
**Centrum voor Wiskunde en Informatica**
Centre for Mathematics and Computer Science

---

D.M. Bakker

# Gradient projection for nonparametric maximum likelihood estimation with interval censored data

Department of Operations Research, Statistics, and System Theory  Report BS-R9027  September

---

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

# Gradient Projection for Nonparametric Maximum Likelihood Estimation with Interval Censored Data

Dick M. Bakker
*Centre for Mathematics and Computer Science*
*P.O. Box 4079, 1009 AB Amsterdam, The Netherlands*

A well-known way of computing the nonparametric maximum likelihood estimator (NPMLE) of a distribution function of censored observations is the EM-algorithm, which is known as slow and hence not practicable, especially in simulation studies. In the case of interval censored observations, computing the NPMLE can be considered as a general strictly concave programming problem and a gradient projection method can be used to maximize a concave function subject to a number of constraints. This is worked out for interval censored observations with two or more censoring points. Also some simulation results are given.

*(1980) Mathematics Subject Classification (1985 Revision)*: 62G05.

*Keywords and Phrases*: Nonparametric maximum likelihood estimator, gradient projection, interval censored observations, EM-algorithm.

## 1. INTRODUCTION

Animals, machines or whatever can be observed at random times. In many situations it is only possible to see whether or not the observed object has entered a certain state and the actual entrance (or exit) times cannot be seen. This leads to so-called interval censored observations. One of the estimators of the distribution function of the transition times is the non-parametric maximum likelihood estimator (NPMLE). This situation is quite common in medical research, where only at certain times patients can be checked on whether or not their disease has entered (or left) a certain state, see e.g. Grüger (1986).

When for each observed object only one such observation is available, the NPMLE can be computed by considering the maximization problem as an isotonic regression problem, see Groeneboom (1987). This study will consider the situation with two such observations for each object. The algorithms for computing the NPMLE in this case are easy to extend to the case of three or more observations.

A formal description of the considered model runs as follows. Let X be a non-negative random variable denoting the "unobservable transition time" with d.f. $F_0(x)$ and let T and U be non-negative random variables with $\Pr(T \leq U) = 1$, the observation times with

Report BS-R9027
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

d.f. H(t,u). The censoring mechanism is such that we only know whether X is less than T, between T and U or greater than U. Let $(X_1,T_1,U_1),\ldots,(X_n,T_n,U_n)$ be a sample of random variables distributed as (X,T,U). The only observations available are $(T_i,U_i,A_i,B_i)$, with $A_i=1_{\{X_i\le T_i\}}$ and $B_i=1_{\{X_i\le U_i\}}$.

The NPMLE of the distribution function $F_0$ is obtained by maximizing the likelihood

$$(1.1) \qquad \prod_{i=1}^{n} F(t_i)^{\alpha_i} (F(u_i)-F(t_i))^{\beta_i-\alpha_i} (1-F(u_i))^{1-\beta_i}$$

as a function of F, where $(t_i,u_i,\alpha_i,\beta_i)$ are realizations of the random variables $(T_i,U_i,A_i,B_i)$.

A well-known way of computing this NPMLE is the EM-algorithm, using the iterations

$$(1.2) \qquad F^{(m+1)}(s_i) = \frac{1}{n}\sum_{j=1}^{n} \Pr^{(m)}\{X_j\le s_i \mid T_j=t_j,\ U_j=u_j,\ A_j=\alpha_j,\ B_j=\beta_j\}$$

where $s_1\le s_2\le\ldots\le s_{2n}$ are the ordered observation times $t_i$ and $u_i$. Unfortunately this algorithm converges very slowly to the solution. Grüger (1986) page 78 used this algorithm. As a convergence criterion he used that the improvement of the estimator (i.e. the maximal change of the estimator values at one iteration step) is less than 0.025. Then the process terminates after about 5 iterations, for a sample of 100 observations. This convergence criterion can lead to big errors, because the convergence of the EM-algorithm is slow and hence an improvement of multiples of 0.025 is possible. For instance in our simulation study we found in a problem with a sample of 20 observations an error w.r.t. the exact solution of 0.012, if the algorithm were stopped after 81 iterations. The last improvement was 0.00035. This does not bode well for bigger samples. Hence one gets the impression that the EM-algorithm cannot be used in this way.

Another way of computing the NPMLE is found by considering the problem as a general concave programming problem. Now a gradient projection method can be used to maximize a concave function subject to a number of linear constraints, see e.g. Rosen (1960) and Luenberger (1973). Before such a method can be used, the likelihood has to be simplified to a strictly concave function on a closed convex space. This guarantees a unique solution of the problem. The gradient projection algorithm in this paper gives the solution of the problem with 20 observations, mentioned above, after 6 iterations with a precision of 0.0002, in about one third of the CP execution time.

One of the purposes of this study is to develop a useful way of computing or estimating the NPMLE, so that it is possible to develop or check theoretical results by simulation.

## 2. THE GRADIENT PROJECTION METHOD

Maximizing (1.1) is equivalent to maximizing the log likelihood

$$(2.1) \qquad \sum_{i=1}^{n} \alpha_i \log(F(t_i)) + (\beta_i-\alpha_i)\log(F(u_i)-F(t_i)) + (1-\beta_i)\log(1-F(u_i))$$

under the restriction that F is a distribution function.

### 2.1. Some simplifications of the log likelihood

Let $0<s_1\leq s_2\leq\ldots\leq s_{2n}<1$ be the ordered observation times $t_i$ and $u_i$, and let $s_0=0$ and $s_{2n+1}=1$. Since the values of the function F between the points $s_i$ and $s_{i+1}$ are not relevant, we suppose $F(x)=F(s_i)$ if $s_i\leq x<s_{i+1}$, i=0,1,2,...,2n. Let $b_{i,1}$ and $b_{i,2}$ be defined by

$$(2.2) \qquad b_{i,1} = \begin{cases} 0 & \text{if } \alpha_i=1 \\ j & \text{if } \alpha_i=0,\ \beta_i=1 \text{ and } t_i=s_j \\ k & \text{if } \beta_i=0 \text{ and } u_i=s_k \end{cases},$$

$$(2.3) \qquad b_{i,2} = \begin{cases} j & \text{if } \alpha_i=1 \text{ and } t_i=s_j \\ k & \text{if } \alpha_i=0,\ \beta_i=1 \text{ and } u_i=s_k \\ 2n+1 & \text{if } \beta_i=0 \end{cases},$$

and let $y_j=F(s_j)$, with F(0)=0 and F(1)=1.

Maximizing (2.1) is the same as maximizing

$$(2.4) \qquad \sum_{i=1}^{n} \log(y_{b_{i,2}}-y_{b_{i,1}})$$

under the restriction that $0=y_0\leq y_1\leq\ldots\leq y_{2n}\leq y_{2n+1}=1$. In this function each variable $y_i$ can occur in at most one term. Hence there is a maximum of 2n variables. First the number of variables will be reduced.

**THEOREM 2.1.** *a.) If $b_{j,2} < b_{i,1}$ for $i\neq j$ and if for all $k\neq j$: $b_{k,2}<b_{j,2}$ or $b_{k,2}>b_{i,1}$, then the maximizing $\mathbf{y} = (y_0,y_1,\ldots,y_{2n+1})$ has to satisfy*

$$y_m = y_{b_{j,2}} \text{ for } m=b_{j,2}+1,\ldots,b_{i,1}.$$

*b.) If $b_{j,2} < b_{k,2}$ and if for all i: $b_{i,1}<b_{j,2}$ or $b_{i,1}>b_{k,2}$, then the maximizing $\mathbf{y}$ has to satisfy*

$$y_m = y_{b_{j,2}} \text{ for } m=b_{j,2}+1,\ldots,b_{k,2}.$$

**REMARK 2.2.** We look for pairs of indices $(b_{i,1},b_{j,2})$ such that $b_{i,1}$ is the biggest index less than $b_{j,2}$. Note that $(b_{i,1},b_{j,2})$ do not have to be indices of endpoints of one observation interval . This means that when we write down the corresponding signs of the $y_i$, we give the $y_i$ corresponding to a list of plus signs and the $y_i$ corresponding to the following list of minus signs the same value.

**PROOF OF THEOREM 2.1.** Consider $\log(y_{b_{i,2}}-y_{b_{i,1}})$. The maximum of this term is reached for $y_{b_{i,2}}=y_{b_{i,2}+1}$ and $y_{b_{i,1}}=y_{b_{i,1}-1}$.

a.) This means that the only indices between $b_{j,2}$ and $b_{i,1}$ are of the form $b_{k,1}$. The maximum of the terms in which these $b_{k,1}$ occur is attained by $y_{b_{k,1}} = y_{b_{k,1}-1}$. Hence $y_{b_{j,2}}=y_{b_{j,2}+1}=\ldots=y_{b_{i,1}}$.

b.) This means that the only indices between $b_{j,2}$ and $b_{k,2}$ are of the form $b_{i,2}$. The maximum of the terms in which these $b_{i,2}$ occur is attained by $y_{b_{i,2}}=y_{b_{i,2}+1}$. Hence $y_{b_{j,2}}=y_{b_{j,2}+1}=\ldots=y_{b_{k,2}}$. □

Using theorem 2.1, the problem can be reformulated as the problem of maximizing

$$(2.5) \qquad \sum_{i=0}^{k-1} \sum_{j=i+1}^{k} a_{i,j} \log(y_j'-y_i')$$

with $0=y_0'<y_1'\leq\ldots\leq y_{k-1}'<y_k'=1$ the values $y_i$ which not have been set equal and $a_{i,j}$ the number of terms of the form $\log(y_j'-y_i')$. The number of variables left is less than n, in practice it is much less.

**THEOREM 2.3.** *The function defined by (2.5) is strictly concave in the variables* $y_0',\ldots,y_k'$.

**PROOF OF THEOREM 2.3.** The logarithm is a strictly concave function, hence the function in (2.5) is concave. Suppose $\mathbf{x}=(x_0,x_1,\ldots,x_k)$ and $\mathbf{y}=(y_0,y_1,\ldots,y_k)$ are points satisfying the side conditions. Then we have

$$(2.6) \qquad \sum_{i=0}^{k-1} \sum_{j=i+1}^{k} a_{i,j}\log\{\lambda(x_j-x_i)+(1-\lambda)(y_j-y_i)\}\geq$$

$$\lambda\sum_{i=0}^{k-1} \sum_{j=i+1}^{k} a_{i,j}\log(x_j-x_i)+(1-\lambda)\sum_{i=0}^{k-1} \sum_{j=i+1}^{k} a_{i,j}\log(y_j-y_i) \text{ for } \lambda \in (0,1).$$

We shall prove that the equality sign only holds if $\mathbf{x}=\mathbf{y}$.

