

Tue Tjur

StatUnit - AN ALTERNATIVE TO STATISTICAL PACKAGES?

4 Preprint
September
1993



Institute of mathematical statistics
University of Copenhagen

ISSN 0902-8846

Tue Tjur

StatUnit - AN ALTERNATIVE TO STATISTICAL PACKAGES?

Preprint 1993 No. 4

INSTITUTE OF MATHEMATICAL STATISTICS
UNIVERSITY OF COPENHAGEN

September 1993

Tue Tjur

StatUnit — an alternative to statistical packages?

Summary. Some aspects of the author's Turbo Pascal unit StatUnit are discussed. The ability of this or a similar procedure library as an alternative to a conventional statistical package is advocated.

American Mathematical Society 1980 subject classification.
Primary 62-04; secondary 62-07, 68N15, 65U05, 62J12.

Key words and phrases.

Statistical computing, Statistics procedure library, Statistical package, Turbo Pascal, Generalized linear model.

0. Introduction.

StatUnit is a Turbo Pascal unit, i.e. a collection of procedures, functions and variables that can be added to any Turbo Pascal program. The present paper describes some aspects of this device, and defends the point of view that this — or a similar piece of software, e.g. a C library — is preferable to conventional statistical packages.

However, this should not be regarded as a general statement meaning that Turbo Pascal and StatUnit would beat SAS, GENSTAT etc. in any context. Our point of view concerns general computer languages versus statistical packages as the primary tool for a professional statistician with a solid programming experience, working with broadly varying sorts of data analysed by many different (standard and non-standard) statistical models. Moreover, we are assuming a computational environment based on IBM-compatibles under DOS. A further — perhaps obvious — reservation is that we are not trying to give good advice to statisticians working in highly specialized branches, such as chemometrics, econometrics or image analysis, where specialized programs (and, sometimes, quite different hardware environments) are required.

1. The statistical packages.

In the early days of statistical computing, most computations were performed directly by programs written in general computer languages, usually FORTRAN. The most primitive form was the kind of programs that performed, say, a two-way analysis of variance on a set of data of prescribed dimensions, read from a file in a given standard format. During the sixties and early seventies, a rapid development took place. Once a program for two-way analysis is written, it is not difficult to do the same for other design dimensions. Groups of such programs (e.g. for one-way-ANOVA, two-way-ANOVA, perhaps k -way-ANOVA, multiple regression etc.) were collected together with simple routines for input/output and data management and glued together by a more or less sophisticated control language. This is what we call a statistical package. Some of these packages (GENSTAT, for example) developed control languages that are really structured programming languages; others (e.g. BMDP, and to some extent SAS) never really escaped their origin as packages of separate programs. Lots of statistical packages, some general and some intended for more specialized applications, have been developed since the mid-sixties; not least during the last 5–10 years, where the explosion of the market for IBM-compatibles has created a large group of users for whom the concept of “machine compatibility” is merely history.

It is not the aim of this paper to give a comparative review of statistical packages. We have only mentioned a few, and these are selected because they are old and well-established, not necessarily because they are the “best” in some sense. Our point is a criticism that goes for all these

packages. This criticism has more to do with their control languages, the “glue”, than with their nature as packages of separate programs.

Our criticism is this. Why not use a powerful, well-established all-purposes programming language (like Pascal) with a fast and efficient compiler (like the Turbo Pascal compiler) as the control language, rather than a home-made shell based on a more or less inefficient interpreter?

The enormous progress in basic computer software during the last 5–10 years is felt only indirectly by the statistical package user. It has become a lot easier to create these packages. In particular, it has become a lot easier to equip these packages with shells of menus, associated editors and help screens. But a fundamental problem with the packages still remains, namely that once you are inside a program there are lots of very simple operations which are almost impossible to perform. The packages are simply not designed to give you full control of your computer, or even full access to your data. It is difficult to give specific examples, because for almost any example you can give there will be some obscure way of getting around with it in almost any package. But the problem is well-known for anyone who has been working seriously with statistical packages. Non-standard data transformations, non-standard graphics, non-standard-formatted input, non-standard statistical models, anything which is non-standard from the package’s point of view, tends to create problems that are much easier to solve in a basic computer language like FORTRAN, ALGOL, Pascal or C.

