NW

stichting
mathematisch
centrum

MC

NW

AFDELING NUMERIEKE WISKUNDE NW 18/75 OCTOBER

B. VAN DOMSELAAR & P.W. HEMKER

NONLINEAR PARAMETER ESTIMATION IN INITIAL VALUE PROBLEMS

2e boerhaavestraat 49 amsterdam

CWI BIBLIOTHEEK
3 0054 00063 3496

BIBLIOTHEEK MATHEMATISCH
AMSTERDAM
5952-84

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

AMS(MOS) subject classification scheme (1970): 65D10, 65L10, 65L99

CONTENTS

Nonlinear parameter estimation in initial value problems

by

B. van Domselaar & P.W. Hemker

ABSTRACT

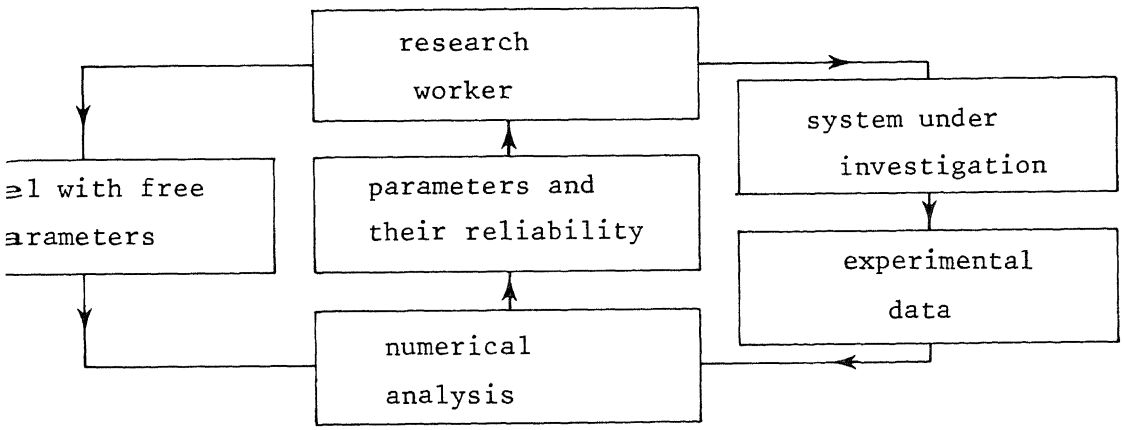
In this report an algorithm is presented for the estimation of parameters in differential equations. Parameters are obtained in the least squares sense. For the minimization for the sum of squares Marquardt's method is used. A multiple shooting technique is implemented in order to improve bad initial estimates of the parameters. The algorithm also provides confidence regions for the parameters obtained and other relevant information about the significance of the data with respect to the parameters asked for. A number of numerical results are reported and, finally, the algorithm is described in the algorithmic language ALGOL 60.

KEY WORDS & PHRASES: *Curve fitting, parameter estimation, multipoint boundary value problems, multiple shooting.*

,DUCTION

this report we consider parameter estimation in a model of a phys-
, tem, which is described by a system of ordinary differential equa-
·he parameters may appear both in the differential equations and in
, esponding initial conditions.
; ore discussing the method for estimating the parameters in a given
, e briefly go into the philosophical background of parameter esti-

, ameter estimation is an important practical problem since it estab-
, link between essential parts of experimental science. The relation
the scientist, his tools and his materials can be illustrated in
L owing scheme:



is fundamental that one never can prove a model to be the correct
ere is, however, the possibility to reject the wrong ones, namely
discrepancy appears between the data and the model. It is important
what role the parameters play in this context: each parameter rep-
a degree of freedom in the model. The parameters have to be chosen
at the model is *consistent* with the system under investigation, that
re is no discrepancy between data and model. Moreover, it is often
le that the model is *unique*, that is, there is no freedom left.
orrect operational model (that is a model which is consistent and

unique) it should be possible to predict the future behaviour of a physical system.

Whether a model is *consistent* or not, can be concluded from a discrepancy with the data in two ways. Either a parameter is found that lies outside the allowed parameter space, or the model does not fit the data, even with the best set of parameter values. In this case we have to reject the model and to construct another one.

On the other hand we can observe that the model is not *unique* in two ways. Either the parameters can not be determined uniquely, that is, the data allow the parameters to lie in a large subspace of the allowed parameter space, or the model fits the data too well, that is, the model fits the data within an error which is less than the experimental error in the data. In this case we have to simplify the model in order to obtain a unique model.

In relation to these important questions, it is clear that not only estimates of the parameters have to be determined, but also estimates for their reliability. This reliability, in its turn, is tightly connected with the accuracy and the reproducibility of the experimental data. Yet, it is even more connected with what data are taken from the physical system.

2. THE MATHEMATICAL PROBLEM

Mathematically, the parameter estimation problem can be stated as follows: A set of n differential equations with their initial conditions is given:

$$\begin{cases} y'(t,p) = g(t,y,p) \\ y(t_0,p) = y_0(p) \end{cases}$$

where $y: \mathbb{R} \times \mathbb{R}^m \to \mathbb{R}^n$ and $g: \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ are functions, t is a real variable and $p \in \mathbb{R}^m$ represents the vector of parameters. In the process considered, p has the value p^*, which is unknown. The problem is to deduce an estimate $\bar{p}$ of the vector p^*.

Suppose we have a set of observations (t_i, y_i), $i=1,\ldots,k$, where y_i is an observed value of a component of y for the value of the argument t_i. Denote the vector of observations Y by

$$Y = (y_1,\ldots,y_k)^T$$

and the function $\hat{Y}\colon \mathbb{R}^m \to \mathbb{R}^k$ by

$$\hat{Y}(p) = (\hat{y}(t_1,p),\ldots,\hat{y}(t_k,p))^T,$$

where $\hat{y}(t_i,p)$ is that component of $y(t_i,p)$ which corresponds to the observed value y_i, $1 \le i \le k$.

Now, we may define the residual function $f\colon \mathbb{R}^m \to \mathbb{R}^k$, depending on p, by:

$$f(p) = \hat{Y}(p) - Y.$$

Then the problem consists of the determination of that vector p which minimizes the sum of squares

$$F(p) = f^T(p)\, f(p) = \|f(p)\|_2^2, \tag{2.1}$$

where $\|\ \|_2$ is the Euclidean norm.

Minimizing the sum of squares has the advantage of computational convenience. Moreover, under the assumption that the error in the data are statistically independent and normally distributed with zero mean and variance σ^2, least squares has the expedient property that confidence intervals for the computed parameters can easily be derived. The assumption on the vector of errors η in the vector of observations Y is denoted by the covariance matrix of η

$$E(\eta\eta^T) = \sigma^2 I,$$

where I is the unit matrix.

For the computation of the vector of theoretical values $\hat{Y}(p)$ we use a numerical integration process.

3. MINIMIZING THE SUM OF SQUARES

In this section we give an outline of the method used for minimizing F(p). Details of the method have been published in BUS et al. [3].
Denote the gradient vector $\nabla F(p)$ by

$$\nabla F(p) = 2\, J^T(p)\, f(p), \tag{3.1}$$

where J(p) is the Jacobian matrix of partial derivatives of the residual function with the entries

$$J_{ij}(p) = \frac{\partial f_i(p)}{\partial p_j} = \frac{\partial \hat{Y}_i(p)}{\partial p_j}, \quad i=1,\ldots,k, \quad j=1,\ldots,m.$$

Starting with an initial approximate p, we seek a stepvector δp, i.e. a correction to p, such that

$$\nabla F(p+\delta p) = 0. \tag{3.2}$$

A Taylor series expansion of (3.2) yields

$$J^T(p+\delta p)\, f(p+\delta p) = J^T(p)\, f(p) + J^T(p)\, J(p)\, \delta p + \left(\frac{\partial}{\partial p} J^T(p)\right) f(p)\delta p + 0|\delta p|^2.$$

By neglecting the higher order terms and the term with the derivative of $J^T(p)$ we obtain

$$J^T(p)\, J(p)\, \delta p = -\, J^T(p)\, f(p). \tag{3.3}$$

This linear system is the starting point for the Gauss-Newton method. The disadvantage of this method is well known: it fails if the Jacobian matrix J(p) is (nearly) singular. Then the length of δp becomes too large. This problem is forestalled in the method of Marquardt (cf. MARQUARDT [8]). Here the stepvector δp is determined by

$$(J^T(p)\, J(p)+\lambda I)\delta p = -\, J^T(p)\, f(p), \tag{3.4}$$

where λ is some nonnegative scalar.
For $\lambda=0$ the vector δp is equal to the vector defined by equation (3.3).
If λ tends to infinity the direction of δp tends to the "steepest descent" direction and its length tends to zero

$$\delta p \approx - J^T(p)\, f(p)/\lambda .$$

The problem is now to find a proper choice of λ. In our implementation equation (3.4) sometimes will be solved for various values of λ. In order to avoid superfluous calculations the singular value decomposition of $J(p)$ is used:

$$J(p) = U(p)\, \Sigma(p)\, V^T(p), \tag{3.5}$$

where $U(p) \in \mathbb{R}^k \times \mathbb{R}^m$ and $V(p) \in \mathbb{R}^m \times \mathbb{R}^m$ are orthogonal matrices and $\Sigma(p) \in \mathbb{R}^m \times \mathbb{R}^m$, with $\Sigma(p) = \mathrm{diag}(\sigma_1,\ldots,\sigma_m)$ is the diagonal matrix of singular values.
Substituting (3.5) in (3.4) yields the stepvector δp by

$$\delta p = - V(p) \left((\Sigma(p))^2+\lambda I \right)^{-1} \Sigma(p)\, U^T(p)\, f(p). \tag{3.6}$$

After a successful iteration step, i.e. after a computation of δp such that $F(p+\delta p) < F(p)$, the vector $p + \delta p$ will be taken as a new initial approximation in the next iteration step. The iteration process is stopped if the change in the sum of squares is less than an a priori given tolerance. Obviously, this tolerance should depend strongly on the accuracy of the observed vector Y.

4. INTEGRATION OF THE DIFFERENTIAL EQUATIONS

For minimizing $F(p)$, we need the residual function $f(p) = \hat{Y}(p) - Y$ and the Jacobian matrix $J(p)$. For the computation of $\hat{Y}(p)$ we have to solve the system of differential equations which describes the model:

$$
\begin{aligned}
&y'(t,p) = g(t,y,p) \\
&y(t_0,p) = y_0(p).
\end{aligned} \tag{4.1}
$$

The Jacobian matrix is computed from a family of systems of differential equations derived from (4.1). For each component of p we solve $y_p = \frac{\partial}{\partial p} y(t,p)$ from the system:

$$
\begin{cases}
y_p'(t,p) = g_p(t,y,p) + y_p(t,p)\, g_y(t,y,p) \\
y_p(t_0,p) = \frac{\partial}{\partial y}\, y_0(p),
\end{cases} \tag{4.2}
$$

where $g_p(t,y,p) = \frac{\partial}{\partial p}\, g(t,y,p)$ and $g_y = \frac{\partial}{\partial y}\, g(t,y,p)$.

