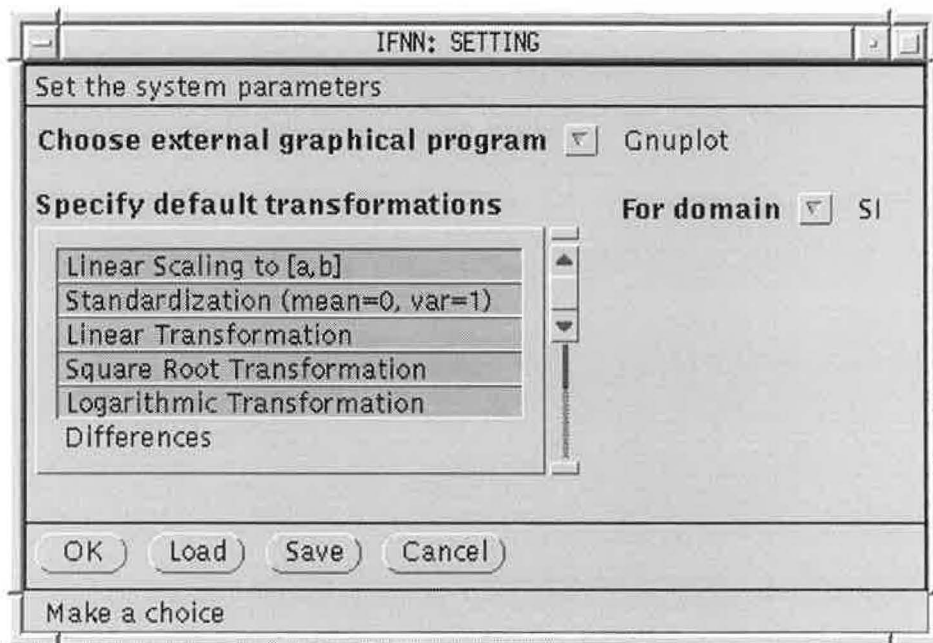


Figure 24: *Setup* window

## 2.8 Quit

To exit from IFNN, use the *Quit* button, and confirm the next question. During a working session, IFNN creates some temporary files. These files will be removed by IFNN when you exit.

## 3 System Programming Guide

### 3.1 Environment Variables

To use the integrated help facility of IFNN you have to set the shell environment variable *HELPPATH* to the directory containing the *vn\_help.info* file. For example, add the following line to your *.cshrc* file:

```
setenv HELPPATH /home1/volkov/DISTR
```

The default value of *HELPPATH* is */home1/volkov/DISTR*.

In order to interact with *NeuralWorks* you should prevent the *.cshrc* file to print something to the standard output. If the statement

```
rsh neuralworkshost ls
```

returns more than just a filelist you should add something like

```
if ( { tty -s } && $?prompt ) then
    statements
    stty cs8
endif
```

to your *.cshrc* file.

Before working with IFNN you also have to set the variable *HOME* to your home directory.

### 3.2 Content of the IFNN Distribution

The IFNN distribution contains the following files:

**C source code files:** *add\_prm.c, add\_pro.c, analys\_p.c, call\_exg.c, cl\_stat.c, edi\_chn.c, file\_lst.c, inp\_chk.c, input\_fi.c, main\_vn.c, monitor.c, nn\_call.c, nw\_b\_ifnn.c, nwib.c, random\_n.c, set\_info.c, st\_stat.c, tbl crt.c, tbl\_f\_dr.c, tr\_ts\_va.c, transfor.c, vn\_glob.c.*

**Header files:** *images\_h.h, random\_n.h, vn\_head.h, vn\_struct.h, vn\_tmp.h.*

**Other files:** *vn\_help.info, Makefile, nw\_bmake.*

**Directories:** *NW2\_SOFTWARE, doc*

### 3.3 IFNN Module Description

#### 3.3.1 Main programs

***main\_vn.c:*** Supervisor program of IFNN, which manages the main menu and controls the main modules.

***nw\_b\_ifnn.c:*** User I/O program for batch mode handling.



***input.fi.c***: Input data file handling, description of the input data structure, checking for inconsistencies and calling the external OpenLook editor.

***transfor.c***: Main program for the transformations module. Manages the internal data set.

***tr\_ts\_va.c***: Main program for the training, testing, validation module. Manages the splitting and saves the results.

***st\_stat.c***: Main program for the statistical module. Prepares the data for the external graphical program.

***analys\_p.c***: Main program for the analysis module. Reads the parameters used for the visualization of the results.

***set\_info.c***: Main program for the setup module. Reads and writes the configuration file.

***nn\_call.c***: Main module for calling *NeuralWorks*. Weight table generation. Calling *NeuralWorks* in the interactive mode. Weight initialization.

***nwib.c***: Main module for working with *NeuralWorks* in batch mode.

### 3.3.2 Service subroutines

***inp\_chk.c***: Preprocessor for the input, transformations and training, testing, validation main programs. Creates data files in the case of an embedding dimension. Divides the internal data structure into two or three sets in accordance with the chosen data partition procedure. Transforms the data with the different methods. Reads the raw data and creates the internal dynamic data structure. Is called from: *input.fi.c*, *transfor.c*, *tr\_ts\_va.c*.