2. StatUnit.

My own experience with computational statistics comes from a combination of consulting and teaching an applied statistics course for many years. In the late seventies, we used GENSTAT in a mainframe environment. This was a great step forwards from the primitive ALGOL programs used earlier. After the entry to the personal computer age, which in this context took place around 1987–88, we turned towards SAS, mainly because we had some problems with the early DOS implementations of GENSTAT. But in the same period, Turbo Pascal came, and gradually it appeared more and more attractive to do a large part of the work in Pascal. Basic data handling, simulation, nonparametric testing, graphics and the fitting of non-standard models, are very often much easier to do in Pascal, and the programs usually run a lot faster. Thus, around 1990, I found myself in the peculiar situation of teaching a course in applied statistics with Turbo Pascal as the primary tool, SAS and GENSTAT being used only for very short standard programs, typically of the form “read data from an ASCII file in standard format and fit a standard model”. I am pretty sure that many other statisticians are in a similar situation. Under these circumstances, it is an obvious idea to write a procedure library (a unit) of counterparts to

the relatively few standard procedures that are actually borrowed from statistical packages.

Selected details about StatUnit follow in later sections. It remains to be said that the concept of a statistical procedures library is clearly not new. For example, the NAG library contains some statistical procedures in FORTRAN and ALGOL. As far as I know, such libraries have rarely been used by statisticians in mainframe environments, probably because the handling of external procedures and variables was such a tedious affair. In particular, the translation of a model formula (with factors, covariates, interactions etc.) to a design matrix, which is an important core device in any proper statistical package, requires a complex of interacting procedures and shared data structures which is not easy to manage under the somewhat restrictive rules for external procedures in the basic programming languages available for mainframe computers. These things have changed completely. The Turbo Pascal concept of a "unit" provides a flexible tool for handling of precompiled procedures, cross-referring each other and sharing data in any complicated manner and with full ability to limit the users "scope" to exactly those variables and procedures that should be visible to him.

3. A StatUnit example.

Suppose we have an ASCII file EXAMPLE.DAT of the form

```
17.07691    1
1.57173     0
...
30.49753    1
```

containing 100 lines, each line giving the value of a covariate x and a corresponding binary response y . Suppose we want to perform a standard logistic regression analysis with y as the dependent variable and $\log(x)$ as the regressor. The following Turbo Pascal program will do this.

```
1 Program Example;
2 uses StatUnit;
3 const n=100;
4 var i:integer;
5 begin
6 Start('Example','EXAMPLE.OUT');
7 DeclareVariate('X Y',n);
8 OpenInFile('EXAMPLE.DAT');
9 for i:=1 to n do
10  begin
11   ReadValue('X',i);
12   ReadValue('Y',i);
13   AssignValue('X', i, ln(Value('X',i)) );
14  end;
```

```

15 FitLogitLinear('Y=1+X');
16 ListParameters;
17 FitLogitLinear('y=1');
18 TestModelChange;
19 Finish;
20 end.

```

(The line numbers are only there for reference purposes.) After the execution of this program, the output file `EXAMPLE.OUT` (which is not shown here) contains output from line 15–18, reporting e.g. the estimates of intercept and slope in the model (line 16) and the likelihood ratio test for the hypothesis “slope=0” (line 18).

Most statisticians will probably find it easy to read and understand this program. Some comments on selected lines follow here.

The statement `uses StatUnit` in line 2 has the effect that all `StatUnit` procedures can be used in the program. At compile time, those actually used will be linked in from the precompiled unit.

The statement `Start('Example', 'EXAMPLE.OUT')` in line 6 is a technical matter, which has to do with the “activation” of `StatUnit`. Some initialization is performed, in particular a primary output file `EXAMPLE.OUT` is opened and a heading (containing the program name `Example`) is written to it.