In this way, we have one additional set of differential equations for each component of p. This yields a large set of differential equations:

$$
\begin{cases}
y'(t,p) = g(t,y,p) \\
y'_{p_1}(t,p) = g_{p_1}(t,y,p) + g_y(t,y,p)\, y_{p_1}(t,y,p) \\
\quad\vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
y_{p_m}(t,p) = g_{p_m}(t,y,p) + g_y(t,y,p)\, y_{p_m}(t,y,p).
\end{cases} \tag{4.3}
$$

This set contains one system of nonlinear differential equations and m systems that are coupled with the first system. Each of these additional systems is linear in the dependent variable y_p. The structure of system (4.3) can be clarified by its jacobian matrix:

$$
\mathrm{JAC} = \begin{pmatrix}
g_y & & & 0 \\
g_{yp_1} & g_y & & \\
\vdots & & \ddots & \\
g_{yp_m} & 0 & & g_y
\end{pmatrix},
$$

where

$$
g_{yp_i} = \frac{\partial}{\partial y}\, g_{p_i}(t,y,p).
$$

This Jacobian matrix clearly demonstrates the one-way coupling in the system.

In addition we notice that the eigenvalues of JAC are all equivalent with the eigenvalues of g_y and thus the stability behaviour of system (4.1) and system (4.3) is similar.

For the solution of system (4.3) we use an implicit linear multistep method, viz. Gear's stiffly stable method (see HEMKER [5]). In each step of the integration process, first the independent set of n differential equations is considered. For each integration step, the computation of y implies the solution of the system of n nonlinear algebraic equations:

$$y_k = h\ \beta\ g(t,y_k,p) + \Phi_k,$$

where y_k denotes the value of y in the k-th integration step and the vector Φ_k contains information about a number of previous steps. After a choice of a suitable starting value $y_k^{(0)}$ this system is solved by a modified Newton-Raphson method:

$$(4.4) \qquad y_k^{(r+1)} = y_k^{(r)} - (I-h\beta g_y(t,y_k^{(r)},p))^{-1}\ (y_k^{(r)}-\Phi_k-h\beta g(t,y_k^{(r)},p)).$$

When this iteration process has converged, we integrate the additional m systems of linear equations directly by

$$(4.5) \qquad y_{p,k} = h\ \beta\ g_p(t,y_k,p) + h\ \beta\ g_y(t,y_k,p)\ y_{p,k} + \Psi_{p,k}.$$

Here, we use the L-U-decomposed form of $(I-h\beta g_y(t,y_k,p)$, that was already available from (4.4).

The possibility of coupling the integration of (4.1) with the integration of (4.2) with this ease, depends crucially on the linear form of the integration formula used. It cannot be achieved, for instance, with Runge-Kutta methods. By (4.4) and (4.5), we compute the residual function f(p) and the Jacobian matrix J(p) row after row, when we are integrating (4.3).

5. THE MULTIPLE SHOOTING TECHNIQUE

When starting the minimization of $F(p)$, it is also desirable to have good initial estimates of p. Because of the nonlinearity in p of $y(t,p)$, the sum of squares $F(p)$ may have more than one local minimum. We use a multiple shooting technique (see for instance STOER & BULIRSCH [9]) to create good initial estimates of p. This technique easily fits into the parameter estimation method described in section 3.

We divide the interval of integration $[t_0, t_k]$ into $\ell + 1$ parts, denoted by a set of break-points $\{T_1,\ldots,T_\ell\}$, which is a subset of $\{t_1,\ldots,t_k\}$. Denote $T_0 = t_0$ and $T_{\ell+1} = t_k$.

Let $\tilde{p} \in \mathbb{R}^{m+\ell}$ be an expanded vector of parameters such that

$$\tilde{p} = (p_1,\ldots,p_m, p_{m+1},\ldots,p_{m+\ell})^T.$$

Here p_{m+i}, $1 \le i \le \ell$, represents an estimate of a component of the vector $\hat{Y}(p^*)$, viz. $p_{m+i} \sim \hat{Y}(T_i,p^*)$.

Denote

$$y(t;T_i,\tilde{p}) = y(t,\tilde{p}) \text{ for } t \in [T_i, T_{i+1}],\ i=0,\ldots,\ell.$$

Successively, on each interval $[T_i, T_{i+1}]$, an initial value problem is solved. On the interval $[T_0, T_1]$ the integration is started with the initial conditions:

$$y(T_0;T_0,\tilde{p}) := y_0(p). \tag{5.1}$$

Thereafter, on the intervals $[T_i,T_{i+1}]$, $i=1,\ldots,\ell$, the integration is continued with the initial conditions

$$y(T_i;T_i,\tilde{p}) := y(T_i;T_{i-1},\tilde{p}), \tag{5.2}$$

for all components of y, except for that particular component of $y(T_i;T_i,\tilde{p})$ for which we have an additional estimate p_{m+i}. For this component we have

(5.3) $\hat{y}(T_i;T_i,\tilde{p}) := p_{m+i}$.

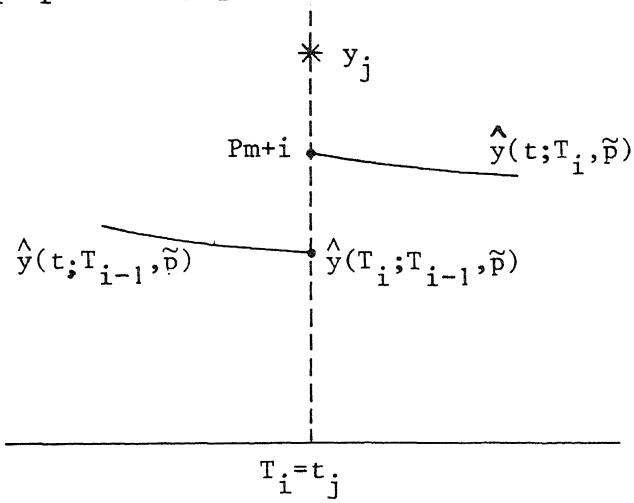


Fig. 5.1.

For these additional estimates, i.e. the extra parameters $p_{m+1},\ldots,p_{m+\ell}$, we have initial estimates available from physical knowledge, viz. the observations of $\hat{y}(T_1,p),\ldots,\hat{y}(T_\ell,p)$.

Hence, we have to solve the initial value problems

(5.4) $y'(t,\tilde{p}) = g(t,y,\tilde{p})$, $t \in [T_i,T_{i+1}]$, $i=0,\ldots,\ell$,

with the initial conditions given by (5.1), (5.2) and (5.3).
Now, beside the least squares conditions for the parameter $\tilde{p}$, we also have to satisfy the continuity conditions

$$p_{m+i} = \hat{y}(T_i;T_{i-1},\tilde{p}),\ i=1,\ldots,\ell.$$

We add these conditions with a weighting factor M, to the original least squares problem. We define the expanded residual function $\tilde{f}\colon \mathbb{R}^{m+\ell} \to \mathbb{R}^{k+\ell}$, depending on $\tilde{p}$ and M, by

$$\tilde{f}(\tilde{p},M) = \begin{pmatrix} \hat{y}(t_1,\tilde{p}) - y_1 \\ \vdots \\ \hat{y}(t_k,\tilde{p}) - y_k \\ (\hat{y}(T_1;T_0,\tilde{p}) - p_{m+1}) * M \\ \vdots \\ (\hat{y}(T_\ell;T_{\ell-1},p) - p_{m+\ell}) * M \end{pmatrix},$$

and we have to minimize the sum of squares

$$\tilde{F}(\tilde{p},M) = \tilde{f}^T(\tilde{p},M)\ \tilde{f}(\tilde{p},M).$$

The expanded Jacobian matrix $\tilde{J}(\tilde{p},M) \in \mathbb{R}^{k+\ell} \times \mathbb{R}^{m+\ell}$ is the matrix of partial derivatives of the expanded residual function and has the following form:

$$\tilde{J}(\tilde{p},M) = \begin{pmatrix} G_1 \\ G_2 \end{pmatrix},$$

where $G_1 \in \mathbb{R}^k \times \mathbb{R}^{m+\ell}$ with $(G_1)_{ij} = \dfrac{\partial \hat{y}(t_i,\tilde{p})}{\partial p_j}$

$G_2 \in \mathbb{R}^{\ell} \times \mathbb{R}^{m+\ell}$ with $(G_2)_{ij} = \left[\dfrac{\partial}{\partial p_j}\ (\hat{y}(T_i;T_{i-1},\tilde{p}) - p_{m+i}\right] * M.$

Further, the method is completely analogous to the method with a single interval of integration.
During minimizing $\tilde{F}(\tilde{p})$, we may enhance the continuity conditions by increasing the weighting factor M. On the other hand, we may omit a break-point and the corresponding parameter if the continuity condition is (almost) satisfied. Of course, we omit all extra parameters if sufficiently accurate initial estimates for p have been obtained.

The advantage of the technique discussed is that a priori knowledge about a state-variable Y is used to obtain reasonable initial estimates for p.

6. THE ALGORITHM

Using the ideas described in the previous sections we construct algorithm A. An implementation of this algorithm, written in the form of an ALGOL 60 procedure, is given in appendix I to this report. We distinguish in algorithm A the following four blocks:

A1: *initialisation.*
The following input must be given:
an approximation p_0 to p^*;
the observations (t_i,y_i), i=1,...k;

a set of break-points $\{T_1,\ldots,T_{\ell_0}\}$, which is a subset of $\{t_1,\ldots,t_k\}$;
tolerance values ε_r' and ε_a';
weighting factors $M_1,\ldots,M_s$ such that $M_{i+1} > M_i$, $i=1,\ldots,s-1$;
tolerance values $\varepsilon_r(M_1),\ldots,\varepsilon_r(M_s)$ and $\varepsilon_a(M_1),\ldots,\varepsilon_a(M_s)$.

A2: *first integration: rejecting or accepting the given break-points.*
Let p_{m+i} be that component of Y corresponding to $\hat{y}(T_i,p)$, $i=1,\ldots,\ell_0$.
The initial conditions for the integration of the system

(6.1) $\quad y'(t,\tilde{p}) = g(t,y,\tilde{p})$

on the interval $[T_0,T_1]$ are given by (5.1). Perform the following block for $i=1,\ldots,\ell_0 + 1$:

Perform the integration of (6.1) on the interval $[T_{i-1},T_i]$.