For each term we have $\log\{\lambda(x_j-x_i)+(1-\lambda)(y_j-y_j)\}\geq\lambda\log(x_j-x_i)+(1-\lambda)\log(y_j-y_i)$. Since $0\leq x_i<x_j\leq 1$ and $0\leq y_i<y_j\leq 1$, equality is only possible if $x_j-x_i=y_j-y_i$. Hence we have for all $a_{i,j}\neq 0$, $x_j-x_i=y_j-y_i$.

The definition of the function ensures that $a_{0,1}\neq 0$ and hence $x_1=y_1$. For all j with $a_{1,j}\neq 0$, we now have $x_j=y_j$. Next for all j and m with $x_j=y_j$ and $a_{j,m}\neq 0$ or $a_{m,j}\neq 0$ we have $x_m=y_m$. This process can be repeated till no more elements are set equal. Since the number of variables is finite, this process terminates.

Let $A_1$ be the set of indices i of elements $x_i$ which have been set equal to $y_i$, starting with $x_1$. The complement $A_1^c$ contains indices of elements $x_i$ which have no "connection" with $x_1$ and hence have not been set equal by the algorithm. Let $i_0$ be the smallest index in $A_1^c$. Each $x_m$ occurs in a term of the form $\log(x_m-x_j)$ with $a_{j,m}\neq 0$. For all j with $1\leq j<i_0$ we have $a_{j,i_0}=0$, since from $x_j\in A_1$ and $a_{j,i_0}\neq 0$ we should have $x_{i_0}\in A_1$, which is a contradiction. The only possibility is that there is a term $\log(x_{i_0})$ with $a_{0,i_0}\neq 0$. This gives $x_{i_0}=y_{i_0}$. We can now enlarge the collection $A_1$ in the same way as when we started with $x_1$ to a collection $A_2$.

When we repeat the whole process, it terminates with $x_i=y_i$ for all i. Hence the function is strictly concave. □

## 2.2. Gradient projection

In more general terms the problem is to maximize a strictly concave function f(x) on a closed convex space K of feasible points, i.e. points which satisfy the side conditions. Hence there is a unique solution. The convex space K is defined by

(2.7) $$K:=\{y\in \mathbb{R}^k:\ 0=y_0\leq y_1\leq y_2\ldots\leq y_k=1\}$$

or more general for a feasible point $y\in K$ holds:

(2.8) $\mathbf{a}_i'y\leq 0$ with $a_{i,i-1}=1$, $a_{i,i}=-1$ and $a_{i,j}=0$ for $j\neq i$, $j\neq i-1$ and $i=1,2,\ldots k$.

The latter more general form is used in Luenberger (1973) page 247 and in the algorithm, although the special structure of the constraints makes it possible to simplify some expressions in practice. A constraint $\mathbf{a}_i'y\leq 0$ is said to be active at a feasible point y if $\mathbf{a}_i'y=0$ and inactive if $\mathbf{a}_i'y<0$.

The algorithm for the gradient projection method given in Luenberger(1973) can be improved as suggested in exercise 14, page 275. The improvement is that the set of active constraints is already changed when the projected gradient vector is small instead of waiting till it is zero. The algorithm now runs as follows:

GRADIENT PROJECTION ALGORITHM

1. Start with a feasible point $\mathbf{x}$,
2. Find the subspace of active constraints M and form $\mathbf{A}_q$, defined as composed of the rows of the q active constraints,
3. Calculate the projection matrix $\mathbf{P}_q=\mathbf{I}-\mathbf{A}_q'(\mathbf{A}_q\mathbf{A}_q')^{-1}\mathbf{A}_q$ and the projection $\mathbf{d}$ of the gradient vector $\nabla f(\mathbf{x})'$ onto M, $\mathbf{d}=\mathbf{P}_q\nabla f(\mathbf{x})'$,
4. Calculate $\beta=(\mathbf{A}_q\mathbf{A}_q')^{-1}\mathbf{A}_q\nabla f(\mathbf{x})'$ and $\gamma=\max_i(-\beta_i)\vee 0$,
5. If $\mathbf{d}=\mathbf{0}$ and $\gamma=0$, stop, the solution is found,

5a. If $\|\mathbf{d}\|_\infty<\varepsilon$ and $\gamma<\varepsilon$, where $\|\mathbf{d}\|_\infty=\max_i|d_i|$, stop, we are near the solution,

6. If $\|\mathbf{d}\|_\infty<\gamma$, drop the active constraint corresponding to $\gamma$ (this is the same as deleting the corresponding row from $\mathbf{A}_q$) and return to 3,
7. If $\|\mathbf{d}\|_\infty\geq\gamma$, find $\alpha_1$ achieving $\max\{\alpha: \mathbf{x}+\alpha\mathbf{d} \text{ is feasible}\}$,
8. Calculate $r=\nabla f(\mathbf{x}+\alpha_1\mathbf{d})\mathbf{d}$, the direction gradient,
9. If $r\geq 0$, f is still increasing in $\mathbf{x}+\alpha_1\mathbf{d}$ in the direction $\mathbf{d}$, hence set $\mathbf{x}$ to $\mathbf{x}+\alpha_1\mathbf{d}$ and return to 2,
10. If $r<0$, find $\alpha_2$ achieving $\max\{f(\mathbf{x}+\alpha\mathbf{d}): 0<\alpha<\alpha_1\}$, set $\mathbf{x}$ to $\mathbf{x}+\alpha_2\mathbf{d}$ and return to 2.

**REMARK 2.4.** The special structure of the constraints (2.7) results in the projection $\mathbf{P}_q\mathbf{x}$ being just taking averages over the sets of active constraints, see paragraph 2.4. Hence we don't have to invert a big matrix.

**REMARK 2.5.** The space of feasible points has an extra condition for the terms $\log(y_j-y_i)$ with $a_{i,j}\neq 0$, since in that case we should have: $y_i\neq y_j$. Since the function value tends to $-\infty$ when $y_i$ is close to $y_j$, the algorithm cannot converge to such a point. If step 7 of the algorithm results in such a point $\alpha_1$, a point close to $\alpha_1$ can be chosen.

**REMARK 2.6.** It is quite likely that this algorithm always converges, although we have no proof for it. Du and Zhang (1986) give a convergence theorem for a gradient method that is closely related to our algorithm. The main difference is a constant factor c in step 6 and 7. They use $\|\mathbf{d}\|_2<c.\gamma$ as a criterion for changing the set of active constraints. The conditions on c would cause a very small c in our case. Especially in the initial phase (when we start with a number of constraints which is as small as possible, see paragraph 2.4) this would slow down the process of making active constraints inactive. The risk of "zig-zagging" or "jamming" (i.e. the phenomenon that a constraint is alternatingly active and inactive without settling down to one of these states) is small. If the set of active constraints does not change any more, the convergence of the algorithm is ensured. Du and Zhang (1986) conjecture that

their algorithm is convergent for any c>0 and have proved this conjecture for problems with three variables.

An explanation of the basic idea of the algorithm is given in Luenberger, we only justify the main steps.

**THEOREM 2.7.** *Let $x_0$ be a boundary point of K, which lies in the subspace M of q active constraints. Then the point $x_0$ is the global maximum of the strictly concave function f(x), subject to the constraints $a_i'x \leq 0$, if, and only if*

*(2.9)* $$P_q \nabla f(x_0)' = 0,$$

*and*

*(2.10)* $$\beta = (A_q A_q')^{-1} A_q \nabla f(x_0)' \geq 0.$$

**REMARK 2.8.** This theorem justifies step 5 of the algorithm.

**PROOF OF THEOREM 2.7.** First the sufficiency of these conditions will be shown. Assume $x_0$ is not a global maximum. Then there is a point $\mathbf{x}_1 \in K$ with $f(\mathbf{x}_1) > f(\mathbf{x}_0)$. Let $\mathbf{d} = \mathbf{x}_1 - \mathbf{x}_0$, then

$$0 < f(\mathbf{x}_1) - f(\mathbf{x}_0) < \mathbf{d}' \nabla f(\mathbf{x}_0)',$$

because for a strictly concave function f: $f(\mathbf{x}+\mathbf{h}) < f(\mathbf{x}) + \mathbf{h}' \nabla f(\mathbf{x})$.
By

(2.11) $$\nabla f(\mathbf{x}_0)' = \mathbf{P}_q \nabla f(\mathbf{x}_0)' + \mathbf{A}_q' \beta$$

and (2.9) we have

(2.12) $$\nabla f(\mathbf{x}_0)' = \mathbf{A}_q' \beta = \sum_{i=1}^{q} \beta_i \mathbf{a}_i.$$

Hence

(2.13) $$0 < \mathbf{d}' \nabla f(\mathbf{x}_0)' = \sum_{i=1}^{q} \beta_i (\mathbf{d}' \mathbf{a}_i) = \sum_{i=1}^{q} \beta_i \mathbf{a}_i' \mathbf{d}.$$

Because $\mathbf{a}_i' \mathbf{x}_0 = 0$ and $\mathbf{d}$ is a feasible direction of ascent, $\mathbf{a}_i' \mathbf{d} \leq 0$. Hence $\beta_j < 0$ for some j which contradicts (2.10). Thus that $\mathbf{x}_0$ is the global maximium.