In line 7 the statement `DeclareVariate('X Y',n)` creates two “variates” of length 100. Variates are arrays of single precision real numbers, stored in dynamic memory (the heap) and identified by `StatUnit` reference names (here `'X'` and `'Y'`).

In lines 8 to 14 data are read and the data transformation $x := \log(x)$ (line 13) is performed.

The statement `FitLogitLinear('Y=1+X')` in line 15 is an example of a model fit directive. Here, the right hand side of the equation $Y=1+X$ is a *model formula*. The symbol 1 represents a constant term and the symbol X means that X is to enter the model in a linear manner. Hence, the model becomes

$$\text{logit}(P(Y_i = 1)) = \alpha + \beta x_i.$$

The syntax for model formulas is similar to what is known e.g. from GLIM, GENSTAT and SAS.

The above example illustrates the application of `StatUnit` to a standard problem that might as well be solved by any proper statistical package. The set of “ready-for-use models” in `StatUnit` is not much different from what is standard for statistical packages. Apart from the procedure `FitLogitLinear`, the core of `StatUnit` has the procedures

`FitLinearNormal` for regression and analysis of variance and `FitLog-Linear` for multiplicative Poisson models. The advantages of `StatUnit` over the statistical packages, which have more to do with flexibility in data handling and program structure than with model selection, are not illustrated by the example.

There are, however, some differences between the class of models available in `StatUnit` and those available in standard packages. This has to do with the procedures `FitGLM` and `FitUserDefined`, which are described in the following two sections.

4. Generalized linear models.

Generalized linear models were introduced by Nelder and Wedderburn (1972) and described in further detail by McCullagh and Nelder (1983). This concept covers a large class of statistical models, where the (one-dimensional, discrete or continuous) observations are independent with distributions from the same one-parameter family. The term “linear” refers to the assumption that the individual parameters for observations are linear combinations of “covariates”, just like the means for observations in an ordinary regression or analysis of variance model. In this way, a lot of useful concepts from classical regression and analysis of variance (main effect, interaction, parallel regression lines etc.) become meaningful and manageable in a much broader class of models.

From a computational point of view, the decisive advantage of these models is that the Newton–Raphson maximization of the log likelihood function can be based on a program for linear normal models with weights. As noticed by Nelder and Wedderburn, the computations performed in each iteration are formally equivalent to the solution of a weighted regression problem. This means that a lot of source code can be reused, in particular the tedious algorithms for translation of a model formula to a (code for computation of the) design matrix, the formation and inversion of the information matrix and the treatment of overparameterizations.

Nelder and Wedderburn assumed that the one-parameter family was an exponential family of order one, with its parameter equal to some one-to-one function of the canonical parameter. However, this assumption is an unnecessary restriction which, in my opinion, makes the concept more complicated than necessary. For this reason I have redefined (and slightly extended) the concept of a generalized linear model by allowing arbitrary one-parameter families of distributions in the definition. The procedure `FitGLM` estimates such an arbitrary generalized linear model, specified by the log density or probability function $\log p(y, \theta)$ (as a function of two variables, the observation y and the parameter θ), the first and second derivatives of this function with respect to θ , and a model specification defining the linear structure and the response variate.

It must be admitted that this definition does not extend the concept of a generalized linear model much in practice. The majority of relevant examples are generalized linear models in the sense of Nelder and Wedderburn as well, and most of them can be handled by GLIM or GENSTAT. The advantage of the definition given here is that it makes the concept of a generalized linear model easier to understand, and, accordingly, the syntax more transparent.