The choice of the initial conditions on the next interval are established by the following test. If the continuity condition

$$|\hat{y}(T_i;T_{i-1},\tilde{p}) - p_{m+i}| < \varepsilon_a(M_1)$$

is satisfied, then the break-point T_i and the parameter p_{m+i} are discarded and the initial conditions for the integration of (6.1) are given by (5.2). Otherwise the initial conditions are given by (5.2) and (5.3).

Suppression of the break-points that have been discarded during this first integration and subsequent renumbering of the remaining break-points yields a new set $\{T_1,\ldots,T_{\ell_1}\}$ and a new set of corresponding parameters $p_{m+1},\ldots,p_{m+\ell_1}$, where $\ell_1 \leq \ell_0$.

Block A3 is performed K times. Here K=min(t,s), where t is the smallest non-negative integer for which $\ell_{t+1} = 0$ and s is the number of given weighting factors.

A3: *iteration step,* $j=1,\ldots,K$.
Minimize $\tilde{F}(\tilde{p},M_j)$ by Marquardt iteration, with the set of break-points $\{T_1,\ldots,T_{\ell_j}\}$. Stop this process if after an iteration step it turns out that

$$\tilde{F}(\tilde{p},M_j) - \tilde{F}(\tilde{p}+\delta\tilde{p},M_j) \leq \varepsilon_r(M_j)\,\tilde{F}(\tilde{p}+\delta\tilde{p},M_j) + \varepsilon_a(M_j)$$

or

$$\tilde{F}(\tilde{p}+\delta\tilde{p},M_j) \leq \varepsilon_a(M_j).$$

Discard break-point T_i and parameter p_{m+i} if the continuity condition

$$|\hat{y}(T_i;T_{i-1},\tilde{p}) - p_{m+i}| < \varepsilon_a(M_{j+1}),\ i=1,\ldots,\ell_j,$$

is satisfied. Renumbering the remaining break-points and parameters delivers the next set $\{T_1,\ldots,T_{\ell_{j+1}}\}$ with the corresponding parameters $p_{m+1},\ldots,p_{m+\ell_{j+1}}$, where $\ell_{j+1} \leq \ell_j$.

A4: *minimization without break-points.*

Let p be an approximation to $\bar{p}$, delivered after K iteration steps A3. Minimize F(p) by Marquardt iteration and stop this process if, after an iteration step, it turns out that

$$F(p) - F(p+\delta p) \leq \varepsilon_r'\,F(p+\delta p) + \varepsilon_a' \quad \text{or} \quad F(p+\delta p) \leq \varepsilon_a',$$

then set $\bar{p} = p + \delta p$

7. STATISTICS

As was pointed out in section 1, it is essential to the problem to find not only a set of parameters $\bar{p}$, but also estimates for their reliability.

For the error η_i in the observed value y_i we assume an $N(0,\sigma^2)$ distribution and so it follows from (3.4) that the error in the estimated value $\bar{p}$ will also be normally distributed. As a first approximation, it is assumed that f(p) is linear in p in a sufficiently large neighbourhood of $\bar{p}$. When f(p) is moderately nonlinear an adjusted theory for confidence regions can be found in BEALE [2].

The approximate confidence region is that set of values of p for which

$$F(p) - F(\bar{p}) \leq m\,\sigma^2\,F_\alpha(m,k-m),$$

where $F_\alpha(m,k-m)$ is the upper α probability point of the Fisher-distribution with m and k-m degrees of freedom.
We may use

$$s^2 = \frac{F(\bar{p})}{k-m},$$

as an adequate independent estimate of σ^2.
Expanding f(p) in Taylor series and retaining only terms up to the first order yields

$$f(p) = f(\bar{p}) + J(\bar{p})\,\delta p,$$

where $\delta_p = p-\bar{p}$.
Then from (2.1) and $J^T(\bar{p})\,f(\bar{p}) = 0$, it follows immediately that

$$F(p) = F(\bar{p}) + \delta p^T\,J^T J\,\delta p,$$

where $J = J(\bar{p})$.
Thus, the confidence region will be an ellipsoid

(7.1) $$\delta p^T\,J^T J\,\delta p \leq \varepsilon,$$

where $\varepsilon = \frac{m}{k-m}\,F(\bar{p})\,F_\alpha(m,k-m)$.

From this ellipsoid we can derive some individual confidence intervals for $\bar{p}_i$. For example, the dependent interval with the ends

$$\bar{p}_i \pm \delta p_i^{(1)}, \text{ where}$$

(7.2)
$$\delta p_i^{(1)} = \sqrt{\left[\varepsilon/(J^T J)_{ii}\right]}$$

and the independent interval with the ends

$$\bar{p}_i \pm \delta p_i^{(2)}, \text{ where}$$

(7.3)

$$\delta p_i^{(2)} = \sqrt{[\varepsilon (J^T J)^{-1}_{ii}]}.$$

Geometrically regarded, the axis δp_i intercepts the ellipsoid at point $\delta p_i^{(1)}$ from the centre of the ellipsoid. The tangent planes to the ellipsoid with normals to the direction of δp_i are defined by equation (7.3). In two dimensions this is illustrated by fig. 7.1.

More information about the ellipsoid can be gained from the matrix $J^T J$. Using the singular value decomposition (cf.eq.(3.5)), the confidence region is given by

$$\delta q^T \sum^2 \delta q \leq \varepsilon,$$

where $\delta q = V^T \delta p$. This indicates that the principal axes of the ellipsoid coincide with the column vectors of V. Moreover, the lengths of the axes are:

$$\delta q_i = \sqrt{\varepsilon}/\sigma_i, \quad i=1,\ldots,m, \tag{7.4}$$

where σ_i is the i-th diagonal element of $\sum$.
Equation (7.4) states that the longest axis, corresponding to the smallest singular value, defines the worst-determined direction in the parameter space. The shortest axis (largest singular value) defines the best-determined direction.

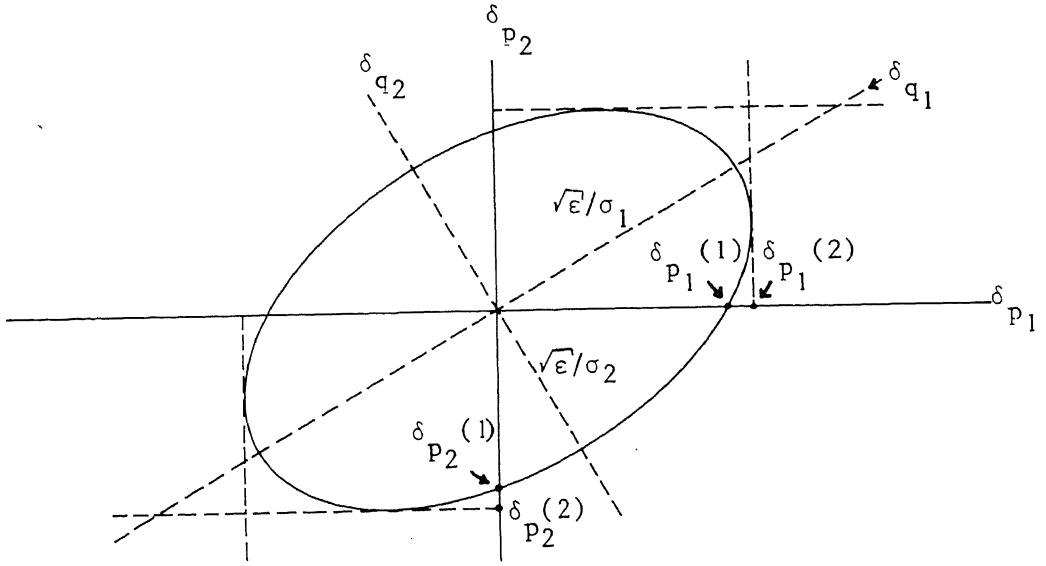


Fig. 7.1.

We denote the deviation of the least squares estimate $\bar{p}$ from the true value p^* by δp. Hence the expectation of δp will be zero and the covariance matrix of δp reads:

$$\begin{aligned} E(\delta p \delta p^T) &= E((J^TJ)^{-1} J^T f(\bar{p}) f^T(\bar{p}) J(J^TJ)^{-1}) = \\ &= (J^TJ)^{-1} J^T E(f(\bar{p})f^T(\bar{p})) J(J^TJ)^{-1} = \\ &= \sigma^2(J^TJ)^{-1} = \sigma^2 Q Q^T, \end{aligned}$$

where the columns of $Q = V\Sigma^{-1}$ are the principal axes of the ellipsoid (7.1). Since the singular value decomposition of J is available during the computation process, this information can easily be obtained. From the covariance matrix we readily derive the correlation matrix with entries

$$r_{ij} = \frac{Q_i^T Q_j}{\|Q_i\|_2 \|Q_j\|_2} = \cos(Q_i, Q_j),$$

where Q_i and Q_j are the i-th and j-th row vector of Q, respectively.

In most cases, these considerations give sufficient information about the reliability of the estimate $\bar{p}$.

8. NUMERICAL RESULTS

The ALGOL 60 version of algorithm A (see Appendix I) was tested on several problems and in this section we give the results of the tests, with some discussion. The testing has been performed on a CD-Cyber 73-28 computer.

The first five problems have been taken from HEMKER [6], where also information about the origins of the problems and the corresponding experimental data can be found.

Henceforth, a single integration of the coupled system (4.3) over the interval $[t_0, t_k]$ will be called a function evaluation. A function evaluation is performed in order to obtain the residual function f(p) as well as the Jacobian matrix F(p). For each problem, we give the number of function evaluations needed to obtain a certain relative and absolute precision for the sum of squares F(p). The values of the relative and absolute precision,

ε_r' and ε_a', have been provided, dependent on the accuracy of the data.

We will also give the relative starting value of λ used, i.e. λ_0. This value is required as an input parameter when using the implementation of the method of Marquardt, that is available in the software library NUMAL (see Hemker [7]).

In order to obtain information about the reliability of the estimates $\bar{p}$, the correlation matrix, the covariance matrix and 1% confidence intervals (c.f. eq. (7.3)) were calculated.

We have tested the algorithm with and without break-points. The use of break-points implicates one additional integration of system (4.1) (see Appendix I).

1) *The ESCEP-problem.*

This problem originates from biochemistry and represents a set of coupled chemical reactions. The mathematical model has the following form:

$$\frac{dy_1}{dt} = -(1-y_2)\,y_1 + p_2 y_2$$

$$\frac{dy_2}{dt} = p_1((1-y_2)y_1 - (p_2+p_3)y_2),$$

$$y_1(0) = 1,\ y_2(0) = 0.$$

Observations were obtained by means of a simulation run with the parameter values

$$p = (1000, 0.99, 0.01)^T.$$

The differential equation, which is stiff, was solved over the range $0 \le t \le 30$. The value of y_2 increases rapidly for a short initial period and then almost remains in a steady state.