***tbl\_f\_dr.c***: Creates figures for the analysis module. Searches all files. Creates the working structure for the analysis results. Calls the external graphical program. Is called from: *analys\_p.c*.

***add\_prm.c***: Reads the additional parameters for the transformations module. Manages the screen for the additional transformations parameters. Calls the transformations functions. Is called from: *transfor.c*.

***add\_pro.c***: Reads the additional parameters for the training, testing, validation module. Manages the screen for the additional parameters. Calls the splitting functions. Is called from: *tr\_ts\_va.c*.

### 3.3.3 Lower end service subroutines

*cl\_stat.c*: Calculates various statistical and transformation measures.

*file\_lst.c*: Handles the file system.

*tbl\_crt.c*: Calculates all measures for the analysis.

*monitor.c*: Calls external processes.

*call\_exg.c*: Calls external graphical programs.

*edi\_chn.c*: Calls the OpenLook text editor.

*random\_n.c*: A lagged Fibonacci sequence random number generator (public domain software).

*vn\_glob.c*: Data warehouse for the internal structure and global parameters.

### 3.3.4 Headers

*vn\_head.h*: Definition of the system widely used functions.

*vn\_struct.h*: Definition of the common structures and constants.

*images\_h.h*: Definition of the images for *st\_stat.c*.

*random\_n.h*: Header for the random number generator.

*vn\_tmp.h*: Empty header for debugging.

### 3.3.5 Temporary files

*dat\_\**: Data for the frequency statistics.

*vn\_plot.pl\**: Command file for the graphical statistics.

*Set\_\**: Data for the graphical module.

### 3.3.6 Other files

*vn\_help.info*: Help messages.

*Makefile*: Make file to build IFNN.

*nwbmake*: Make file to build *nw\_ifnn*.

### 3.3.7 Directories

**NW2\_SOFTWARE:** Contains the software for the interaction with *NeuralWorks*.

**doc:** Contains the documentation.

### 3.4 Precision Setting

The floating point numbers are saved by IFNN with a precision of `ndigits` after the decimal point. You can change `ndigits` by editing the following part of the makefile and recompiling all:

```
\# Makefile for IFNN
```

```
CC = gcc -DPRECISION= integer value for new precision
```

```
SSPKGHOME = ../..
```

`ndigits` should not be greater than 32.

### 3.5 Interaction with External Programs

The graphical user interface of IFNN needs X Windows/OpenLook and the XView library. To build graphical representations IFNN uses some external programs. IFNN supports the freeware graphical programs GNUPLOT and ACE/gr.

### 3.6 Building IFNN and System Requirements

IFNN works under UNIX and X Windows with the OpenLook tool (XView library). IFNN can be used on any computer under a UNIX clone OS (SunSPARC with SunOS, IBM PC with Linux, etc.), 5 MB disk memory and more than 8 MB RAM. During work IFNN creates some temporary files, which requires no less than 2 MB free disk memory.

Using an other window manager then OpenLook can cause some serious problems. During intensive work it may also occur, that too many files are opened, in which case we recommend to exit from IFNN and start again.

To **build IFNN**, you have to do the following steps:

1. Edit the two makefiles *Makefile* and *nwbmake* to set the appropriate macros and variables according to your system setting.
2. Build *nwb\_ifnn* by starting *nwbmake*.
3. Build *ifnn* with the `make` command.

4. Copy *nwb\_ifnn* to the *NeuralWorks* main directory.
5. Copy *ifnn* to an executable directory.

### 3.7 Implementing Future Extensions

To add a new weight initialization policy to the *Weight Initialization* procedure (see section 2.4.1) do the following steps:

In the *nn.call.c* file exists the *add\_call0* subroutine, which contains the sequel program fragment:

```

switch (choice) {
/*----- Choose Fan-in Dependent Uniform Distributed -----*/
case 0:
inactivOK = repeat;
xv_set(repeat, PANEL_INACTIVE, TRUE, NULL);
frame=(Frame)xv_create(own_frame, FRAME,
XV_X, xv_get(own_frame, XV_X)+20,
XV_Y, xv_get(own_frame, XV_Y)+20,
XV_WIDTH, 300,
XV_HEIGHT, 80,
FRAME_LABEL, (char*)own_label("Weight
Initialization Policy"),
FRAME_LEFT_FOOTER, "Make Specifications",
FRAME_SHOW_FOOTER, TRUE,
XV_SHOW, TRUE,
NULL);

xv_create(panel1, PANEL_BUTTON,
PANEL_LABEL_STRING, "Cancel",
PANEL_LAYOUT, PANEL_HORIZONTAL,
PANEL_NOTIFY_PROC, back_step0,
XV_KEY_DATA, FRAME_KEY, frame,
XV_KEY_DATA, TRANS_VALUE, repeat,
XV_HELP_DATA, "vn_help:dia_can",
NULL);
window_fit(frame);
break;
/*-----*/
case 1: /* Future contents for other user defined subroutines */

In this place you can include your own part.

break; /* More details see the documentation */