Example. Suppose we have a data file EXAMPLE.DAT of the form

0.7571	13
4.7117	22
...	
7.9793	25
9.4457	23

consisting of, say, 200 lines. Imagine a situation where the Poisson distributed counts $y = 13, 22, \dots$ are sums of two independent, non-observable Poisson components, one with mean proportional to the covariate $x = 0.7571, 4.7117, \dots$, and one (a “background noise”) with a constant intensity. That is, we are thinking of a model of the form

$$Y_i \sim \text{Poisson}(\lambda_i) \text{ with } \lambda_i = \lambda_0 + \beta x_i.$$

This is a generalized linear model. The relevant one-parameter family is the Poisson distribution, parameterized by its mean λ (not to be confused with the standard log-linear models for counts, where the Poisson distribution should be parameterized by $\log(\lambda)$). The following program estimates this model.

```

1  program Example;
2  uses StatUnit;
3  const n=200;
4  var i:integer;
5  {$F+}
6  function logp(y,lambda:double):double;
7    begin logp:=y*ln(lambda)-lambda; end;
8  function Dlogp(y,lambda:double):double;
9    begin Dlogp:=y/lambda-1; end;
10 function D2logp(y,lambda:double):double;
11   begin D2logp:=-y/sqr(lambda); end;
12 function m(lambda:double):double;
13   begin m:=lambda; end;
14 function v(lambda:double):double;
15   begin v:=lambda; end;
16 {$F-}
17 begin
18   Start('Example','EXAMPLE.OUT');
```

```

19 DeclareVariate('X Y',n);
20 OpenInFile('EXAMPLE.DAT');
21 for i:=1 to n do
22   begin
23     ReadValue('X',i);
24     ReadValue('Y',i);
25   end;
26 InitParameter( 1, Mean('Y') );
27 FitGLM('Y=1+X',logp,Dlogp,D2logp,m,v);
28 ListParameters;
29 DeclareVariate('FITTED RES NRES',n);
30 SaveFitted('FITTED','RES','NRES');
31 List('X Y FITTED NRES');
32 Finish;
33 end.

```

The important directive here is

```
FitGLM('Y=1+X',logp,Dlogp,D2logp,m,v)
```

in line 27. The five function arguments are declared in line 6–15. `logp` is the function $\log(p(y, \lambda)) = y \log(\lambda) - \lambda$, the logarithmized probability function for the Poisson distribution (up to an additive function of y). `Dlogp` and `D2logp` are merely the first and second derivatives of this with respect to λ . The last two functions `m` and `v` return the mean and variance in the Poisson distribution as a function of the parameter. These functions are not used directly by `FitGLM`, but they are stored for possible later use by other procedures. They are actually used in line 29–31, where fitted values and normed residuals are listed in parallel with covariate values and counts. Notice the `{F+}` ... `{F-}` embracement of the five function declarations. This is a technical matter, which has to do with the Turbo Pascal compiler. Compilation in “force far calls” mode is required for procedures entering as arguments in other procedures. Apart from this, the program should be easy to read.

Notice the statement `InitParameter(1, Mean('Y'))` in line 26. The default action of `FitGLM` is to take zero as the starting value for all parameters. This would not work here because $\log(p(y, \lambda))$ is undefined for $\lambda = 0$. The statement sets the first parameter (the intercept λ_0) equal to `Mean('Y')`, i.e. the average of the responses y_i . This is obviously a good choice if β is close to zero. If this is not enough, one could do even better by an initial estimation of intercept and slope by ordinary least squares. This could be done as follows.

```

26a FitLinearNormal('Y=1+X');
26b InitParameter( 1, Parameter(1) );
26c InitParameter( 2, Parameter(2) );

```

5. User defined models.

The procedure `FitUserDefined` can, in its simplest form, be used for Newton–Raphson maximization of an arbitrary log likelihood function. All one has to do is to specify the log likelihood and its first and second derivatives. The tedious part of the job (matrix inversions, the handling of overparametrizations etc.) is performed by the procedure.