We have tested our algorithm with five different sets of observations:

A. values of $y_1(t)$ and $y_2(t)$ from the simulation run, rounded to four decimal places, for the values
t = 0.0002 (0.0002) 0.002, 0.02(0.02) 0.1, 1.0, 2.0, 5.0 (5.0) 30.0.

B. as A, but only the observations of $y_2(t)$ were taken.

C. as B, but every second observation of B was left out.

D. as B, but with values for $t < 0.04$ excluded.

E. as B, but with values for $t > 0.04$ excluded.

In all cases, the starting values of the parameters were

$$p_0 = (1600,.8,1.2)^T,$$

and the relative starting value of λ was $\lambda_0 = {}_{10} - 2$.

The parameters in the model are the rate constants of the chemical reactions and its values are nonnegative. We take advantage of this information and scale the parameters by estimating their natural logarithms.

We tested the algorithm with three break-points and without break-points. The number of function evaluations, needed to fit the data within four decimals accuracy, are given in table 8.1.

Table 8.1

	A	B	C	D	E
0 break-points	9	13	9	12	5*)
3 break-points	11	11	12	11	7*)

*) the model did not fit the data within the accuracy of four decimals.

A complete testrun with data B is given as an illustration in Appendix I.

The model with data A,B or C is a correct operational model, since each set of data contains information from the initial as well as from the steady state period of y_2. The calculated confidence intervals of the natural logarithms of the estimates were:

$$\delta p = (\pm\ 0.32_{10}-3,\ \pm\ 0.25_{10}-3,\ \pm\ 0.23_{10}-2)^T.$$

These (absolute) confidence intervals for the logarithms are simply related to the (relative) confidence intervals of the original parameters. For ex-

ample, a deviation of 0.05 in ln(p) corresponds to a relative accuracy of about 5% in p.

The data D consist of observations only from the steady state period. The model is consistent with data D but not unique. The first parameter in the model, which is responsible for the behaviour of y_2 in the initial period, could not be determined and kept the value 1600.

With data E, which only consist of observations from the initial period, the other two parameters cannot be determined. The model seems to be not consistent with the data E. The sum of squares attained the value $F(\bar{p}) = 4_{10}-5$ and this value could not be improved. The parameters became the values:

$$\bar{p} = (997.15,\ 0.470,\ 0.526)^T$$

and the confidence intervals of their natural logarithms were

$$\delta p = (\pm\ 0.97_{10}-2,\ \pm\ 1.17,\ \pm\ 1.01)^T.$$

2) *Bellman's problem*

This test problem originates from a chemical reaction and the differential equation reads:

$$\frac{dy_1}{dt} = p_1(126.2-y_1)\ (91.9-y_1)^2 - p_2y_1^2,$$

$$y_1(1) = 0.$$

The 14 observations of y_1 are given in one decimal place accuracy. The parameters p_1 and p_2 are the rate constants of the chemical reactions and the values are small and have a different order of magnitude. Therefore it is advised to scale the parameters by estimating the logarithms. Nevertheless, we did not follow this advice in the first instance.

The starting values of the parameters were $p_0 = (_{10}-6, _{10}-4)^T$, and our algorithm found the estimates

$$\bar{p} = (0.45_{10}-5,\ 0.27_{10}-3)^T,$$

with the confidence intervals

$$\delta p = (\pm\ 0.54_{10}-6,\ \pm\ 0.19_{10}-3)^T$$

and the calculated sum of squares was $F(\bar{p}) = 22.0$.

The problem is ill conditioned and this can be seen from the condition number of $J^T(\bar{p})\ J(\bar{p})$ which was about $_{10}+6$. The influence of the value of λ_0 on the progress of the process is illustrated by table 8.2. The smaller the value of λ_0, the smaller is the number of function evaluations needed to obtain the given results. This is due to the starting values of the parameters which are close to the final estimates.

Table 8.2

	$\lambda_0 = 10^{-2}$	$\lambda_0 = 10^{-3}$	$\lambda_0 = 10^{-4}$
0 break-points	20	16	13
2 break-points	18	15	11

By estimating the natural logarithms of the parameters the problem becomes well conditioned and only 4 function evaluations were needed to obtain the same results.

3) *Gear's problem.*

This problem also originates from chemical reactions and these reactions are described by the following system of differential equations:

$$\frac{dy_1}{dt} = -\ p_1y_1 + p_2y_2(2-y_2-2y_1)$$

$$\frac{dy_2}{dt} = -\ p_3y_2 + p_4(2-y_2-2y_1)\ (1-y_2-y_1) - \frac{dy_1}{dt}\ ,$$

$$y_1(0) = 0.25,\ y_2(0) = 0.5\ .$$

The 4 observations of both y_1 and y_2 were given in 3 decimal places accuracy. The starting values of the parameters were

$$p_0 = (1,1,1,1)^T.$$

The solution found was

$$\bar{p} = (0.7943,\ 0.8441,\ 0.8956,\ 0.9433)^T$$

and the confidence intervals were

$$\delta p = (\pm\ 0.034,\ \pm\ 0.030,\ \pm\ 0.064,\ \pm\ 0.078)^T.$$

The problem was tested with $\lambda_0 = {}_{10}{}^{-4}$.

The model is not unique, since it fits the data within an error which is less than the experimental error in the data. With two break-points, 4 function evaluations were needed to obtain these results and without break-points 3 function evaluations were needed.

4) *Barnes' problem.*

The following system of differential equations describes a set of coupled chemical reactions, which has many applications in theoretical biology.

$$\frac{dy_1}{dt} = p_1 y_1 - p_2 y_1 y_2$$

$$\frac{dy_2}{dt} = p_2 y_1 y_2 - p_3 y_2,$$

$$y_1(0) = 1,\ y_2(0) = 0.3.$$

The 10 observations of both y_1 and y_2 were given in two decimal places accuracy. The results show that the model is not consistent with the data, since the sum of squares did not attain the accuracy of the data.

The starting values of the parameters were

$$p_0 = (1{,}1,\ 1.3)^T$$

and the parameters obtained were

$$\bar{p} = (0.86,\ 2.07,\ 1.81)^T$$

with the confidence intervals

$$\delta p = (\pm\ 0.19,\ \pm\ 0.35,\ \pm\ 0.38)^T$$

and the calculated sum of squares in the minimum was

$$F(\bar{p}) = 0.164.$$

The problem was tested with $\lambda_0 = 10^{-2}$.

In order to obtain these results, 6 function evaluations were needed without using break-points and 7 function evaluations were needed when using 3 break-points.

5) *The exponential fitting problem*

This artificial problem has the property that the parameters also appear in the initial values of the differential equations.
We consider the sum of exponentials

$$y_1(t) = p_5 + p_1 e^{p_2 t} + p_3 e^{p_4 t}. \tag{8.1}$$

To this function, a system of linear differential equations is associated

$$\frac{dy_1}{dt} = y_2$$

$$\frac{dy_2}{dt} = -\ p_4 p_2 y_1 + (p_4 + p_2)\ y_2 + p_4 p_2 p_5.$$

BIBLIOTHEEK MATHEMATISCH C
AMSTERDAM

The initial conditions are

$$y_1(0) = p_1 + p_3 + p_5$$
$$y_2(0) = p_1p_2 + p_3p_4.$$

The 17 observations of y_1 were obtained from equation (8.1) in an accuracy of four decimal places, using the parameter values

$$p = (-3, -20, +2, -1, +1)^T,$$

for the values t = .02(.02) .1, .2(.2) 1.0, 5.0(5.0) 20.0.
We have tested this problem with $\lambda_0 = {}_{10}{}^{-4}$ and with starting values of the parameters

$$p_0 = (-5, -10, +5, -.5, +.5)^T.$$

Both with two break-points and without break-points the algorithm attained the given solution after 10 function evaluations.

6) *The enzyme effusion after a heart-infarct* (see VAN DOMSELAAR [4])

This problem involves fitting a model of enzyme effusion into the blood after a heart-infarct. Shortly after a heart-infarct the enzyme activity in the blood is large, thereafter it decreases by demolition to its normal value. The model has the following form:

$$\frac{dy_1}{dt} = p_1y_1 + \frac{p_4}{v_1}(y_2-y_1) + p_1\, y_{1im} + v_1\, q\frac{1}{t\sqrt{2\pi}}\exp\left(-0.5\left(\frac{\ln(t)-p_2}{p_3}\right)^2\right)$$
$$\frac{dy_2}{dt} = \frac{p_4}{v_2}(y_1-y_2),$$
$$y_1(0.1) = y_2(0.1) = y_{1im},$$

where y_1 is the intravascular and y_2 the extravascular enzyme activity. The unknown parameters in the model are p_1, the constant of demolition; p_2,

a measure for the place of the maximum of the effusion; p_3, a measure for the spreading of the effusion and p_4, the permeability constant.

The following data are known (for a particular patient):

v_1 is the volume inside the blood-vessels and has the value 2.6,

v_2 is the volume outside the blood-vessels and has the value 2.7,

q is a measure for the quantity of the total extra effusion and has the value 4991,

y_{lim} is the normal effusion and has the value 27.8.

For several values of t the enzyme activity, y_1, was measured and these experimental results are given table 8.3.

Table 8.3.

t	y_1	t	y_1
2.5	20.0	44.4	58.7
3.8	23.5	47.9	41.9
7.0	63.6	53.1	40.2
10.9	267.5	59.0	31.3
15.0	427.8	65.1	30.0
18.2	339.7	73.1	30.6
21.3	331.9	81.1	23.5
22.9	243.5	91.2	24.8
24.9	212.0	101.9	26.1
26.8	164.1	115.4	33.3
30.1	112.7	138.7	17.8
34.0	88.1	163.2	16.8
37.8	76.2	186.7	16.8
42.4	62.3		

The problem was solved with $\lambda_0 = {}_{10}{}^{-1}$ and the initial estimates of the parameter values:

$$p_0 = (0.16,\ 2.6,\ 0.3,\ 0.32)^T.$$

The parameters obtained were

$$\bar{p} = (0.27,\ 2.65,\ 0.364,\ 0.21)^T$$

with the confidence intervals

$$\delta p = (\pm\ 0.08,\ \pm\ 0.32,\ \pm\ 0.098,\ \pm\ 0.29)^T$$

and the final sum of squares was

$$F(\bar{p}) = 4038.2.$$

By using 3 break-points, 16 function evaluations were needed to obtain these results and without using break-points 15 function evaluations were needed.

9. CONCLUSIONS

The method for estimating parameters in differential equations, given in HEMKER [6], formed the base for the method described in this report. The method of Hemker has been improved by using the method of Marquardt for minimizing the sum of squares and by using a multiple shooting technique for improving bad starting values of the parameters.

Algorithm A performed well on all problems that have been studied in section 8. For the problems tested, the use of break-points had almost no influence on the results. Nevertheless, we believe that the multiple shooting technique is a useful expansion of a method for estimating parameters in differential equations. Therefore, we advise to use break-points if no good initial approximation can be given.