The necessity of the conditions is a result of the Kuhn-Tucker conditions, see e.g. Luenberger (1973) page 233. □

**THEOREM 2.9.** *Let $x$ be a feasible point and suppose a component of $\beta=(A_qA_q')^{-1}A_q\nabla f(x)'$ corresponding to an active constraint is negative. If this constraint is set inactive, the projection $\hat{d}$ of the gradient vector onto the remaining subspace of active constraints is a feasible direction of ascent.*

**REMARK 2.10.** This theorem justifies step 6 of the algorithm.

**PROOF OF THEOREM 2.9.** Let $\mathbf{g}=\nabla f(\mathbf{x})'$ and $\mathbf{d}=\mathbf{P}_q\mathbf{g}=(\mathbf{I}-\mathbf{A}_q'(\mathbf{A}_q\mathbf{A}_q')^{-1}\mathbf{A}_q)\mathbf{g}$. Suppose $\beta_j<0$ and hence $\mathbf{a}_j'\mathbf{y}\leq 0$ becomes inactive. Let $\mathbf{A}_{q-1}$ denote the matrix $\mathbf{A}_q$ with the row $\mathbf{a}_j$ deleted. We have $\mathbf{A}_q\mathbf{x}=\mathbf{0}$, which leads to $\mathbf{A}_{q-1}\mathbf{x}=\mathbf{0}$ and $\mathbf{a}_j'\mathbf{x}=0$. Further $\mathbf{P}_{q-1}\mathbf{P}_q\mathbf{g}=\mathbf{P}_q\mathbf{g}$, because $\mathbf{P}_q$ is a projection onto a part of the subspace corresponding to the projection $\mathbf{P}_{q-1}$. We have

$$\mathbf{g}=\mathbf{P}_q\mathbf{g}+\mathbf{A}_q'\beta \tag{2.14}$$

and

$$(\mathbf{P}_{q-1}\mathbf{A}_q')_i=\begin{cases}\mathbf{0} & \text{if } i\neq j\\ \mathbf{P}_{q-1}\mathbf{a}_j & \text{if } i=j\end{cases}. \tag{2.15}$$

Hence

$$\mathbf{P}_{q-1}\mathbf{g}=\mathbf{P}_q\mathbf{g}+\mathbf{P}_{q-1}\mathbf{a}_j\beta_j. \tag{2.16}$$

For the new direction $\hat{\mathbf{d}}=\mathbf{P}_{q-1}\mathbf{g}$ we have

$$\mathbf{a}_j'\hat{\mathbf{d}}=\mathbf{a}_j'\mathbf{P}_{q-1}\mathbf{g}=\mathbf{a}_j'\mathbf{P}_q\mathbf{g}+\mathbf{a}_j'\mathbf{P}_{q-1}\mathbf{a}_j\beta_j=\mathbf{a}_j'\mathbf{P}_{q-1}'\mathbf{P}_{q-1}\mathbf{a}_j\beta_j<0, \tag{2.17}$$

since $\mathbf{a}_j'\mathbf{P}_q=0$. This makes $\hat{\mathbf{d}}$ a feasible direction. Since $\mathbf{g}-\hat{\mathbf{d}}$ is orthogonal to $\hat{\mathbf{d}}$,

$$\mathbf{g}'\hat{\mathbf{d}}=(\mathbf{g}-\hat{\mathbf{d}}+\hat{\mathbf{d}})'\hat{\mathbf{d}}=\|\hat{\mathbf{d}}\|_2^2>0. \tag{2.18}$$

Hence it is a direction of ascent. □

## 2.3. Parallel tangents

When the set of active constraints does not change, another improvement can be made. In theory the path of ascent is zig-zag in character, because a new search direction is orthogonal to the previous one. This phenomenon suggests an accelerated gradient method

which searches for a maximum on the line connecting a point $\mathbf{x}_0$ and a point $\mathbf{y}$ two gradient steps later, see e.g. Luenberger (1973) page 184.

Let k be the total number of variables, q the number of active constraints and p=k–q the number of inactive constraints. Then there are p unequal variables. To solve a problem of p variables, the algorithm for parallel tangents (PARTAN) is defined as:

**PARTAN**

1. Start with an arbitrary feasible point $\mathbf{x}_0$,
2. Set i to 1,
3. Find the point $\mathbf{x}_1$ by a gradient projection step from $\mathbf{x}_0$. If the set of active constraints is changed in $\mathbf{x}_1$, set $\mathbf{x}_0$ to $\mathbf{x}_1$ and return to 2,
4. Set i to i+1 (start of the partan part),
5. Find the point $\mathbf{y}$ by a gradient projection step from $\mathbf{x}_1$. If the set of active constraints is changed in $\mathbf{y}$, set $\mathbf{x}_0$ to $\mathbf{y}$ and return to 2,
6. Calculate $\mathbf{d}=\mathbf{y}-\mathbf{x}_0$ and find $\alpha_1$ achieving $\max\{\alpha: \mathbf{x}_0+\alpha\mathbf{d} \text{ is feasible}\}$,
7. Calculate $r=\nabla f(\mathbf{x}_0+\alpha_1\mathbf{d})\mathbf{d}$, the direction gradient,
8. If $r\geq 0$, f is still increasing in $\mathbf{x}_0+\alpha_1\mathbf{d}$ in the direction $\mathbf{d}$, hence set $\mathbf{x}_0$ to $\mathbf{x}_0+\alpha_1\mathbf{d}$ and return to 2,
9. If $r<0$, find $\alpha_2$ achieving $\max\{f(\mathbf{x}_0+\alpha\mathbf{d}): 0<\alpha<\alpha_1\}$,
10. If i is greater or equal than p, set $\mathbf{x}_0$ to $\mathbf{x}_0+\alpha_2\mathbf{d}$ and return to 2,
11. Set $\mathbf{x}_0$ to $\mathbf{x}_1$, $\mathbf{x}_1$ to $\mathbf{x}_0+\alpha_2\mathbf{d}$ and return to 4.

Each step of the process consists of a gradient projection step and an additional move that provides further increase of the objective function, hence the convergence of the PARTAN algorithm is at least as good as the convergence of gradient projection. Restarting the process after p times the partan part (step 10) accelerates the convergence of the algorithm, see Luenberger (1973).

## 2.4. Application of the gradient projection with PARTAN

The special structure of the constraints (2.7) makes it easy to calculate the projection $\mathbf{P}_q\mathbf{x}$ and $\beta$. The constraints can be written in the form

$$
\begin{aligned}
&1.\quad -y_1 \leq 0\\
&2.\quad y_1-y_2 \leq 0\\
&3.\quad y_2-y_3 \leq 0\\
&\qquad\vdots\\
&k.\quad y_{k-1}-y_k \leq 0
\end{aligned}
\tag{2.19}
$$

The construction of function (2.5) gives $y_1>0$ and $y_{k-1}<y_k=1$. Let $l_1<l_2<\ldots<l_p$ be the indices of the inactive constraints, then we have $l_1=1$ and $l_p=k$. For the projection $\mathbf{d}=\mathbf{P}_q\mathbf{x}=(d_1,d_2,\ldots,d_k)$ of a point $\mathbf{x}=(x_1,x_2,\ldots,x_k)$ we have $d_{l_i}=d_{l_i+1}=\ldots=d_{l_{i+1}-1}$ for $i=1,2,\ldots,p-1$. The variables $d_{l_i},d_{l_i+1},\ldots,d_{l_{i+1}-1}$ only depend on $x_{l_i},x_{l_i+1},\ldots,x_{l_{i+1}-1}$, hence the blocks can be considered separately for the projection. We now have

$$d_{l_i}=d_{l_i+1}=\ldots=d_{l_{i+1}-1}=\frac{x_{l_i}+x_{l_i+1}+\ldots+x_{l_{i+1}-1}}{l_{i+1}-l_i} \tag{2.20}$$

The projection $\mathbf{P}_q$ is the same as taking averages over a set of active constraints.

For $\beta=(\mathbf{A}_q\mathbf{A}_q')^{-1}\mathbf{A}_q\nabla f(\mathbf{x})'$ we have

$$\mathbf{A}_q'\beta=\nabla f(\mathbf{x})'-\mathbf{P}_q\nabla f(\mathbf{x})'. \tag{2.21}$$

Let $\beta_i$ belong to the i-th constraint, then

$$\beta=(\beta_2,\ldots,\beta_{l_2-1},\beta_{l_2+1},\ldots,\beta_{l_i-1},\beta_{l_i+1},\ldots,\beta_{l_p-1})' \tag{2.22}$$

$\mathbf{A}_q'\beta$ can be considered as a vector composed of components of the form

$$(\beta_{l_i+1},-\beta_{l_i+1}+\beta_{l_i+2},\ldots,-\beta_{l_{i+1}-2}+\beta_{l_{i+1}-1},-\beta_{l_{i+1}-1})', \text{ for } i=1,2,\ldots,p-1. \tag{2.23}$$

Since

$$\sum_{j=l_i}^{l_{i+1}-1}(\nabla f(\mathbf{x})_j-(\mathbf{P}_q\nabla f(\mathbf{x})')_j)=0, \text{ for } i=1,2,\ldots p-1, \tag{2.24}$$

we have

$$\beta_i=\sum_{j=1}^{i}(\nabla f(\mathbf{x})_j-(\mathbf{P}_q\nabla f(\mathbf{x})')_j), \text{ for } i=2,3,\ldots,k, \tag{2.25}$$

with $\beta_{l_j-1}=0$ for $j=2,3,\ldots,p$. Now we have

$$\gamma=\max_i(-\beta_i)=\max_i\sum_{j=1}^{i}((\mathbf{P}_q\nabla f(\mathbf{x})')_j-\nabla f(\mathbf{x})_j). \tag{2.26}$$

**EXAMPLE**: Suppose we have 8 variables and $l_1=1$, $l_2=5$ and $l_3=8$. Note that $x_8=1$. Then

$$\mathbf{A}_q=\begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \end{pmatrix}$$

and

$$A_q'\beta = A_q' \begin{pmatrix} \beta_2 \\ \beta_3 \\ \beta_4 \\ \beta_6 \\ \beta_7 \end{pmatrix} = \begin{pmatrix} \beta_2 \\ -\beta_2+\beta_3 \\ -\beta_3+\beta_4 \\ -\beta_4 \\ \beta_6 \\ -\beta_6+\beta_7 \\ -\beta_7 \end{pmatrix}$$

Hence

$$\beta_2 = g_1 - d_1,$$
$$\beta_3 = g_1 - d_1 + g_2 - d_2,$$
$$\beta_4 = g_1 - d_1 + g_2 - d_2 + g_3 - d_3 = -(g_4 - d_4),$$
$$\beta_6 = g_5 - d_5,$$
$$\beta_7 = g_5 - d_5 + g_6 - d_6 = -(g_7 - d_7),$$

with **g** defined as in the proof of theorem 2.9.