An additional feature of this procedure has to do with the frequently occurring situation where the parameters of primary interest enter in a linear way, similar to the way they enter in a generalized linear model, with a small (fixed) number of “additional” parameters determining other aspects of the model (scale parameter, shape parameters etc.). Examples are

- Cox’s partial likelihood for the proportional hazards model (no additional parameters).
- McCullagh’s model for grouped continuous data from a linear position parameter model, grouped by unknown cutpoints (cutpoints as additional parameters. See McCullagh 1980).
- log–linear models for negative binomial data (one additional parameter)
- conditional logistic regression (e.g. analysis of case–control studies by conditioning on sums of responses in case–control groups, no additional parameters)
- a large number of non–linear regression models with normal errors (with the variance and, perhaps, parameters determining the form of the non–linear dependence, as additional parameters)
- position–scale parameter problems with (non–normal) error distribution of a given type, common scale, general linear position parameter structure (one additional parameter, the scale parameter).

To settle the idea, consider the last example. Suppose we have a (fixed) probability density $p(x)$, typically one that is symmetric around zero. This is considered our “normalized” error distribution, and the kind of models we are thinking of can be stated as follows. Independent observations y_1, \dots, y_n are assumed to be distributed as

$$Y_i = \beta_1 x_{i1} + \dots + \beta_k x_{ik} + \sigma U_i$$

where the normalized “errors” U_i are independent with density p . If p is the normalized normal density, this is merely the standard multiple regression model. For fixed scale parameter σ , the model is a generalized linear model (in our sense, not in the sense of Nelder and Wedderburn), but when σ is regarded as a parameter to be estimated the model becomes more complicated. However, it is easy to express the log likelihood

and its two first derivatives in terms of the function $\log(p)$ and its two first derivatives. The decisive advantage of the procedure `FitUserDefined` in this context is that the “linear part” of the model, involving the parameters β_1, \dots, β_k , can be specified by a model formula. Thus, the creation of “dummies” for factor levels etc. can be left to `FitUserDefined`. In the procedure blocks for computation of the log likelihood and its derivatives, the elements x_{ij} of the design matrix and the (present values of the) position parameters $\beta_1 x_{i1} + \dots + \beta_k x_{ik}$ can be referenced directly, by calls to the StatUnit functions `Xmatrix` and `Fitted`. Hence, it is an easy job to write a program for fit of a model of this kind. More generally, one may write a standard procedure `FitPosScale` to be called on the form

```
FitPosScale(MODEL,logP,DlogP,D2logP)
```

which, from an ordinary model specification `MODEL` (like `'Y=1+X'`) and the three functions `logP`, `DlogP` and `D2logP` ($\log(p)$ and its two first derivatives) fits a model of this kind. This will not be explained in details here, but a small unit `PosScale`, containing a procedure like this, can be found in the StatUnit manual and on the accompanying diskette. Having performed this task once and for all, we may, for example, fit a linear regression model with logistic error distribution by a program of the following form.

```

1  program Logistic;
2  uses StatUnit,PosScale;
3  const n= ... ;
4  var i:integer;
5  {$F+}
6  function logP(y:double):double;
7    begin logP:= y-2*ln(1+exp(y)) end;
8  function DlogP(y:double):double;
9    begin DlogP:= 1-2*exp(y)/(1+exp(y)) end;
10 function D2logP(y:double):double;
11   begin D2logP:= -2*exp(y)/sqr(1+exp(y)) end;
12 {$F-}
13 begin
14   Start('Logistic','LOGISTIC.OUT');
15   DeclareVariate('X Y',n);
16   ... reading X and Y from an input file ...
17   FitPosScale('Y=1+X',logP,DlogP,D2logP);
18   ListParameters;
19   Finish;
20 end.
```

6. Program structure.

The last example illustrates an important aspect of StatUnit programs, which really applies to Pascal programs in general. It is recommendable that main bodies of programs are kept short, with most statements being performed indirectly by calls of procedures declared before the program or in separate units. Procedures in Turbo Pascal can be nested to any depth, procedures can be declared within procedures, and since programs are compiled, not just interpreted, the price in computer time for keeping programs well-organized and readable in this way is vanishing. You will soon discover that many programming tasks, which are non-standard from StatUnit's point of view, can easily be managed by creation of your own "standard" procedures. There is a considerable difference in efficiency between the powerful procedure concept in Pascal and the more heavy macro- and procedure concepts available in advanced statistical packages.