10. REFERENCES

[1] BARD, Y., *Nonlinear parameter estimation*, Academic Press, New York (1974).

[2] BEALE, E.M.L., *Confidence regions in non-linear estimation*, J. Royal Statistical Society 22 (1960) 41-88.

[3] BUS, J.C.P., B. VAN DOMSELAAR & J. KOK, *Nonlinear least squares estimation*, Mathematical Centre report NW 17/75 (1975).

[4] DOMSELAAR, B. VAN, *A mathematical analysis of the heart-infarct* (dutch), Mathematical Centre report NN 4/74 (1974).

[5] HEMKER, P.W., *An ALGOL 60 procedure for the solution of stiff differential equations*, Mathematical Centre report MR 128/71 (1971).

[6] HEMKER P.W., *Parameter estimation in non-linear differential equations*, Mathematical Centre report MR 134/72 (1972).

[7] HEMKER, P.W. (ed.), *NUMAL, a library of numerical procedures in ALGOL-60*, Vol. 0 up to 8 (1975).

[8] MARQUARDT, D.W., *An algorithm for least-squares estimation of nonlinear parameters*, SIAM J. 11 (1963) 431-441.

[9] STOER, J. & R. BULIRSCH, *Einführung in die Numerische Mathematik 2*, Springer Verlag, Berlin (1973).

Appendix I: Description and source text of the ALGOL 60 procedure PEIDE.

The procedure PEIDE is an implementation of algorithm A in section 6. Before giving the description and source text, we describe some details of the ALGOL 60 procedure.

The method of Marquardt has been implemented in the procedure MARQUARDT, which is described in BUS, et al. [3] and inserted in HEMKER [7]. The procedure FUNCT calculates both the residual function f(p) and the Jacobian matrix J(p) by solving system (4.3).

In order to obtain information about the starting values of the parameters, the procedure PEIDE starts with calculating $f(p_0)$ without using the given break-points.

Inside the procedure, the weighting factors $M_1 = 1$, $M_2 = 4$, $M_3 = 9$ and $M_4 = 16$ are given. The corresponding tolerance values are computed by

$$\varepsilon_a(M_1) := F(p_0) \left(\frac{\varepsilon'_a}{F(p_0)}\right)^{1/4} M_1,$$

$$\varepsilon_a(M_i) := \varepsilon_a(M_{i-1}) \left(\frac{\varepsilon'_a}{F(p_0)}\right)^{1/4} M_i, \quad i=2,3,4,$$

$$\varepsilon_r(M_1) := F(p_0) \left(\frac{\varepsilon'_r}{F(p_0)}\right)^{1/4} M_1,$$

$$\varepsilon_r(M_i) := \varepsilon_r(M_{i-1}) \left(\frac{\varepsilon'_r}{F(p_0)}\right)^{1/4} M_i, \quad i=2,3,4.$$

When the procedure MARQUARDT is used, a relative starting value of λ, i.e. λ_0, is required. Inside the procedure PEIDE, MARQUARDT can be called several times and, after each call, the value of λ_0 is adjusted to the last value of λ used.

calling sequence:

the heading of the procedure peide is:

```
procedure peide(n, m, nobs, nbp, par, rv, bp, jtjinv, in, out,
   deriv, jac dfdy, jacdfdp, call ystart, data, monitor);
value n,m,nobs; integer n,m,nobs,nbp;
array par,rv,jtjinv,in,out; integer array bp;
procedure call ystart,data,monitor;
boolean procedure deriv,jac dfdy,jac dfdp;
```

the meaning of the formal parameters is:

n: <arithmetic expression>;
the number of differential equations;

m: <arithmetic expression>;
the number of unknown variables;

nobs: <arithmetic expression>;
the number of observations; nobs should satisfy nobs≥m;

nbp: <variable>;
entry: the number of break-points; if no break-points are used then set nbp=0;
exit: with normal termination of the process nbp=0; otherwise, if the process has been broken off (see out[1]), the value of nbp is the number of break-points used before the process broke off;

par: <array identifier>;
array par[1 : m+nbp];
entry: par[1:m] should contain an initial approximation to the required parameter vector;
exit: par[1:m] contains the calculated parameter vector; if out[1]>0 and nbp>0 then par[m+1:m+nbp] contains the values of the newly introduced parameters before the process broke off;

rv: <array identifier>;
array rv[1 : nobs+nbp];
exit: rv[1:nobs] contains the residual vector at the calculated minimum; if out[1]>0 and nbp>0 then rv[nobs+1:nobs+nbp] contains the additional continuity requirements at the break-points before the process broke off;

bp: <array identifier>;
integer array bp[0 : nbp];
entry: bp[i], i=1,...,n, should correspond to the index of that time of observation which will be used as a break-point (1≤bp[i]≤nobs); the break-points have to be ordered such that bp[i]≤bp[j] if i≤j;
exit: with normal termination of the process bp[1:nbp] contains no information; otherwise, if out[1]>0 and nbp>0 then bp[i], i=1,...,nbp, contains the index of that time of observation which was used as a break-point before the process broke off;

jtjinv: <array identifier>;
array jtjinv[1 : m, 1 : m];

exit: the inverse of the matrix j' × j where j denotes the matrix of partial derivatives drv[i] / dpar[j] (i=1,...,nobs ; j=1,...,m) and j' denotes the transpose of j; this matrix can be used if additional information about the result is required; e.g. statistical data such as the covariance matrix, correlation matrix and confidence intervals can easily be calculated from jtjinv and out[2];

in: <array identifier>;
array in[0 : 6];
entry: in this array the user should give some data to control the process;
in[0]: the machine precision;
for the cyber 73 a suitable value is $_{10}$-14;
in[1]: the ratio: the minimal steplength for the integration of the differential equations divided by the distance between two neighbouring observations; mostly, a suitable value is $_{10}$-4;
in[2]: the relative local error bound for the integration process; this value should satisfy in[2]≤in[3]; this parameter controls the accuracy of the numerical integration; mostly, a suitable value is in[3]/100;
in[3], in[4]:
the relative, resp. the absolute tolerance for the difference between the euclidean norm of the ultimate and penultimate residual vector;
the process is terminated if the improvement of the sum of squares is less than
in[3] × (sum of squares) + in[4] × in[4];
these tolerances should be chosen in accordance with the relative, resp. absolute errors in the observations;
note that the euclidean norm of the residual vector is defined as the square root of the sum of squares;
in[5]: the maximum number of times that the integration of the differential equations is performed;
in[6]: a starting value used for the relation between the gradient and the gauss-newton direction (see [1]); if the problem is well conditioned then a suitable value for in[6] will be 0.01; if the problem is ill conditioned then in[6] should be greater, but the value of in[6] should satisfy:
in[0] < in[6] ≤ 1/in[0];

out: <array identifier>; array out[1 : 7];
exit : in array out some by-products are delivered;
out[1]: this value gives information about the termination of the process;
out[1]=0: normal termination;
if out[1]>0 then the process has been broken off

and this may occur because of the following reasons:

out[1]=1: the number of integrations performed exceeded the number given in in[5];

out[1]=2: the differential equations are very nonlinear; during an integration the value of in[1] was decreased by a factor 10000 and it is advised to decrease in[1], although this will increase computing time;

out[1]=3: a call of deriv delivered the value false;

out[1]=4: a call of jac dfdy delivered the value false;

out[1]=5: a call of jac dfdp delivered the value false;

out[1]=6: the precision asked for can not be attained; this precision is possibly chosen too high, relative to the precision in which the residual vector is calculated (see in[3]);

out[2]: the euclidean norm of the residual vector calculated with values of the unknowns delivered;

out[3]: the euclidean norm of the residual vector calculated with the initial values of the unknown variables;

out[4]: the number of integrations performed, needed to obtain the calculated result; if out[4]=1 and out[1]>0 then the matrix jtjinv can not be used;

out[5]: the maximum number of times that the requested local error bound was exceeded in one integration; if it is a large number, it may be better to decrease the value of in[1];

out[6]: the improvement of the euclidean norm of the residual vector in the last iteration step of the process of marquardt;

out[7]: the condition number of j' × j , i.e. the ratio of its largest to smallest eigenvalues;

deriv: <procedure identifier>;
this procedure defines the right hand side of the differential equations;
the heading of this procedure should be:
boolean procedure deriv(par, y, t, df); value t;
real t; array par,y,df;
entry: par,y,t;
par[1:m] contains the current values of the unknowns and should not be altered;
y[1:n] contains the solutions of the differential equations at time t and should not be altered;
exit: array df[1 : n];
an array element df[i] should contain the right

hand side of the i-th differential equation;
after a successful call of deriv, the boolean procedure should deliver the value true;
however, if deriv delivers the value false, then the process is terminated (see out[1]);
hence, proper programming of deriv makes it possible to avoid calculation of the right hand side with values of the unknown variables which cause overflow in the computation;

jac dfdy: <procedure identifier>;
the heading of this procedure should be:
boolean procedure jac dfdy(par, y, t, fy); value t;
real t; array par,y,fy;
entry: par,y,t;
see deriv;
exit: array fy[1 : n,1 : n];
an array element fy[i,j] should contain the partial derivative of the right hand side of the i-th differential equation with respect to y[j], i.e. df[i]/dy[j];
the boolean value should be assigned to this procedure in the same way as it is done for the value of deriv;

jac dfdp: <procedure identifier>;
the heading of this procedure should be:
boolean procedure jac dfdp(par, y, t, fp); value t;
real t; array par,y,fp;
entry: par,y,t;
see deriv;
exit: array fp[1 : n,1 : m];
an array element fp[i,j] should contain the partial derivative of the right hand side of the i-th differential equation with respect to par[j], i.e. df[i]/dpar[j];
the boolean value should be assigned to this procedure in the same way as it is done for the value of deriv;

call ystart: <procedure identifier>;
this procedure defines the initial values of the initial value problem;
the heading of this procedure should be:
boolean procedure call ystart(par, y, ymax);
array par,y,ymax;
entry: par;
par[1:m] contains the current values of the unknown variables and should not be altered;
exit: y,ymax;
y[1:n] should contain the initial values to the corresponding differential equations;
the initial values may be functions of the unknown variables par; in that case, the initial values of dy/dpar also have to be supplied; note that

dy[i]/dpar[j] corresponds with y[5×n+i×n+j]
(i=1,...,n , j=1,...,m);
ymax[i], i=1,...,n, should contain a rough
estimate to the maximal absolute value of y[i]
over the igration interval;

data: <procedure identifier>;
this procedure takes the data to fit into the procedure peide;
the heading of this procedure should be:
procedure data(nobs, tobs, cobs, obs); value nobs;
integer nobs; array tobs,obs; integer array cobs;
entry: nobs;
nobs has the same meaning as in peide;
exit: array tobs[0 : nobs];
the array element tobs[0] should contain the time, corresponding to the initial values of y given in the procedure call ystart; an array element tobs[i], 1≤i≤nobs, should contain the i-th time of observation; the observations have to be ordered such that tobs[i]≤tobs[j] if i≤j;
integer array cobs[1:nobs];
an array element cobs[i] should contain the component of y observed at time tobs[i]; note that 1≤cobs[i]≤n;
array obs[1:nobs];
an array element obs[i] should contain the observed value of the component cobs[i] of y at the time tobs[i];

monitor: <procedure identifier>;
this procedure can be used to obtain information about the course of the iteration process; if no intermediate results are wanted, a dummy procedure satisfies;
the heading of this procedure should be:
procedure monitor(post,ncol,nrow,par,rv,weight,nis);
value post,ncol,nrow,weight,nis;
integer post,ncol,nrow,weight,nis; array par,rv;
inside peide, the procedure monitor is called at two different places and this is denoted by the value of post:
post=1: monitor is called after an integration of the differential equations; at this place are available: the current values of the unknown variables par[1:ncol], where ncol=m+nbp, the calculated residual vector rv[1:nrow], where nrow=nobs+nbp, and the value of nis, which is the number of integration steps performed during the solution of the last initial value problem;
post=2: monitor is called before a minimization of the euclidean norm of the residual vector with the

procedure marquardt (see section 5.1.3.1.3) is started; available are the current values of par[1:ncol] and the value of the weight, with which the continuity requirements at the break-points are added to the original least squares problem.