The algorithm starts with a feasible point **x**. There are several possibilities to select such a point. The simplest way is to start with only inactive constraints and $x_i = \frac{i}{k}$ for i=1,2,...,k. From the situation with one observation point we know that for the solution most constraints are active. Hence the initial phase, in which a lot of inactive constraints are made active, will take a lot of time.

One can also start with a number of inactive constraints which is as small as possible. An algorithm to select such a point is given below.

STARTING POINT

1. Start with $l_1=1$ and m=1, the first constraint is always active,
2. Set m to m+1, ignore all terms $\log(y_j - y_i)$ with $i<l_{m-1}$ and determine the smallest j with $a_{i,j}\neq 0$, set $l_m$ to j,
3. If $l_m$ is less than k return to 2,
4. Set $x_j$ to $\frac{i}{m}$ for $j=l_i,...l_{i+1}-1$, for i=1,...,m−1, and $x_k$ to 1.

Unfortunately we have no guarantee that the inactive constraints found this way are also inactive for the solution of the maximizing problem. In our simulation study this need not be the case. Nevertheless this starting point gives a faster rate of convergence than the previous one. For other starting points we need more insight into the behaviour of the NPMLE.

## 3. SIMULATION RESULTS

The algorithms mentioned in this paper are used in the program NPMLE for simulations, see the appendix. This program is written in Fortran77 (Balfour and Marwick (1979)).

To get an impression for the precision of the NPMLE in seven special cases, we give some results for a sample of 1000 observations (T,U,A,B). The pictures are made with DISSPLA, a graphic library. The numbers correspond with the class numbers in the program NPMLE.

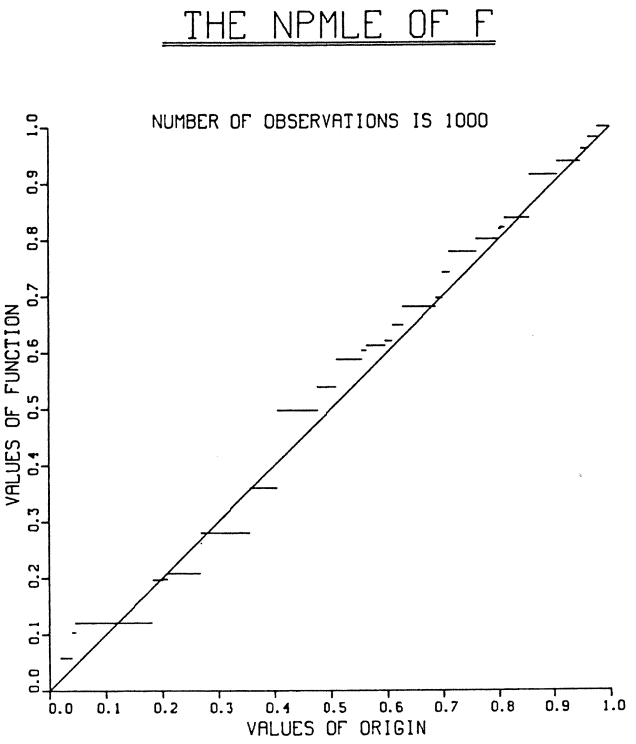


FIGURE 3.1. 1. $F(x) = x$, $h(t,u) = \frac{1}{1-t}$.

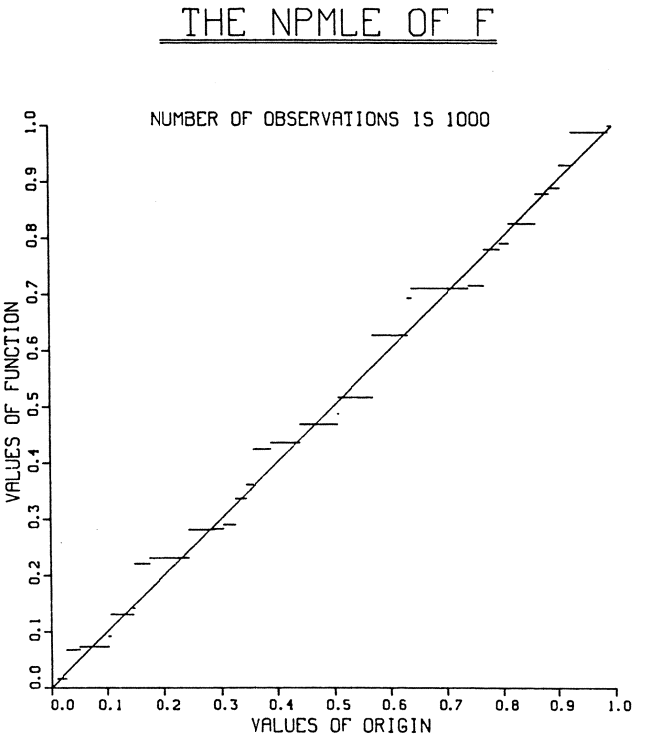


FIGURE 3.2. 2. $F(x)=x$, $h(t,u) = \frac{1}{2\sqrt{t}(1-t)}$

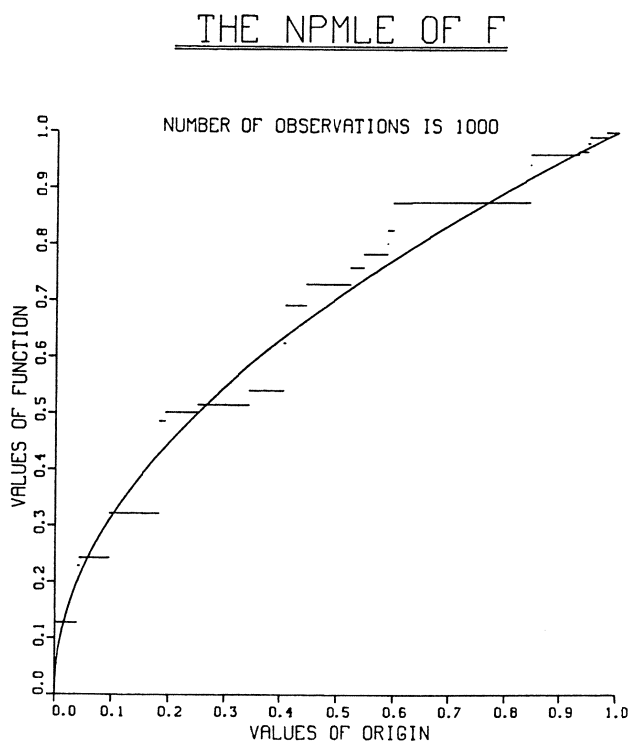


FIGURE 3.3. 3. $F(x)=\sqrt{x}$, $h(t,u) = \frac{1}{1-t}$.

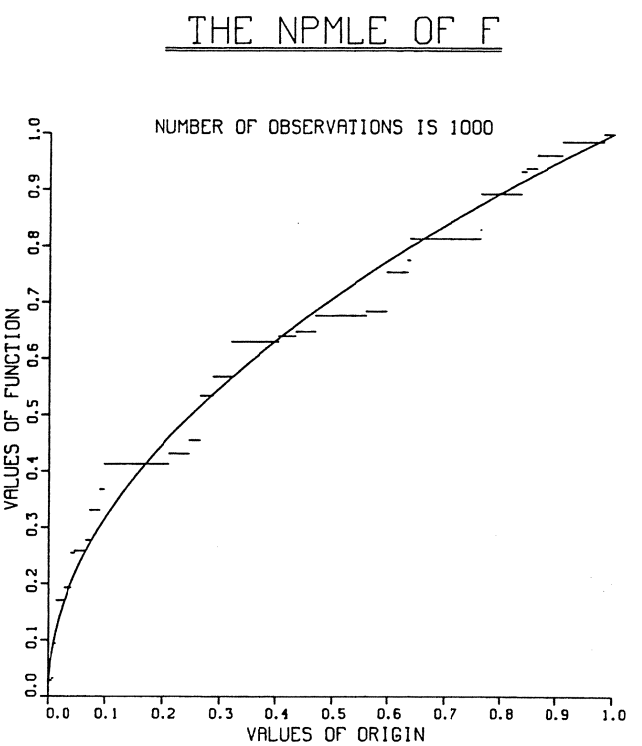


FIGURE 3.4. 4. $F(x)=\sqrt{x}$, $h(t,u) = \frac{1}{2\sqrt{t}(1-t)}$.

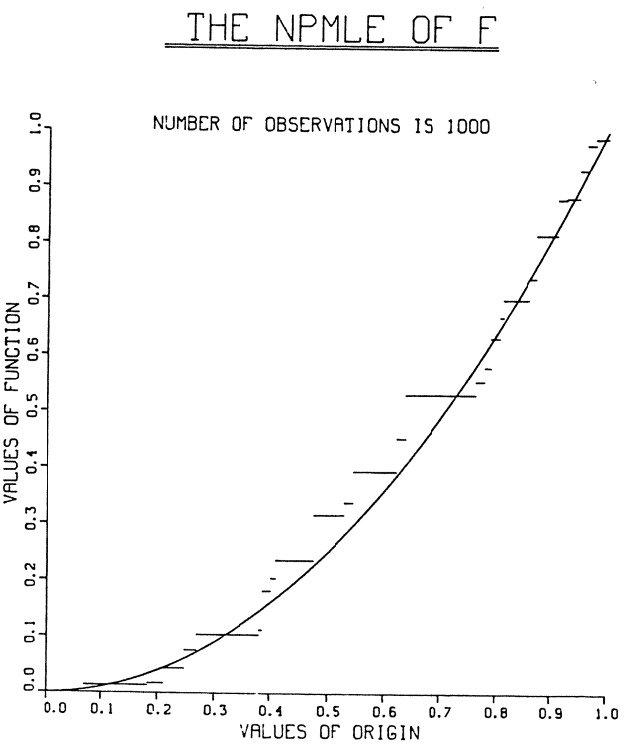


FIGURE 3.5. 5. $F(x)=x^2$, $h(t,u) = \frac{1}{1-t}$.