As an example (for further details, see the StatUnit manual), suppose you are frequently in a situation where you want to select the best multiple regression model in an interactive manner. StatUnit is not prepared for this, but it is a straightforward programming exercise to write a small procedure, named e.g. `SelectLinearModel`, which enables you to input model specifications from the keyboard, examine output on the screen, and select a "final" model for output to the primary output file.

You can even use StatUnit to write your own statistical packages for special purposes, if desired. An advanced example of this kind is the procedure `View` on the auxiliary unit `InterAct`, which allows you to perform a menu-controlled exploratory analysis of the data present anywhere in a program.

7. Data handling.

Computations on the unit (single-observation) level can be performed directly by Turbo Pascal's standard functions and the StatUnit functions `Value` (returning a value of a variate) and `AssignValue` (setting a value of a variate). The statement

```
AssignValue( 'X', i, ln(Value('X',i)) )
```

in line 13 of the logistic regression program in section 3 is a simple example of this. The syntax may appear somewhat clumsy and inefficient, compared to similar Pascal statements like

```
X[i]:=ln(X[i]);
```

StatUnit has facilities for more elegant ways of doing it, which will not be explained here. But again, if you prefer to have the operation of logarithmizing a variate as a standard procedure, it is easy to write a procedure `Logarithmize` (e.g. with two string arguments, one containing

the name of the variate to be logarithmized and one containing the name of the result) and put it on your own unit.

StatUnit contains a few functions (*Mean*, *Variance*, ...) for computation of one-dimensional statistics for variates. Additional functions of this kind are easily constructed by the user, as desired.

In parallel to the concept of a variate, StatUnit has the concept of a *factor*, similar to GENSTAT factors and SAS's class variables. Factors are physically stored as arrays of bytes, i.e. integers in the range 0, 1, ..., 255.

Missing values and restrictions can be handled in a way which need not be explained in details here. Restrictions are taken into account by all StatUnit procedures for which it seems relevant, including the model fit procedures.

For fast storage and retrieval, respectively, StatUnit has two procedures *SaveData* and *GetData*, to create and read files in an internal binary format (StatUnit data sets). The examples given in the present paper are slightly misleading concerning the input of data. The first program in a statistical project will typically be one that reads data from one or several more or less complicated ASCII files, creates the relevant variates and factors and stores them on a StatUnit data set. From then on, input of data is performed by a single call to *GetData*.

Other facilities to be mentioned without further explanation are

- a procedure *Sort* for parallel sorting of variates and factors (based on the QuickSort algorithm),
- procedures *List* and *List1* for (parallel and across-the-page) listing of data,
- some procedures for control of the output stream, e.g. redirection to the screen or DOS's paper basket 'nul', change of maximal line length etc.

8. Additional units.

An important advantage of a procedure library like StatUnit, in contrast to standard statistical packages, is that a sharp limit between the users scope and the scope of the "library author" is not present. In principle, any user can take STATUNIT.PAS into an editor and change the source code. I do not recommend it, and I would certainly not allow anyone to publish a modified version. But on a more moderate level, any user can, at least in principle, see what is going on by reading the source code, and add his own improvements or extensions in the form of additional units. It is not difficult to write additional units with procedures for special purposes, and the efficiency of these can come very close to what could be obtained by incorporation of similar procedures in the core of

StatUnit. Examples of such units, which, apart from their own specific purposes, may serve as models for similar user created units, follow here.

OURUNIT.PAS — a unit with many auxiliary procedures for different purposes. Among these are procedures for output of two- and three-way tables of counts, a procedure `Tabulate` for the formation of unit counts or variate sums in the groups determined by one or several classifying factors, two graphics procedures (using an associated unit `StatPlot` for graphics on a VGA screen or a HP laser printer) for scatter plots and histograms, procedures performing Bartlett's test for variance homogeneity, Wilcoxon's two sample test, Spearman's rank correlation and Pearson's χ^2 test for goodness of fit of a log-linear model. There is also a procedure for computation of "correctly normed" (t-distributed) residuals in regression analysis, useful for "exact outlier-detection".