procedures used (see hemker[7]) :
inivec = cp31010,
inimat = cp31011,
mulvec = cp31020,
mulrow = cp31021,
dupvec = cp31030,
dupmat = cp31035,
vecvec = cp34010,
matvec = cp34011,
elmvec = cp34020,
sol = cp34051,
dec = cp34300,
marquardt = cp34440.

source text:

```
code 34444;
procedure peide(n,m,nobs,nbp,par,res,bp,jtjinv,in,out,deriv,jac dfdy,
   jac dfdp, call ystart,data,monitor);
   value n,m,nobs; integer n,m,nobs,nbp;
   array par,res,jtjinv,in,out;
   integer array bp;
   procedure  call ystart,data,monitor;
   boolean procedure deriv,jac dfdy,jacdfdp;
   begin integer i,j,extra,weight,ncol,nrow,away,npar,ii,jj,max,
      nfe,nis;
      real eps,eps1,xend,c,x,t,hmin,hmax,res1,in3,in4,fac3,fac4;
      array aux[1:3],obs[1:nobs],save[-38:6×n],tobs[0:nobs],
      yp[1:nbp+nobs,1:nbp+m],ymax[1:n],y[1:6×n×(nbp+m+1)],fy[1:n,1:n],
      fp[1:n,1:m+nbp];
      integer array cobs[1:nobs];
      boolean first,sec,clean;

      procedure inivec(l,u,a,x); code 31010;
      procedure inimat(l1,u1,l2,u2,a,x); code 31011;
      procedure mulvec(l,u,s,a,b,x); code 31020;
      procedure mulrow(l,u,i,j,a,b,x); code 31021;
      procedure dupvec(l,u,s,a,b); code 31030;
      procedure dupmat(l1,u1,l2,u2,a,b); code 31035;
      real procedure vecvec(l,u,s,a,b); code 34010;
      real procedure matvec(l,u,i,a,b); code34011;
      procedure elmvec(l,u,s,a,b,x); code34020;
      procedure sol(a,n,p,b); code 34051;
      procedure dec(a,n,aux,p); code 34300;
      procedure marquardt(m,n,p,r,c,f,j,i,o); code 34440;

      real procedure interpol(startindex,jump,k,tobsdif);
         value startindex,jump,k,tobsdif;
         integer startindex,jump,k; real tobsdif;
         begin integer i; real s,r; s:=y[startindex]; r:=tobsdif;
            for i:=1 step 1 until k do
            begin startindex:=startindex+jump;
               s:=s+y[startindex]×r; r:=r×tobsdif
            end; interpol:=s
         end interpol;

      procedure jac dydp(nrow,ncol,par,res,jac,locfunct);
         value nrow,ncol; integer nrow,ncol;
         array par,res,jac; procedure locfunct;
         begin
            dupmat(1,nrow,1,ncol,jac,yp)
         end jacobian;
```

```
boolean procedure funct(nrow,ncol,par,res);
   value nrow,ncol; integer nrow,ncol; array par,res;
   begin integer l,k,knew,fails,same,kpold,n6,nnpar,j5n,
      cobsii;
      real xold,hold,a0,tolup,tol,toldwn,tolconv,h,ch,chnew,
      error,dfi,tobsdif;
      boolean evaluate,evaluated,decompose,conv;
      array a[0:5],delta,last delta,df,y0[1:n],jacob[1:n,1:n];
      integer array p[1:n];

      real procedure norm2(ai); real ai;
         begin real s,a; s:= 10-100;
            for i:= 1 step 1 until n do
            begin a:= ai/ymax[i]; s:= s + a × a end;
          norm2:= s
         end norm2;

      procedure reset;
         begin if ch < hmin/hold then ch:= hmin/hold else
            if ch > hmax/hold then ch:= hmax/hold;
            x:= xold; h:= hold × ch; c:= 1;
            for j:= 0 step n until k×n do
            begin for i:= 1 step 1 until n do
               y[j+i]:= save[j+i] × c;
               c:= c × ch
            end;
            decompose:=true
         end reset;

      procedure order;
         begin c:= eps × eps; j:= (k-1) × (k + 8)/2 - 38;
            for i:= 0 step 1 until k do a[i]:= save[i+j];
            j:= j + k + 1;
            tolup  := c × save[j];
            tol    := c × save[j + 1];
            toldwn := c × save[j + 2];
            tolconv:= eps/(2 × n × (k + 2));
            a0:= a[0];  decompose:= true;
         end order;

      procedure evaluate jacobian;
         begin evaluate:= false;
            decompose:= evaluated:= true;
            if ¬ jac dfdy(par,y,x,fy) then
            begin save[-3]:=4; goto return end;
         end evaluate jacobian;
```

```
   procedure decompose jacobian;
      begin decompose:= false;
         c:= -a0 × h;
         for j:= 1 step 1 until n do
         begin for i:= 1 step 1 until n do
            jacob[i,j]:= fy[i,j] × c;
            jacob[j,j]:= jacob[j,j] + 1
         end;
         dec(jacob,n,aux,p)
      end decompose jacobian;

   procedure calculate step and order;
      begin real a1,a2,a3;
         a1:= if k ≤ 1 then 0 else
              0.75 × (toldwn/norm2(y[k×n+i])) ↑ (0.5/k);
         a2:= 0.80 × (tol/error) ↑ (0.5/(k + 1));
         a3:= if k ≥ 5 ∨ fails ≠ 0
              then 0 else
              0.70 × (tolup/norm2(delta[i] - last delta[i]))↑
              (0.5/(k+2));
         if a1 > a2 ∧ a1 > a3 then
         begin knew:= k-1; chnew:= a1 end else
         if a2 > a3 then
         begin knew:= k  ; chnew:= a2 end else
         begin knew:= k+1; chnew:= a3 end
      end calculate step and order;

   if sec then begin sec:=false; goto return end;
   npar:=m; extra:=nis:=0; ii:=1;
   jj:=if nbp=0 then 0 else 1;
   n6:=n×6;
   inivec(-3,-1,save,0);
   inivec(n6+1,(6+m)×n,y,0);
   inimat(1,nobs+nbp,1,m+nbp,yp,0);
   t:=tobs[1]; x:=tobs[0];
   call ystart(par,y,ymax);
   hmax:=tobs[1]-tobs[0]; hmin:=hmax×in[1];
   evaluate jacobian; nnpar:=n×npar;

new start:
   k:= 1; kpold:=0; same:= 2; order;
   if ¬ deriv(par,y,x,df) then
   begin save[-3]:=3; goto return end;
   h:=sqrt(2 × eps/sqrt(norm2 (matvec(1,n,i,fy,df))));
   if h > hmax then h:= hmax else
   if h < hmin then h:= hmin;
   xold:= x; hold:= h; ch:= 1;
   for i:= 1 step 1 until n do
   begin save[i]:=y[i]; save[n+i]:=y[n+i]:=df[i]×h end;
   fails:= 0;
```

```
for l:= 0 while x < xend do
begin if x + h ≤ xend then x:= x + h else
   begin h:= xend-x; x:= xend; ch:= h/hold; c:= 1;
      for j:= n step n until k×n do
      begin c:= c× ch;
         for i:= j+1 step 1 until j+n do
         y[i]:= y[i] × c
      end;
      same:= if same<3 then 3 else same+1;
   end;

   comment prediction;
   for l:= 1 step 1 until n do
   begin for i:= 1 step n until (k-1)×n+1 do
      for j:= (k-1)×n+1 step -n until i do
      y[j]:= y[j] + y[j+n];
      delta[l]:= 0
   end;  evaluated:= false;

   comment correction and estimation local error;
   for l:= 1,2,3 do
   begin if ¬ deriv(par,y,x,df) then
      begin save[-3]:=3; goto return end;
      for i:= 1 step 1 until n do
      df[i]:= df[i] × h - y[n+i];
      if evaluate then evaluate jacobian;
      if decompose then decompose jacobian;
      sol(jacob,n,p,df);
      conv:= true;
      for i:= 1 step 1 until n do
      begin dfi:= df[i];
         y[  i]:= y[  i] + a0 × dfi;
         y[n+i]:= y[n+i] +      dfi;
         delta[i]:= delta[i] +  dfi;
         conv:= conv ∧ abs(dfi) < tolconv × ymax[i]
      end;
      if conv then
      begin error:= norm2(delta[i]);
         goto convergence
      end
   end;

   comment acceptance or rejection;
   if ¬ conv then
   begin if ¬ evaluated then evaluate:= true
      else
      begin ch:=ch/4; if h<4×hmin then
         begin save[-1]:= save[-1]+10;
            hmin:=hmin/10;
            if save[-1]>40 then goto return
         end
      end;
      reset
   end else convergence:
```

```
if error > tol then
begin fails:= fails + 1;
   if h > 1.1 × hmin then
   begin if fails > 2 then
      begin  reset; goto new start
      end else
      begin calculate step and order;
         if knew ≠ k then
         begin k:= knew; order end;
         ch:= ch × chnew; reset
      end
   end else
   begin if k = 1 then
      begin comment violate eps criterion;