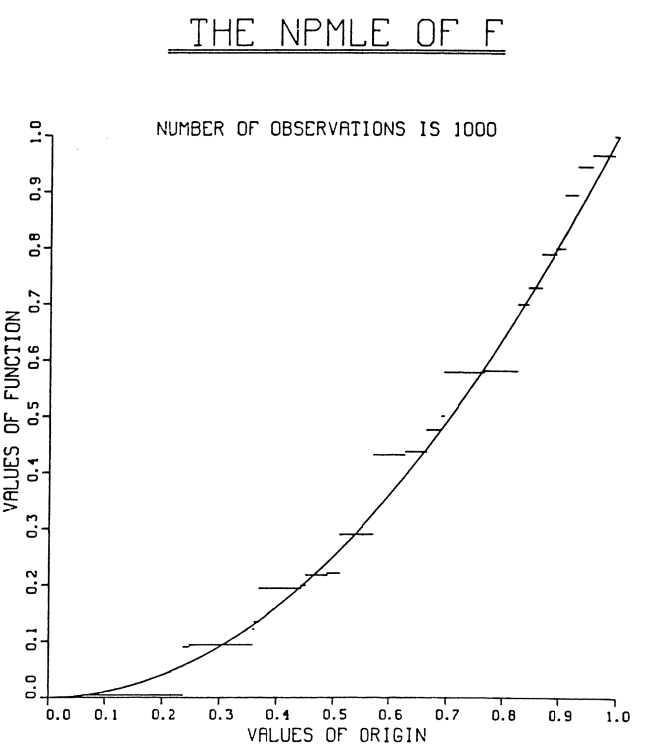


FIGURE 3.6. 6. $F(x)=x^2$, $h(t,u) = \frac{1}{2\sqrt{t}(1-t)}$.

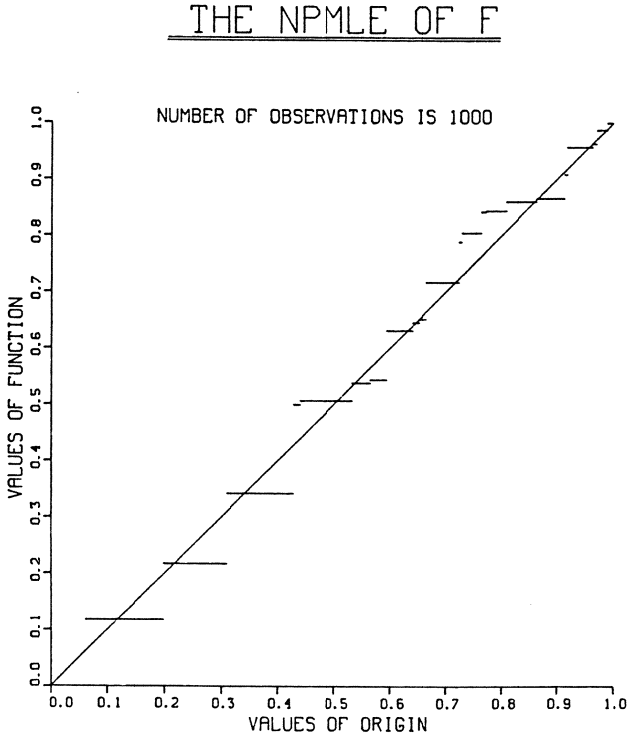


FIGURE 3.7. 7. $F(x)=x$, $h(t,u) = \frac{2t}{1-t}$.

The gradient projection method terminates if the projected gradient is small enough. We have used $||\mathbf{d}||_\infty < 10^{-4}$ as a criterion. The number of references to ZBREN (the IMSL subroutine we used to determine a zero of a function) before the process terminates varies strongly. We took a maximum of 250 and in most cases this is enough. For three different combinations of $F_0$ and h, corresponding with figure 3.1, 3.5 and 3.7, we have done 100 simulations with 1000 observations. The minimum number of references to ZBREN was 39 in class 1, 40 in class 5 and 41 in class 7. In all three cases there were not more than three simulations which did not terminate before 250 references. The simulations were done at a cyber 995E, and the average CP time to determine a solution was 6.6 seconds.

**Acknowledgement**

This research was done for a Master's thesis at the department of mathematics of the University of Amsterdam. I want to thank professor Piet Groeneboom for his help and patience. Without his knowledge it would not have been possible for me to do this study.

## REFERENCES

BALFOUR, A. AND MARWICK, D.H. (1979).
*Programming in Standard Fortran 77*, Heinemann Educational Books, London.

DU, D.-Z. AND ZHANG, X.-S. (1986).
A convergence theorem of Rosen's gradient projection method, *Mathematical Programming 36*, 135-144, North-Holland.

GROENEBOOM, P. (1987).
*Asymptotics for incomplete censored observations*, Report 87-18, Department of Mathematics, University of Amsterdam.

GRÜGER, J. (1986).
*Nichtparametrische Analyse sporadischer beobachtbarer Krankheitsverlaufdaten*, Dissertation Dortmund University.

LUENBERGER, D.G. (1973).
*Introduction to linear and nonlinear programming*, Addison-Wesley, London.