```

```

/*-----*/
    default:
} /* End of Switch */

```

The new part should implement the following features:

1. Create a new window by the OpenLook *xv\_create* command like:

```

frame=(Frame)xv_create(own_frame, FRAME,
                        XV_X,           120,
                        XV_Y,           20,
                        XV_WIDTH,       300,
                        XV_HEIGHT,      80,
                        FRAME_LABEL,    "New Weight Initialization Policy",
                        FRAME_LEFT_FOOTER, "Set parameters",
                        FRAME_SHOW_FOOTER, TRUE,
                        NULL);

```

2. Arrange the window contents needed by the new method.

3. Create an *OK* button like:

```

xv_create(panel1, PANEL_BUTTON,
          PANEL_LABEL_STRING,    "OK",
          PANEL_LAYOUT,         PANEL_HORIZONTAL,
          PANEL_NOTIFY_PROC,    new_future_subroutine,
          XV_KEY_DATA,          FRAME_KEY,      frame,
          XV_KEY_DATA,          PANEL_MAIN,    panel2_1,
          XV_KEY_DATA,          TRANS_VALUE,   repeat,
          NULL);

```

4. Implement the *new\_future\_subroutine*, which works with the neural network weights.

## Appendix

### A IFNN Messages

This chapter covers the messages, which IFNN and X Windows may issue during operation. When you see a message at the bottom side of a window (in the footer part), press the *Help* button or look at the list in this chapter.

There exist four categories of IFNN messages: **error**, **information**, **X Windows shell/XView** and **fatal error**.

In the case of error messages, you should analyse the problem at hand and then continue working, whereas information messages just serve to provide information. X Windows shell/XView messages should be treated like the error messages. Fatal error messages only appear when IFNN crashes or if not, it is recommended to stop to continue working with IFNN, analyse the problem and then continue working.

#### A.1 Error messages

Error messages may be displayed in the footer part of the IFNN windows, or may appear -marked by red color- in the *ERROR* window.

'2': *Read Columns from* or *Read Rows from* is greater than the number of columns or rows in the data file.

'3': *Read Columns to* or *Read Rows to* is greater than the number of columns or rows in the data file.

'4': File has other data types than specified.

'5': Two columns with the same number.

'6': Error in file system: open/close file, disk full, etc.

'-2': Division by zero.

'-3': Logarithm argument is less than or equal zero.

'-4': The dimension of *g* and *i* is not equal to the number of selected columns.

'File 'file name' doesn't exist'

'Sum not equal to 100%': The sum of the percentage values for the training (validation) and testing file is not equal to 100%.

'Sum not equal to the number of records': The sum of the absolute values for the training (validation) and testing file is not equal to the number of data records.

- 'No time intersection and others is not allowed'
- 'Lower values of X greater than upper values'
- 'Lower values of Y greater than upper values'
- 'X must be greater than 0 for log scale'
- 'Y must be greater than 0 for log scale'
- 'Error when calling the graphical program'
- 'Too many graphs': There exists a system resources dependent limit for the number of simultaneously opened graphs.
- 'Incorrect value'
- 'Empty table': It's not possible to save an empty table.
- 'Epoch size exceeds the number of training records'
- 'Uncorrect format of the file': The file specified in the *File* field has not the correct format.
- 'Denominator is equal to zero': A denominator of a relative residual value is equal to zero.

## A.2 Information messages

All information messages are displayed in the footer of the IFNN windows.

- 'Make a choice'
- 'Specify the input data file'
- 'Specify columns'
- 'No errors'
- 'Choose transformation procedure'
- 'Empty data set'
- 'Statistical frequency chart'
- 'Choose type of representation'
- 'Table saved to *Weights.tbl*'
- 'Not enough data points to create plot'

### A.3 Messages from the X Windows shell

These messages are displayed in the standard XView stream. Usually it is the window from which IFNN was called.

**'XView warning: Menu too large for screen (Command menu package)':**

Occurs when clicking on the *List* button, and the list is too large to fit into a window. Type the name by keyboard or remove old files from the directory.

### A.4 Fatal error messages

**'Segmentation fault':** System error due to memory protection or IFNN internal errors. Inform the system administrator.

**'IFNN system error x':** Error coming from UNIX, where x represents the UNIX error number.



## B Example of an Input Data File

The following file is an example for an input data set, containing spatial interaction (SI) data:

```
1 1 0.000000    0.0 11848000.0 11848000.0
1 2 180048.0   219.0 11848000.0   37056.0
1 3  79223.0  1009.0 11848000.0 40266000.0
1 4  26887.0  1514.0 11848000.0 16327000.0
1 5 198144.0   974.0 11848000.0 29920000.0
```

## C References

NeuralWare, Inc. (1993): **Handbooks of NeuralWorks Professional II/PLUS and NeuralWorks Explorer**. Pittsburgh, PA: NeuralWare, Inc.