         save[-2]:= save[-2] + 1;
         same:= 4; goto error test ok
      end;
      k:=1; reset; order; same:= 2
   end
end else error test ok:

begin fails:= 0;
   for i:= 1 step 1 until n do
   begin c:= delta[i];
      for l:= 2 step 1 until k do
      y[l×n+i]:= y[l×n+i] + a[l] × c;
      if abs(y[i]) > ymax[i] then
      ymax[i]:=  abs(y[i])
   end;
   same:= same-1;
   if same= 1 then
   dupvec(1,n,0,last delta,delta) else
   if same= 0 then
   begin calculate step and order;
      if chnew > 1.1 then
      begin
         if k ≠ knew then
         begin if knew > k then
            mulvec(knew×n+1,knew×n+n,-knew×n,y,delta,
            a[k]/knew);
            k:= knew; order
         end;
         same:= k+1;
         if chnew × h > hmax
         then chnew:= hmax/h;
         h:= h × chnew; c:= 1;
         for j:= n step n until k×n do
         begin c:= c × chnew;
            mulvec(j+1,j+n,0,y,y,c)
         end; decompose:=true
      end
      else same:= 10
   end of a single integration step of y;
   nis:=nis+1;
```

```
comment start of a integration step of yp;
if clean then
begin hold:=h; xold:=x; kpold:=k; ch:=1;
   dupvec(1,k×n+n,0,save,y)
end else
begin if h≠hold then
   begin ch:=h/hold; c:=1;
      for j:=n6+nnpar step nnpar until
      kpold×nnpar+n6 do
      begin c:=c×ch;
         for i:=j+1 step 1 until j+nnpar do
         y[i]:=y[i]×c
      end; hold:=h
   end;
   if k>kpold then
   inivec(n6+k×nnpar+1,n6+k×nnpar+nnpar,y,0);
   xold:= x; kpold:= k; ch:= 1;
   dupvec(1,k×n+n,0,save,y);
   evaluate jacobian;
   decompose jacobian;
   if ¬ jac dfdp(par,y,x,fp) then
   begin save[-3]:=5; goto return end;
   if npar>m then inimat(1,n,m+1,npar,fp,0);

   comment prediction;
   for l:=0 step 1 until k-1 do
   for j:=k-1 step -1 until l do
   elmvec(j×nnpar+n6+1,j×nnpar+n6+nnpar,nnpar,y,y,1);

   comment correction;
   for j:=1 step 1 until npar do
   begin j5n:=(j+5)×n;
      dupvec(1,n,j5n,y0,y);
      for i:=1 step 1 until n do df[i]:=
      h×(fp[i,j]+matvec(1,n,i,fy,y0))
      -y[nnpar+j5n+i];
      sol(jacob,n,p,df);
      for l:=0 step 1 until k do
      begin i:=l×nnpar+j5n;
         elmvec(i+1,i+n,-i,y,df,a[l])
      end
   end
end;

for l:=0 while x≥t do
begin
   comment calculation of a row of the jacobian
           matrix and an element of the residual
           vector;
   tobsdif:=(tobs[ii]-x)/h; cobsii:=cobs[ii];
   res[ii]:=interpol(cobsii,n,k,tobsdif)-obs[ii];
```

```
            if ¬ clean then
            begin for i:=1 step 1 until npar do
               yp[ii,i]:=interpol(cobsii+(i+5)×n,nnpar,k,
                                  tobsdif);
            comment introducing of break-points;
               if bp[jj]≠ii then else
               if first ∧ abs(res[ii])<eps1 then
               begin nbp:=nbp-1; dupvec(jj,nbp,1,bp,bp);
                  bp[nbp+1]:=0
               end else
               begin extra:=extra+1;
                  if first then par[m+jj]:=obs[ii];
                  comment introducing a jacobian row and a
                            residual vector element for
                            continuity requirements;
                  yp[nobs+jj,m+jj]:=-weight;
                  mulrow(1,npar,nobs+jj,ii,yp,yp,weight);
                  res[nobs+jj]:=weight×(res[ii]+obs[ii]-
                                  par[m+jj])
               end
            end;
            if ii=nobs then goto return else
            begin t:=tobs[ii+1];
               if bp[jj]=ii ∧ jj<nbp then jj:=jj+1;
               hmax:=t-tobs[ii]; hmin:=hmax×in[1]; ii:=ii+1
            end;
         end;

         comment break-points introduce new initial values
                  for y and yp;
         if extra>0 then
         begin for i:=1 step 1 until n do
            begin y[i]:=interpol(i,n,k,tobsdif);
               for j:=1 step 1 until npar do
               y[i+(j+5)×n]:=interpol(i+(j+5)×n,nnpar,k,
                                      tobsdif)
            end;
            for l:=1 step 1 until extra do
            begin  cobsii:=cobs[bp[npar-m+l]];
               y[cobsii]:=par[npar+l];
               for i:=1 step 1 until npar+extra do
               y[cobsii+(5+i)×n]:=0;
               inivec(1+nnpar+(l+5)×n,nnpar+(l+6)×n,y,0);
               y[cobsii+(5+npar+l)×n]:=1
            end;
            npar:=npar+extra; extra:=0;
            x:=tobs[ii-1]; evaluate jacobian; nnpar:=n×npar;
            goto new start
         end
      end
   end step;
```

```
    return:
      if save[-2]>max then max:=save[-2];
      funct:=save[-1]≤40 ∧ save[-3]=0;
      if ¬ first then
      monitor(1,ncol,nrow,par,res,weight,nis)
   end funct;

i:= -39;
for c:= 1,1,9,4,0,2/3,1,1/3,36,20.25,1,6/11,
         1,6/11,1/11,84.028,53.778,0.25,.48,1,.7,.2,.02,
         156.25, 108.51, .027778, 120/274, 1, 225/274,
         85/274, 15/274, 1/274, 0, 187.69, .0047361
do begin i:= i + 1; save[i]:= c end;

data(nobs,tobs,obs,cobs); weight:=1;
first:=sec:=false; clean:=nbp>0;
aux[2]:=10-12; eps:=in[2]; eps1:=10+10;
xend:=tobs[nobs]; out[1]:=0; bp[0]:=max:=0;

comment smooth integration without break-points;
if ¬ funct(nobs,m,par,res) then goto escape;
res1:=sqrt(vecvec(1,nobs,0,res,res)); nfe:=1;
if in[5]=1 then
begin out[1]:=1; goto escape end;
if clean then
begin first:=true; clean:=false;
   fac3:=sqrt(sqrt(in[3]/res1)); fac4:=sqrt(sqrt(in[4]/res1));
   eps1:=res1×fac4;
   if ¬ funct(nobs,m,par,res) then goto escape;
   first:=false
end else nfe:=0;

ncol:=m+nbp; nrow:=nobs+nbp;
sec:=true;
 in3:=in[3]; in4:=in[4]; in[3]:=res1;

begin real w; array aid[1:ncol,1:ncol];
   weight:=away:=0;
   out[4]:=out[5]:=w:=0;
   for weight:=(sqrt(weight)+1)↑2 while
   weight≠16 ∧ nbp>0 do
   begin if away=0 ∧ w≠0 then
      begin comment if no break-points were omitted then one
                          function evaluation is saved;
         w:=weight/w;
         for i:=nobs+1 step 1 until nrow do
         begin for j:=1 step 1 until ncol do
            yp[i,j]:=w×yp[i,j];
            res[i]:=w×res[i]
         end; sec:=true; nfe:=nfe-1
      end;
```

```
        in[3]:=in[3]×fac3×weight; in[4]:=eps1;
        monitor(2,ncol,nrow,par,res,weight,nis);
        marquardt(nrow,ncol,par,res,aid,funct,jac dydp,in,out);
        if out[1]>0 then goto escape;

        comment the relative starting value of lambda is
                  adjusted to the last value of lambda used;
        away:=out[4]-out[5]-1;
        in[6]:=in[6] × 5↑away × 2↑(away-out[5]);
        nfe:=nfe+out[4];
        w:=weight; eps1:=(sqrt(weight)+1)↑2×in[4]×fac4;
        away:=0;

        comment useless break-points are omitted;
        for j:=1 step 1 until nbp do
        begin if abs(obs[bp[j]]+res[bp[j]]-par[j+m])<eps1
           then
           begin nbp:=nbp-1; dupvec(j,nbp,1,bp,bp);
              dupvec(j+m,nbp+m,1,par,par);
              j:=j-1; away:=away+1; bp[nbp+1]:=0
           end
        end;
        ncol:=ncol-away; nrow:=nrow-away
      end;

      in[3]:=in3; in[4]:=in4; nbp:=0; weight:=1;
      monitor(2,m,nobs,par,res,weight,nis);
      marquardt(nobs,m,par,res,jtjinv,funct,jac dydp,in,out);
      nfe:=out[4]+nfe
    end;
  escape: if out[1]=3 then out[1]:=2 else
          if out[1]=4 then out[1]:=6;
          if save[-3]≠0 then out[1]:=save[-3];
          out[3]:=res1;
          out[4]:=nfe;
          out[5]:=max
end peide;
```