ROSEN, J.B. (1960).
The gradient projection method for nonlinear programming, Part 1. Linear constraints, *Journal of the society for industrial and applied mathematics, vol.8*, march 1960, USA, 181-217.

```
      PROGRAM NPMLE

* *********************************************
* GRADIENT PROJECTION FOR NONPARAMETRIC MAXIMUM
* LIKELIHOOD ESTIMATION OF INTERVAL CENSORED OBSERVATIONS
* *********************************************
* ************** DICK M. BAKKER ***************
* *********************************************
* ****  MATHLIB LIBRARY SUBROUTINES ARE USED ***
* *********************************************
* ****** READ FROM TAPE1
* ****** WITH INFORMATION:
* ******      ITMAX    (MAXIMUM NUMBER OF REFERENCES TO ZBREN)
* ******      NTSIMUL   N   ICLASS  (POSSIBLY REPEATED)
* ******      (NUMBER OF SIMULATIONS,OBSERVATIONS AND CLASS)
* ****** WRITE ON TAPE2
* ****** RESULTS ON TAPE3 FOR FUTHER ANALYSIS
* ****** INFORMATION FROM MATHLIB ON TAPE6
* *********************************************
      PARAMETER (NT=1000)
      PARAMETER (NVART=500)
      PARAMETER (ERROR=1.0E-4)
      PARAMETER (MXZBREN=30,ERZBREN=1.0E-7)

*        NT    : MAXIMUM NUMBER OF OBSERVATIONS.
*        NVART: MAXIMUM NUMBER OF JUMPS PERMITTED.
*        ERROR: STOP PARAMETER.
*        MXZBREN, ERZBREN: STOP PARAMETERS FOR ZBREN.
* *********************************************
      INTEGER IGRX(NT,2),IBLOK(NVART),NTBLOK,ICLASS,KLAAR,
     +          NTITERA,NTSIMUL,ITMAX
      REAL   SPR(NT),F(NVART),PG(NVART),
     +       PGNRM,GAMMA

* IGRX   : LIMITS FOR THE INTERVALS OF X, FIRST RANKS, LATER JUMPS.
* IBLOK : LIST OF ACTIVE/INACTIVE CONSTRAINTS; 0 ALS F(I-1)=F(I)
*                                              1 ALS F(I-1)<F(I)
* NTBLOK: NUMBER OF BLOCKS.
* ICLASS- 1 F(X)=X,G(T)=T,G(U)=(U-T)/(1-T)
*         2 F(X)=X,G(T)=SQRT(T),G(U)=(U-T)/(1-T)
*         3 F(X)=SQRT(X),G(T)=T,G(U)=(U-T)/(1-T)
*         4 F(X)=SQRT(X),G(T)=SQRT(T),G(U)=(U-T)/(1-T)
*         5 F(X)=X*X,G(T)=T,G(U)=(U-T)/(1-T)
*         6 F(X)=X*X,G(T)=SQRT(T),G(U)=(U-T)/(1-T)
*         7 F(X)=X,G(T)=T*T,G(U)=(U-T)/(1-T)
* NTITERA: NUMBER OF ITERATIONS FOR REACHING THE SOLUTION.
* PG     : DIRECTION IN WHICH A MAXIMUM IS SEARCHED.
* PGNRM  : NORM OF PG.
* GAMMA  : MEASURE FOR THE GRADIENT OUT OF THE SPACE OF
*          ACTIVE CONSTRAINTS.
* SPR    : PLACE OF THE JUMPS.
* F      : ESTIMATION OF THE DISTRIBUTION FUNCTION.
* KLAAR  : GIVES THE STATE OF THE SOLUTION;
*          0 - SOLUTION NOT FOUND YET.
*          1 - SOLUTION FOUND.
*          2 - SOLUTION ESTIMATED.
*          3 - MAXIMUM NUMBER OF ITERATIONS REACHED.
* *********************************************
      COMMON/FUN/IGRX
      COMMON/SIZE/N,NV,MAXN
      COMMON/SPRBLOK/SPR
      COMMON/FBLOK/F
      COMMON/PGBLOK/PG
      COMMON/IBLOCK/IBLOK
      COMMON/F0BLOK/F0(NVART)
      COMMON/F1BLOK/F1(NVART)
      COMMON/WORK1/W1(2*NT+1)
      COMMON/WORK2/W2(2*NT+1)
      COMMON/WORK3/W3(2*NT+1)
      COMMON/WORK4/W4(2*NT+1)

      EXTERNAL SIMULA,MAXBLOK,MINBLOK,PARTAN,PRINSOL,STOPT

      WRITE(2,'(''1'')')
      REWIND 1
      READ(1,*) ITMAX
    1 FORMAT(' THIS IS A PROGRAM FOR ESTIMATING THE NPMLE FOR',
     +       ' INCOMPLETE DOUBLE CENSORED OBSERVATIONS',
     +       /,' MAXIMUM NUMBER OF OBSERVATIONS:',I5,
     +       /,' MAXIMUM NUMBER OF JUMPS:       ',I5,
     +       /,' THE STOP PARAMETERS ARE:',
     +       /,' MAXIMUM NUMBER OF REFERENCES TO ZBREN:',I4,
     +       /,' MAXIMUM PERMITTED FOR GAMMA AND PGNRM:',E10.2,
     +       /,' FOR THE REFERENCES TO ZBREN WE HAVE:',
     +       /,' MAXIMUM NUMBER OF ITERATIONS         :',I4,
     +       /,' RELATIVE PRECISION                   :',E10.2)
      WRITE(2,1) NT,NVART,ITMAX,ERROR,MXZBREN,ERZBREN
   50 FORMAT(/,' NUMBER OF SIMULATIONS  :',I5,
     +       /,' NUMBER OF OBSERVATIONS :',I5,'  OF CLASS:',I4)

  100 CONTINUE
      READ(1,*,ERR=9900,END=9900) NTSIMUL,N,ICLASS
      WRITE(2,50) NTSIMUL,N,ICLASS
      IF ((ICLASS.GT.7).OR.(ICLASS.LE.0)) THEN
          WRITE(2,*) ' THIS IS NO ALTERNATIVE'
          GOTO 100
      END IF
      MAXN = 2 * N + 1
      DO 9000 I = 1, NTSIMUL
         CALL SIMULA (NT,IGRX,ICLASS)
*                USES:WORK2,WORK3 AND WORK4 LOCAL
*                     WORK1 WE NEED THIS INFORMATION IN MAXBLOK.
         KLAAR = 0
         CALL MAXBLOK(NT,IGRX)
*                USES: WORK1 WITH INFO FROM SIMULA
*                      WORK2,WORK3,WORK4 LOCAL
         IF (NV.GT.NVART) THEN
            CALL STOPT(NT, IGRX)
            GOTO 100
         END IF
*                 COMMON BLOCKS WITH WORK ARRAYS ARE ALL FREE.
         CALL MINBLOK(NT,IGRX,NTBLOK)
         CALL PARTAN(NT,IGRX,NTBLOK,NTITERA,ERROR,
     +                  ITMAX,ERZBREN,MXZBREN,PGNRM,GAMMA,KLAAR)
         CALL PRINSOL(NTBLOK,NTITERA,PGNRM,GAMMA,ICLASS,KLAAR)
 9000 CONTINUE
      GOTO 100
 9900 ENDFILE3
      WRITE(2,'(''1'')')
      RETURN
      END

* ********

      SUBROUTINE SIMULA (NT,IGRX,ICLASS)

* ICLASS- 1 F(X)=X, G(T)=T, G(U)=(U-T)/(1-T)
```

```
*         2 F(X)=X, G(T)=SQRT(T), G(U)=(U-T)/(1-T)
*         3 F(X)=SQRT(X), G(T)=T, G(U)=(U-T)/(1-T)
*         4 F(X)=SQRT(X), G(T)=SQRT(T), G(U)=(U-T)/(1-T)
*         5 F(X)=X*X, G(T)=T, G(U)=(U-T)/(1-T)
*         6 F(X)=X*X, G(T)=SQRT(T), G(U)=(U-T)/(1-T)
*         7 F(X)=X, G(T)=T*T, G(U)=(U-T)/(1-T)
* ********
* THE ARRAYS CONTAINS THE INFORMATION FOR T FROM 1 UPTO N AND FOR
* U FROM N+1 UPTO 2N.
* ********************************************
* INPUT: ICLASS,N
* *****
* OUTPUT: WORK1;TU : THE SORTED INTERVAL LIMITS T AND U.
*         IGRX     : RANKNUMBERS OF THE INTERVAL OF X.
* ********************************************
      INTEGER IGRX(NT,2),ICLASS

      COMMON/SIZE/N,NV,MAXN
      COMMON/WORK1/TU(1)
      COMMON/WORK2/TUN(1)
      COMMON/WORK3/IPERM(1)
      COMMON/WORK4/IRANK(1)

      EXTERNAL RNUNF,SVRGP

* SIMULATION OF T AND U.
      IF ((ICLASS.EQ.1).OR.(ICLASS.EQ.3).OR.(ICLASS.EQ.5)) THEN
         DO 100 I = 1, N
            HT = RNUNF()
            TUN(I) = HT
            TUN(N+I) = HT + (1.0 - HT) * RNUNF()
  100    CONTINUE
      ELSE IF ((ICLASS.EQ.2).OR.(ICLASS.EQ.4).OR.(ICLASS.EQ.6)) THEN
         DO 200 I = 1, N
            HT = RNUNF()
            HT = HT * HT
            TUN(I) = HT
            TUN(N+I) = HT + (1.0 - HT) * RNUNF()
  200    CONTINUE
      ELSE IF (ICLASS.EQ.7) THEN
         DO 300 I = 1, N
            HT = SQRT(RNUNF())
            TUN(I) = HT
            TUN(N+I) = HT + (1.0 - HT) * RNUNF()
  300    CONTINUE
      END IF
      TUN(MAXN) = 1.0
      DO 400 I = 1, MAXN
  400    IPERM(I) = I
      CALL SVRGP (MAXN,TUN,TU,IPERM)
      DO 500 I = 1, MAXN
  500    IRANK(IPERM(I)) = I
* SIMULATION OF X AND DETERMINING THE INTERVAL OF X.
      DO 600 I = 1, N
         X = RNUNF()
         IF ((ICLASS.EQ.3).OR.(ICLASS.EQ.4)) THEN
            X = X * X
         ELSE IF ((ICLASS.EQ.5).OR.(ICLASS.EQ.6)) THEN
            X = SQRT(X)
         END IF
         IF (X.GT.TUN(N+I)) THEN
            IGRX(I,1) = IRANK(N+I)
            IGRX(I,2) = MAXN
         ELSE IF (X.GT.TUN(I)) THEN
            IGRX(I,1) = IRANK(I)
            IGRX(I,2) = IRANK(N+I)
         ELSE
            IGRX(I,1) = 0
            IGRX(I,2) = IRANK(I)
         END IF
  600 CONTINUE
      RETURN
      END

* ********

      SUBROUTINE MAXBLOK (NT,IGRX)

* THIS SUBROUTINE DETERMINES A MAXIMUM NUMBER OF BLOCKS.
* ********************************************
* INPUT: WORK1;TU  : THE SORTED INTERVAL LIMITS T AND U FROM SIMULA.
*        IGRX      : RANKNUMBERS OF THE INTERVAL OF X.
* *****
* OUTPUT: IGRX     : NUMBERS OF THE JUMPS WHICH LIMITS THE INTERVAL
*                    OF X
*         SPR      : PLACE OF THE JUMPS.
*         NV       : MAXIMUM NUMBER OF JUMPS FOUND.
* ********************************************
      INTEGER IGRX(NT,2)

      COMMON/SIZE/N,NV,MAXN
      COMMON/WORK1/TU(1)