POSSCALE.PAS — the unit for position-scale parameter models mentioned in section 5.

COXUNIT.PAS — a unit with a procedure for Cox's proportional hazards model in the simplest case (time independent covariates, no ties).

MCUNIT.PAS — McCullagh's models for discrete ordinal data, arbitrary "link function", arbitrary linear parameter structure (McCullagh 1980)

NEGBIN.PAS — Log linear models for negative binomial responses. Useful when ordinary Poisson models fail due to overdispersion.

CLOGIT.PAS — Conditional logistic regression. For matched case-control studies and conditional estimation in the Rasch model.

ANOVA.PAS — Analysis of variance models in orthogonal designs, including variance component models in the "balanced" case.

INTERACT.PAS — contains a procedure `View` for exploratory data analysis. From `View`'s menus you can draw scatter plots and histograms on the screen. Tables of counts and listings of data can be produced, examined, and optionally written to the primary output file.

The units mentioned here are documented by comments in their interface sections and distributed with StatUnit. Apart from this, many illustrative examples are included.

The StatUnit manual and diskette can be obtained from Institute of Mathematical Statistics (address below) for D.Kr. 45. Turbo Pascal 5.0 or later on an IBM-compatible with a numerical coprocessor is required. For graphics, a VGA color screen is required, for hardcopies a HP LaserJet or a plotter/printer with HP-GL.

References.

- J. A. Nelder and R. W. M. Wedderburn (1972) *Generalized Linear Models*
J.R.S.S. A **135**, pp. 370-384.
- P. McCullagh and J. A. Nelder (1983, sec. ed. 1989)
Generalized Linear Models
Chapman and Hall, London.
- P. McCullagh (1980)
Regression models for ordinal data
J.R.S.S. B **42**, pp. 109-142.
- T. Tjur (1993)
StatUnit - Turbo Pascal unit for statistical analysis
(the StatUnit manual)
Institute of Mathematical Statistics, University of Copenhagen.

Tue Tjur
Institute of Mathematical Statistics
University of Copenhagen
Universitetsparken 5
DK-2100 Copenhagen
DENMARK

PREPRINTS 1992

COPIES OF PREPRINTS ARE OBTAINABLE FROM THE AUTHOR OR FROM THE INSTITUTE OF MATHEMATICAL STATISTICS, UNIVERSITETSPARKEN 5, DK-2100 COPENHAGEN Ø, DENMARK. TELEPHONE + 45 35 32 08 99.

- No. 1 Johansen, Søren: The Role of the Constant Term in Cointegration Analysis of Nonstationary Variables.
- No. 2 Paruolo, Paolo: Asymptotic Inference on the Moving Average Impact Matrix in Cointegrated I(1) VAR Systems.
- No. 3 Johansen, Søren and Juselius, Katarina: Identification of the Long-Run and the Short-Run Structure. An Application to the ISLM Model.
- No. 4 Johansen, Søren: Identifying Restrictions of Linear Equations.

Preprints 1993

COPIES OF PREPRINTS ARE OBTAINABLE FROM THE AUTHOR
OR FROM THE INSTITUTE OF MATHEMATICAL STATISTICS,
UNIVERSITETSPARKEN 5, DK-2100 COPENHAGEN Ø, DENMARK.
TELEPHONE +45 35 32 08 99.

- | | |
|-------|---|
| No. 1 | Hansen, Henrik and Johansen, Søren: Recursive Estimation in Cointegration VAR-Models. |
| No. 2 | Stockmarr, A. and Jacobsen, M.: Gaussian Diffusions and Autoregressive Processes: Weak Convergence and Statistical Inference. |
| No. 3 | Nishio, Atsushi: Testing for a Unit Root against Local Alternatives |
| No. 4 | Tjur, Tue: StatUnit - An Alternative to Statistical Packages? |