      COMMON/SPRBLOK/SPR(1)
      COMMON/WORK2/IXLP(1)
      COMMON/WORK3/IXR(1)
      COMMON/WORK4/ISP(1)

      EXTERNAL SVIGN

* COPY IGRX TO IXLP AND IXR.
      DO 10 I = 1, N
         IXLP(I) = IGRX(I,1)
         IXR(I) = IGRX(I,2)
   10 CONTINUE
      DO 15 I = 1,MAXN
         ISP(I) = 0
   15 CONTINUE
* SORT THE LEFT AND RIGHT INTERVAL LIMITS,
* WALK DOWN THROUGH THE LEFT LIMITS AND
* DETERMINE THE RIGHT LIMIT JUST ABOVE A LEFT ONE,
* THIS IS A CANDIDATE JUMP.
      CALL SVIGN(N,IXLP,IXLP)
      CALL SVIGN(N,IXR,IXR)
      IR=N
      DO 200 IL = N,1,-1
         IF (IXLP(IL).EQ.0) GOTO 300
  100    CONTINUE
         IF (IXR(IR).GT.IXLP(IL)) THEN
            IR=IR-1
            IF ( IR .EQ. 0) GOTO 300
            GOTO 100
         END IF
         ISP(IXR(IR+1)) = 1
  200 CONTINUE
  300 CONTINUE
      ISP(IXR(1)) = 1
* IXLP AND IXR ARE FREE.
* ISP GIVES THE CANDIDATE JUMPS BY AN 1.
```

```
* NEXT GIVE IXLP(I) THE NUMBER OF JUMPS UPTO AND
* INCLUDING I.
* SPR(K) IS THE VALUE OF CANDIDATE JUMP K.
      IXLP(1)=ISP(1)
      K = 0
      IF (ISP(1).EQ.1) THEN
         K = 1
         SPR(K) = TU(1)
      END IF
      DO 350 I=2,MAXN
         IXLP(I) = IXLP(I-1) + ISP(I)
         IF (ISP(I).EQ.1) THEN
            K = K+1
            SPR(K) = TU(I)
         END IF
  350 CONTINUE
      NV=IXLP(MAXN)
* CHANGE IGRX INTO THE NUMBERS OF THE FIRST AND THE
* LAST JUMP WHICH ARE COVERED BY THE INTERVAL OF X.
      DO 1000 I=1,N
         IF (IGRX(I,1).NE.0) THEN
            IGRX(I,1) = IXLP(IGRX(I,1)) + 1
         ELSE
            IGRX(I,1) = 1
         END IF
         IGRX(I,2) = IXLP(IGRX(I,2))
 1000 CONTINUE
      RETURN
      END

* *********

      SUBROUTINE STOPT (NT,IGRX)

* GIVES THE INFORMATION TO RESTART THE ANALYSIS
* IF THE PARAMETERS ARE TOO SMALL.
* ***********************************************
      COMMON/SIZE/N,NV,MAXN
      COMMON/SPRBLOK/SPR(1)

   10 FORMAT(' THE PARAMETER NVART IS TOO SMALL, WE NEED AT LEAST:',I7,
     +      /,' THE ANALYSIS IS STOPPED WITH:')
   20 FORMAT(2I5)
   30 FORMAT(I7,F15.8)
      WRITE(2,10) NV
      WRITE(2,20) (IGRX(I,2),I=1,N)
      WRITE(2,30) (I,SPR(I),I=1,NV)
      RETURN
      END

* *********

      SUBROUTINE MINBLOK (NT,IGRX,NTBLOK)

* THIS SUBROUTINE DETERMINES A MINIMUM NUMBER OF BLOCKS THAT GIVES
* A FEASIBLE POINT. HENCE WE DETERMINE A JUMP AS RIGHTMOST AS POSSIBLE
* AND IGNORE ALL THE INTERVALS WHICH CONTAIN THIS JUMP WHEN WE
* DETERMINE THE NEXT JUMP.
* ***********************************************
* INPUT: IGRX  : FIRST AND LAST JUMP COVERED BY THE INTERVAL OF X.
* *****
* OUTPUT: IBLOK,F,NTBLOK : START VALUES FOR PARTAN.
* ***********************************************
      INTEGER IGRX(NT,2),NTBLOK
      COMMON/SIZE/N,NV,MAXN
      COMMON/FBLOK/F(1)
      COMMON/IBLOCK/IBLOK(1)

      IBLOK(1) = 1
      DO 10 I = 2, NV
   10    IBLOK(I) = 0
      NTELBL = 1
      NS = 1
* NS IS THE LATEST JUMP FOUND.
  100 CONTINUE
* LG THE MINIMUM LEFT LIMIT OF INTERVALS WE CONSIDER.
      LG = NS + 1
      NS = NV
      DO 300 I = 1, N
          IF (IGRX(I,1).GE.LG) THEN
            IF (IGRX(I,2).LT.NS) NS = IGRX(I,2)
          END IF
  300 CONTINUE
      IBLOK(NS)=1
      NTELBL=NTELBL+1
      IF(NS.LT.NV) GOTO 100
      NTBLOK = NTELBL
* START VALUES FOR THE DETERMINED BLOCKS.
      F(1) = 1.0/REAL(NTELBL)
      DO 500 I = 2, NV-1
         IF (IBLOK(I).EQ.0) THEN
            F(I)=F(I-1)
         ELSE
            F(I)=F(I-1)+F(1)
         END IF
  500 CONTINUE
      F(NV) = 1.0
      RETURN
      END

* *********

      SUBROUTINE LOGLIK (NVAR,F,LGLKHD)

* WE USE F(NVAR)=1.0 .
* ***********************************************
* INPUT: F
* *****
* OUTPUT: LGLKHD
* ***********************************************
      PARAMETER (NT=1000)
* ***********************************************
      INTEGER NVAR
      REAL LGLKHD,F(NVAR)

      INTRINSIC LOG

      COMMON/SIZE/N,NV,MAXN
      COMMON/FUN/IGRX(NT,2)

      LGLKHD = 0.0
      DO 100 I = 1, N
         IL = IGRX(I,1) - 1
         IR = IGRX(I,2)
         IF (IL.EQ.0) THEN
            LGLKHD = LGLKHD + LOG(F(IR))
         ELSE
```

```
            LGLKHD = LGLKHD + LOG(F(IR) - F(IL))
          END IF
  100 CONTINUE
      RETURN
      END

* *********

      SUBROUTINE GRDLGLK (NVAR,F,GRD)

* THE GRADIENT VECTOR OF LOGLIK.
* WE USE F(NVAR)=1.0 .
* ********************************************
* INPUT: F
* *****
* OUTPUT: GRD: THE GRADIENT.
* ********************************************
      PARAMETER (NT=1000)
* ********************************************
      INTEGER NVAR
      REAL F(NVAR),GRD(NVAR)

      COMMON/SIZE/N,NV,MAXN
      COMMON/FUN/IGRX(NT,2)

      DO 100 I = 1 ,NV
  100 GRD(I) = 0.0
      DO 200 I = 1, N
         IL = IGRX(I,1) - 1
         IR = IGRX(I,2)
         IF (IL.EQ.0) THEN
            GRD(IR) = GRD(IR) + 1.0/F(IR)
         ELSE
            HLP = 1.0 / (F(IR) - F(IL))
            GRD(IL) = GRD(IL) - HLP
            GRD(IR) = GRD(IR) + HLP
         END IF
  200 CONTINUE
      GRD(NV) = 0.0
      RETURN
      END

* *********

      REAL FUNCTION GRDLBDA (BLABDA)

* THIS FUNCTION GIVES THE DIRECTIONAL DERIVATIVE OF LOGLIK AT
* (F + BLABDA * PG) WITH RESPECT TO THE VECTOR PG.
* ********************************************
* INPUT: BLABDA,F,PG
* *****
* OUTPUT: GRDLBDA
* ********************************************
      COMMON/SIZE/N,NV,MAXN
      COMMON/FBLOK/F(1)
      COMMON/PGBLOK/PG(1)
      COMMON/WORK1/Y(1)
      COMMON/WORK2/GRD(1)

      EXTERNAL GRDLGLK,SDOT

      BLBDA = BLABDA
      DO 100 I=1,NV
  100    Y(I) = F(I) + BLBDA * PG(I)
      CALL GRDLGLK (NV,Y,GRD)
      GRDLBDA = SDOT (NV,GRD,1,PG,1)
      RETURN
      END

* *********

      SUBROUTINE PROJGRD (NV,GRDF,PGNRM)

* THIS SUBROUTINE DETERMINES THE PROJECTION OF THE GRADIENT GRDF
* ONTO THE SPACE RELATED WITH THE ACTIVE CONSTRAINTS IN IBLOK.
* ********************************************
* INPUT: IBLOK,GRDF
* *****
* OUTPUT: PG   : PROJECTION OF GRDF
*         PGNRM: NORM OF PG
* ********************************************
      REAL GRDF(NV),PGNRM

      COMMON/PGBLOK/PG(1)
      COMMON/IBLOCK/IBLOK(1)

      EXTERNAL ISAMAX

      NS=1
* NS IS THE BEGINNING OF THE NEW BLOCK.
  100 CONTINUE
      I=NS
  200 CONTINUE
      I=I+1
      IF ((IBLOK(I).EQ.0).AND.(I.LT.NV)) GOTO200
* I IS NOW THE BEGINNING OF THE NEXT BLOCK.
* THE LIMITS OF THE BLOCK ARE NS AND I-1.
      SOMNS = 0.0
      DO 300 J=NS,I-1
  300    SOMNS = SOMNS + GRDF(J)
      SOMNS = SOMNS / (I-NS)
      DO 400 J=NS,I-1
  400    PG(J)=SOMNS
      NS=I
      IF (NS.LT.NV) GOTO100
      PG(NV)=0.0
      IMAX = ISAMAX(NV,PG,1)
      PGNRM = ABS(PG(IMAX))
      RETURN
      END

* *********

      SUBROUTINE SUPLBDA (NT,IGRX,SPLBDA,ISPLBDA)

* THIS SUBROUTINE DETERMINES THE MAXIMUM VALUE FOR SPLBDA
* IN F + SPLBDA * PG WHICH IS PERMIITED FOR F(I) <= F(I+1)
* ISPLBDA GIVES THE CONSTRAINT THAT CAN BECOME INACTIVE,
* AND BECAME NEGATIVE WHEN THE POINT IS NOT PERMITTED.
* INPUT: IGRX,IBLOK,F,PG
* *****
* OUTPUT: SPLBDA : MAXIMUM VALUE PERMITTED
*         ISPLBDA: RELATED CONSTRAINT
* ********************************************
      INTEGER ISPLBDA,IGRX(NT,2)
      REAL SPLBDA

      COMMON/SIZE/N,NV,MAXN
```

```
      COMMON/FBLOK/F(1)
      COMMON/PGBLOK/PG(1)
      COMMON/IBLOCK/IBLOK(1)

      SPLBDA = 0.0
      ISPLBDA = 0
      IF (PG(1).LT.0.0) THEN
         SPLBDA = (1.0/REAL(N) - F(1)) / PG(1)
         ISPLBDA = -1
      END IF
      DO 100 I=2,NV -1
         IF (PG(I-1).GT.PG(I)) THEN
            HLP=(F(I)-F(I-1))/(PG(I-1)-PG(I))
            IF ((ISPLBDA.EQ.0).OR.(HLP.LT.SPLBDA)) THEN
               SPLBDA=HLP
               ISPLBDA=I
            END IF
         END IF
  100 CONTINUE
      IF (PG(NV-1).GT.0.0) THEN
         HLP = (1.0 - 1.0/REAL(N) - F(NV-1)) / PG(NV-1)
         IF ((HLP.LT.SPLBDA).OR.(ISPLBDA.EQ.0)) THEN
            SPLBDA = HLP
            ISPLBDA = -NV
         END IF
      END IF
* THEORY GIVES SPLBDA NOT EQUAL 0 WHEN PG NOT EQUAL 0.
      IF (ISPLBDA.EQ.0) GOTO 9999
* CHECK IF SPLBDA GIVES A PERMITTED POINT.
* SEARCH THE LIMITS OF THE BLOCKS WHICH CAN BE PUT TOGETHER.
      IF (ISPLBDA.GT.0) THEN
         IL = ISPLBDA - 1
         IR = ISPLBDA + 1
  300    CONTINUE
         IF (IBLOK(IL).EQ.0) IL = IL - 1
         IF (IBLOK(IR).EQ.0) IR = IR + 1
         IF((IBLOK(IL).EQ.0).OR.(IBLOK(IR).EQ.0)) GOTO 300
         IR = IR - 1
* THE BLOCK IS FROM IL UPTO AND INCLUDING IR
         DO 500 I = 1, N
            IF ((IGRX(I,1).GT.IL).AND.(IGRX(I,2).LE.IR)) THEN
*              THIS INTERVAL IS COMPLETELY IN THE NEW BLOCK,
*              HENCE THE POINT IS NOT FEASIBLE.
               SPLBDA = SPLBDA -
     +                 (1.0/REAL(N))/(PG(ISPLBDA-1)-PG(ISPLBDA))
               ISPLBDA = -ISPLBDA
               GOTO 700
            END IF
  500    CONTINUE
      END IF
  700 CONTINUE
      RETURN
  900 FORMAT(' F + SPLBDA * PG REACHES NO LIMIT,',
     +       ' THIS IS NOT POSSIBLE. THERE MUST BE SOMETHING WRONG',
     +       ' IN THE PROGRAM.',
     +     /,' THIS WAS SUBROUTINE SUPLBDA.')
 9999 WRITE(2,900)
      STOP
      END

* *********

      SUBROUTINE VERBLOK (NV,NTELBL,GRDF,PGNRM,
     +                    GAMMA,ERROR,NEWBLOK,KLAAR)

* WHEN THE IMPROVEMENT OF THE SOLUTION IS TOO SLOW WITH THE
* ACTUAL SET OF ACTIVE CONSTRAINTS, ONE CONSTRAINT WILL BE
* DROPPED DEPENDING ON THE VALUE OF GAMMA.
* WHEN GAMMA AND PGNRM ARE BOTH SMALL ENOUGH, WE ARE CLOSE
* TO THE SOLUTION AND THE SEARCH PROCEDURE WILL TERMINATE.
* *****************************************
* INPUT: IBLOK,NTELBL,GRDF,PG,PGNRM,ERROR
* *****
* OUTPUT: IBLOK,NTELBL,KLAAR,NEWBLOK,GAMMA
* *****************************************
      INTEGER NV,NTELBL,KLAAR
      REAL GRDF(NV),PGNRM,ERROR,GAMMA
      LOGICAL NEWBLOK

      COMMON/PGBLOK/PG(1)
      COMMON/IBLOCK/IBLOK(1)

      NEWBLOK = .FALSE.
      IMAX = 0
      GAMMA = 0.0
      SOM   = 0.0
      DO 100 J = 1, NV-1
         SOM =SOM + PG(J) - GRDF(J)
         IF (SOM.GT.GAMMA) THEN
            GAMMA = SOM
            IMAX = J + 1
         END IF
  100 CONTINUE

      IF ((GAMMA.LT.ERROR).AND.(PGNRM.LT.ERROR)) THEN
         IF ((PGNRM.EQ.0.0).AND.(GAMMA.EQ.0.0)) THEN
            KLAAR = 1
         ELSE
            KLAAR = 2
         END IF
      ELSE IF (PGNRM.LT.GAMMA) THEN
         IBLOK(IMAX) = 1
         NEWBLOK = .TRUE.
         NTELBL = NTELBL + 1
      END IF
      RETURN
      END

* *********

      SUBROUTINE PARTAN (NT,IGRX,NTBLOK,NTITERA,ERROR,ITMAX,
     +                   ERZBREN,MXZBREN,PGNRM,GAMMA,KLAAR)

* THIS SUBROUTINE DETERMINES THE NPMLE WITH MODIFIED
* GRADIENT PROJECTION AND PARALLEL TANGENTS.
* *****************************************
* INPUT: IBLOK,NTBLOK,F  : STARTVALUES FROM MINBLOK.
*        KLAAR           : 0
*        ITMAX           : MAXIMUM NUMBER OF REFERENCES TO ZBREN.
*        MXZBREN         : MAXIMUM NUMBER OF ITERATIONS OF ZBREN.
*        ERZBREN         : RELATIVE PRECISION OF ZBREN.
*        ERROR           : MAXIMUM VALUE PERMITTED FOR PGNRM AND GAMMA.
*        IGRX            : FUNCTION PARAMETERS AS STATED IN MAXBLOK.
* *****
* OUTPUT: F,IBLOK,NTBLOK,NTITERA,PGNRM,GAMMA,KLAAR
* *****************************************
      INTEGER NV,ISPLBDA,ITMAX,ITEL,MAXF,
     +        IGRX(NT,2),KLAAR,MXZBREN,NTBLOK,NTELBL,NTITERA
```

```
      REAL PGNRM,SPLBDA,GAMMA,ERROR,ERZBREN,
     +     FGRDLBD,BLBDA0,ERRABS
      LOGICAL NEWBLOK

      COMMON/SIZE/N,NV,MAXN
      COMMON/FBLOK/F(1)
      COMMON/PGBLOK/PG(1)
      COMMON/IBLOCK/IBLOK(1)
      COMMON/WORK2/GRDF(1)
      COMMON/F0BLOK/F0(1)
      COMMON/F1BLOK/F1(1)

      EXTERNAL GRDLGLK,PROJGRD,VERBLOK,SUPLBDA,
     +         GRDLBDA,ZBREN,SCOPY,SAXPY

      ITEL = 0
      ERRABS = 0.0
      NTELBL = NTBLOK

  100 CALL SCOPY(NV,F,1,F0,1)
      CALL GRDLGLK (NV,F,GRDF)
  200 CALL PROJGRD (NV,GRDF,PGNRM)
      CALL VERBLOK (NV,NTELBL,GRDF,PGNRM,
     +              GAMMA,ERROR,NEWBLOK,KLAAR)
      IF (KLAAR.NE.0) GOTO 9990
      IF (NEWBLOK) GOTO 200
      CALL SUPLBDA (NT,IGRX,SPLBDA,ISPLBDA)
* GRDF IS FREE FOR USE IN GRDLBDA !!!!
      FGRDLBD = GRDLBDA(SPLBDA)
      IF (FGRDLBD.GE.0.0)THEN
            CALL SAXPY (NV,SPLBDA,PG,1,F,1)
            IF (ISPLBDA.GT.0) THEN
                IBLOK(ISPLBDA) = 0
                NTELBL = NTELBL - 1
            END IF
            GOTO 100
      END IF
      ITEL = ITEL+1
      BLBDA0 = 0.0
      MAXF = MXZBREN
      CALL ZBREN (GRDLBDA,ERRABS,ERZBREN,BLBDA0,SPLBDA,MAXF)
      CALL SAXPY (NV,SPLBDA,PG,1,F,1)
* THE VALUES OF GRDF AND PG ARE NOT NEEDED ANYMORE.
      CALL SCOPY (NV,F,1,F1,1)
      ITLPART = 0
  300 ITLPART = ITLPART + 1
      CALL GRDLGLK (NV,F,GRDF)
      CALL PROJGRD (NV,GRDF,PGNRM)
      CALL VERBLOK (NV,NTELBL,GRDF,PGNRM,
     +              GAMMA,ERROR,NEWBLOK,KLAAR)
      IF (KLAAR.NE.0) GOTO 9990
      IF (NEWBLOK) THEN
         CALL SCOPY (NV,F,1,F0,1)
         GOTO 200
      END IF
      CALL SUPLBDA (NT,IGRX,SPLBDA,ISPLBDA)
      FGRDLBD = GRDLBDA (SPLBDA)
      IF (FGRDLBD.GE.0.0) THEN
         CALL SAXPY (NV,SPLBDA,PG,1,F,1)
            IF (ISPLBDA.GT.0) THEN
                IBLOK(ISPLBDA) = 0
                NTELBL = NTELBL - 1
            END IF
         GOTO 100
      END IF

      ITEL = ITEL +1
      BLBDA0 = 0.0
      MAXF = MXZBREN
      CALL ZBREN (GRDLBDA,ERRABS,ERZBREN,BLBDA0,SPLBDA,MAXF)
      CALL SAXPY (NV,SPLBDA,PG,1,F,1)
* THE PARTAN STEP.
      DO 900 I = 1, NV
         PG(I) = F(I) - F0(I)
  900 CONTINUE
      CALL SCOPY(NV,F0,1,F,1)
      CALL SUPLBDA (NT,IGRX,SPLBDA,ISPLBDA)
      FGRDLBD = GRDLBDA (SPLBDA)
      IF (FGRDLBD.GE.0.0) THEN
         CALL SAXPY (NV,SPLBDA,PG,1,F,1)
         IF (ISPLBDA.GT.0) THEN
            IBLOK(ISPLBDA) = 0
            NTELBL = NTELBL - 1
         END IF
         GOTO 100
      END IF
      ITEL = ITEL +1
      BLBDA0 = 0.0
      MAXF = MXZBREN
      CALL ZBREN (GRDLBDA,ERRABS,ERZBREN,BLBDA0,SPLBDA,MAXF)
      CALL SAXPY (NV,SPLBDA,PG,1,F,1)
      IF (ITEL.GE.ITMAX) THEN
         KLAAR = 3
         CALL GRDLGLK(NV,F,GRDF)
         CALL PROJGRD (NV,GRDF,PGNRM)
         CALL VERBLOK (NV,NTELBL,GRDF,PGNRM,
     +                 GAMMA,ERROR,NEWBLOK,KLAAR)
      ELSE IF (ITLPART.LT.NTELBL) THEN
         CALL SCOPY(NV,F1,1,F0,1)
         CALL SCOPY(NV,F,1,F1,1)
         GOTO 300
      ELSE
         GOTO 100
      END IF
 9990 NTBLOK = NTELBL
      NTITERA = ITEL
      RETURN
      END

* *********

      SUBROUTINE PRINSOL (NTBLOK,NTITERA,PGNRM,GAMMA,ICLASS,KLAAR)

* PRINT THE RESULTS.
* STATE OF THE SOLUTION ON TAPE2.
* SOLUTION ON TAPE3, LAYOUT FOR FURTHER ANALYSIS:
*
* N  ICLASS  NV  NTBLOK  NTITERA
* I  SPR(I)  F(I)        (NTBLOK TIMES)
* LGLKHD  PGNRM  GAMMA
* ****************************************
      INTEGER NTBLOK,NTITERA,ICLASS,KLAAR
      REAL PGNRM,GAMMA,LGLKHD

      COMMON/SIZE/N,NV,MAXN
      COMMON/FBLOK/F(1)
      COMMON/SPRBLOK/SPR(1)
      COMMON/IBLOCK/IBLOK(1)
```

```
      EXTERNAL LOGLIK,ISUM

*  STATE  OF THE SOLUTION.
     1 FORMAT(/,' EXACT SOLUTION FOUND.                          ')
     2 FORMAT(/,' THE GRADPROJ. AND GAMMA ARE SMALL ENOUGH.      ')
     3 FORMAT(/,' MAXIMUM NUMBER OF REFERENCES TO ZBREN REACHED.')
     4 FORMAT(/,' NO SOLUTION FOUND.')
       IF (KLAAR.EQ.1) THEN
          WRITE(2,1)
       ELSE IF (KLAAR.EQ.2) THEN
          WRITE(2,2)
       ELSE IF (KLAAR.EQ.3) THEN
          WRITE(2,3)
       ELSE
          WRITE(2,4)
          RETURN
       END IF
   100 FORMAT(2X,5I7)
   200 FORMAT(2X,I6,2X,F8.6,2X,E23.15)
   300 FORMAT(2X,3E23.15)
       NTBLOK = ISUM(NV,IBLOK,1)
       WRITE(3,100) N,ICLASS,NV,NTBLOK,NTITERA
       DO 400 I=1,NV
          IF(IBLOK(I).EQ.1) WRITE(3,200) I,SPR(I),F(I)
   400 CONTINUE
       CALL LOGLIK (NV,F,LGLKHD)

       WRITE(3,300) LGLKHD,PGNRM,GAMMA

       RETURN
       END
```