Appendix II: An example of a computer run.

In this Appendix we give the complete testrun of the ESCEP-model with data B (see section 8).

```
code 34445;

comment the following procedure takes care of the output of the
        example program. it also interprets the numerical data
        that can be used to obtain statistical results;

procedure communication(post,fa,n,m,nobs,nbp,par,res,bp,jtjinv,
                        in,out,weight,nis);
value post,fa,n,m,nobs,nbp,weight,nis;
integer post,n,m,nobs,nbp,weight,nis; real fa;
array par,res,bp,jtjinv,in,out;
begin integer i,j; real c; array conf[1:m];
 real procedure vecvec(l,u,s,a,b); code 34010;
 if post=5 then
 begin output(61,‹x,/,10b,‹the first residual vector›,//,16b,
   ‹i›,4b,‹res[i]›,/›);
   for i:=1 step 1 until nobs do
   output(61,‹15b,zd,2b,+.4d10+zd,/›,i,res[i]);
 end else if post=3 then
 begin output(61,‹x,/,
   ‹the euclidean norm of the residual vector:›,
   .7d10+zd,2/,5b,‹calculated parameters›,/›,
   sqrt(vecvec(1,nobs,0,res,res)));
   for i:=1 step 1 until m do
   output(61,‹9b,+.7d10+zd,/›,par[i]);
   output(61,‹/,
   ‹number of integration steps performed: ›,zzd,//›,nis);
 end else if post=4 then
 begin if nbp=0 then output(61,‹x,//,5b,
   ‹the minimization is started without break-points››) else
   begin output(61,‹x,5/,20b,
     ‹the minimization is started with  w e i g h t =›,zd,
     3/›,weight);
     output(61,‹/,5b,
     ‹the extra parameters are the observations:››);
     for i:=1 step 1 until nbp do
     output(61,‹8b,zd,2b›,bp[i]);
   end;
   output(61,‹6/,10b,
   ‹starting values of the parameters›,/›);
   for i:=1 step 1 until m do
   output(61,‹20b,+.7d10+zd,/›,par[i]);
   output(61,‹//,
   ‹rel. tolerance for the eucl. norm of the res. vector:›
   ,b,.7d10+zd,/,
```

```
  ¢abs. tolerance for the eucl. norm of the res. vector:¢
  ,b,.7d10+zd,/,¢relative starting value of lambda¢,19b,
  ¢:¢,b,.7d10+zd¢,in[3],in[4],in[6])
end else if post=1 then
begin
output(61,¢10b,¢starting values of the parameters¢,/¢);
for i:=1 step 1 until m do
output(61,¢20b,+.7d10+zd,/¢,par[i]);
output(61,¢2/,¢number of equations¢,3b,¢:¢,zd,/,
¢number of observations:¢,zd,2/,
¢machine precision¢,30b,¢:¢,+.d10+zd,/,
¢relative local error bound for integration¢,5b,¢:¢,+.d10+zd,/,
¢relative tolerance for residue¢,17b,¢:¢,+.2d10+zd,/,
¢absolute tolerance for residue¢,17b,¢:¢,+.2d10+zd,/,
¢maximum number of integrations to perform¢,6b,¢:¢,zzd,/,
¢relative starting value of lambda¢,14b,¢:¢,+.2d10+zd,/,
¢relative minimal steplength¢,20b,¢:¢,+.2d10+zd,/¢,
n,nobs,in[0],in[2],in[3],in[4],in[5],in[6],in[1]);
if nbp=0 then output(61,¢//,
¢there are no break-points¢¢) else
begin output(61,¢//,
  ¢break-points are the observations :¢¢);
  for i:=1 step 1 until nbp do
  output(61,¢zzd,b¢,bp[i])
end;
output(61,¢//,
¢the alpha-point of the f-distibution :¢,
zd.dd¢,fa);
end else if post=2 then
begin output(61,¢×¢); if out[1]=0 then output(61,¢2/,
¢normal termination of the process¢¢)
else if out[1]=1 then output(61,¢2/,
¢number of integrations allowed was exceeded¢¢)
else if out[1]=2 then output(61,¢2/,
¢minimal steplength was decreased four times¢¢)
else if out[1]=3 then output(61,¢2/,
¢a call of deriv delivered false¢¢)
else if out[1]=4 then output(61,¢2/,
¢a call of  jac dfdy  delivered false ¢¢)
else if out[1]=5 then output(61,¢2/,
¢a call of  jac dfdp  delivered false ¢¢)
else if out[1]=6 then output(61,¢2/,
¢precision asked for may not be attained¢¢);
if nbp=0 then output(61,¢2/,
¢last integration was performed without break-points¢¢) else
begin output(61,¢2/,
  ¢the process stopped with break-points: ¢¢);
  for i:=1 step 1 until nbp do
  output(61,¢zzd,b¢,bp[i])
end;
```

```
output(61,¢4/,
  ¢eucl. norm of the last residual vector :¢,.7d10+zd,/,
  ¢eucl. norm of the first residual vector:¢,.7d10+zd,/,
  ¢number of integrations performed¢,7b,¢:¢,zzd,/,
  ¢last improvement of the euclidean norm :¢,.7d10+zd,/,
  ¢conditon number of  j'xj¢,15b,¢:¢,.7d10+zd,/,
  ¢local error bound was exceeded (maxim.):¢,zzd,7/¢,
  out[2],out[3],out[4],out[6],out[7],out[5]);

  comment statistics for the parameters;
  output(61,¢//,b,¢parameters¢,12b,¢confidence interval¢,
  /¢);
  for i:=1 step 1 until m do
  begin conf[i]:=sqrt(mxfaxjtjinv[i,i]/(nobs-m))xout[2];
     output(61,¢+.7d10+zd,12b,+.7d10+zd,/¢,par[i],conf[i]);
  end;
  c:=if nobs=m then 0 else out[2]xout[2]/(nobs-m);
  output(61,¢5/,¢correlation matrix¢,11b,¢covariance matrix¢,
  /¢);
  for i:=1 step 1 until m do
  begin for j:=1 step 1 until m do
     begin if i=j then output(61,¢29b¢);
        if i>j then output(61,¢+.7d10+zd,b¢,
        jtjinv[i,j]/sqrt(jtjinv[i,i]xjtjinv[j,j]))
        else output(61,¢+.7d10+zd,b¢,jtjinv[i,j]xc)
     end; output(61,¢/¢);
  end; output(61,¢x¢)
  end;

  output(61,¢3/,10b,¢the last residual vector¢,//,15b,
  ¢i¢,4b,¢res[i]¢,/¢);
  for i:=1 step 1 until nobs do
  output(61,¢14b,zd,2b,+.4d10+zd,/¢,i,res[i])
end communication;
```

ıe user program reads:

```
≥gin integer i,m,n,nobs,nbp; real time,fa;
array par[1:6],res[1:26],jtjinv[1:3,1:3],in[0:6],out[1:7];
integer array bp[0:3];
>rocedure peide(n,m,no,nb,p,r,bp,j,i,o,d,jdy,jdp,cy,da,mo);
≥ode 34444;
>rocedure communication(p,f,m,n,no,np,pa,r,bp,j,i,o,w,ni);
≥ode 34445;

>oolean procedure jac dfdp(par,y,x,fp);
    real x; array par,y,fp;
    begin real y2; y2:=y[2];
        fp[1,1]:=fp[1,3]:=0;
        fp[1,2]:=y2×exp(par[2]);
        fp[2,1]:=exp(par[1])×(y[1]×(1-y2)-(exp(par[2])+exp(par[3]))×y2);
        fp[2,2]:=-exp(par[1]+par[2])×y2;
        fp[2,3]:=-exp(par[1]+par[3])×y2;
        jac dfdp:=true
    end jac dfdp;

>rocedure data(nobs,tobs,obs,cobs);
    value nobs; integer nobs;
    array tobs,obs,cobs;
    begin integer i;
        tobs[0]:=0;
        output(61,≮×,4/,4b,≮the observations were:≯,
        //,b,≮i≯,3b,≮tobs[i]≯,3b,≮cobs[i]≯,3b,
        ≮obs[i]≯,/≯);
        for i:=1 step 1 until nobs do
        begin
            input(60,≮3(n)≯,tobs[i],cobs[i],obs[i]);
            output(61,≮zd,3b,zd.4d,6b,d,6b,.4d,/≯,i,tobs[i],cobs[i],
            obs[i])
        end
    end data;

>rocedure call ystart(par,y,ymax);
    array par,y,ymax;
    begin y[1]:=ymax[1]:=ymax[2]:=1;
        y[2]:=0
    end call ystart;

>oolean procedure deriv(par,y,x,df);
    real x; array par,y,df;
    begin real y2; y2:=y[2];
        df[1]:=-(1-y2)×y[1]+exp(par[2])×y2;
        df[2]:=exp(par[1])×((1-y2)×y[1]-(exp(par[2])+exp(par[3]))×y2);
        deriv:=true
    end deriv;
```

```
boolean procedure jac dfdy(par,y,x,fy);
   real x; array par,y,fy;
   begin fy[1,1]:=-1+y[2];
      fy[1,2]:=exp(par[2])+y[1];
      fy[2,1]:=exp(par[1])×(1-y[2]);
      fy[2,2]:=-exp(par[1])×(exp(par[2])+exp(par[3])+y[1]);
      jac dfdy:=true
   end jac dfdy;

procedure monitor(post,ncol,ow,par,res,weight,nis);
 value post,ncol,nrow,weight,nis;
 integer post,ncol,nrow,weight,nis; array par,res;;

output(61,‹2/,30b,‹e s c e p - problem›,3/›);
m:= 3; n:=2; nobs:=23; nbp:=3;
par[1]:=ln(1600); par[2]:=ln(.8); par[3]:=ln(1.2); in[0]:=₁₀-14;
in[3]:=₁₀-4; in[4]:=₁₀-4; in[5]:=50; in[6]:=₁₀-2;
in[1]:=₁₀-4; in[2]:=₁₀-5;
bp[1]:=17; bp[2]:=19; bp[3]:=21;
fa:=4.94;
comment  fa  denotes the alpha-point of the fisher-distribution;

communication(1,fa,n,m,nobs,nbp,par,res,bp,jtjinv,in,out,0,0);
time:=clock;

peide(n,m,nobs,nbp,par,res,bp,jtjinv,in,out,deriv,jac dfdy,jac dfdp,
call ystart,data,monitor);

time:=clock-time;
communication(2,fa,n,m,nobs,nbp,par,res,bp,jtjinv,in,out,0,0);
output(61,‹3/,5b,
‹the calculation in peide consumed›,b,zzd.dd,2b,
‹seconds›,x›,time)
end
```

this program delivers:

e s c e p - problem

starting values of the parameters
$+.7377759_{10}$ +1
$-.2231436_{10}$ +0
$+.1823216_{10}$ +0

number of equations : 2
number of observations:23

machine precision :$+.1_{10}-13$
relative local error bound for integration :$+.1_{10}$ -4
relative tolerance for residue :$+.10_{10}$ -3
absolute tolerance for residue :$+.10_{10}$ -3
maximum number of integrations to perform : 50
relative starting value of lambda :$+.10_{10}$ -1
relative minimal steplength :$+.10_{10}$ -3

break-points are the observations : 17 19 21

the alpha-point of the f-distibution : 4.94

the observations were:

i	tobs[i]	cobs[i]	obs[i]
1	0.0002	2	.1648
2	0.0004	2	.2753
3	0.0006	2	.3493
4	0.0008	2	.3990
5	0.0010	2	.4322
6	0.0012	2	.4545
7	0.0014	2	.4695
8	0.0016	2	.4795
9	0.0018	2	.4862
10	0.0020	2	.4907
11	0.0200	2	.4999
12	0.0400	2	.4998
13	0.0600	2	.4998
14	0.0800	2	.4998
15	0.1000	2	.4998
16	1.0000	2	.4986
17	2.0000	2	.4973
18	5.0000	2	.4936
19	10.0000	2	.4872
20	15.0000	2	.4808
21	20.0000	2	.4743
22	25.0000	2	.4677
23	30.0000	2	.4610

normal termination of the process

last integration was performed without break-points

eucl. norm of the last residual vector :.1430776_{10} -3
eucl. norm of the first residual vector:.1331071_{10} +1
number of integrations performed : 12
last improvement of the euclidean norm :.2223694_{10} -4
conditon number of j'×j :.2582882_{10} +3
local error bound was exceeded (maxim.): 37

parameters	confidence interval
+.6907670_{10} +1	+.3209313_{10} -3
-.1003941_{10} -1	+.1687774_{10} -3
-.4605292_{10} +1	+.1942501_{10} -2

correlation matrix

+.3851320_{10} +0

-.2170393_{10} +0 -.6392889_{10} +0

covariance matrix

+.6949857_{10} -8 +.1407628_{10} -8 -.9129848_{10} -8

+.1922119_{10} -8 -.1414245_{10} -7

+.2546094_{10} -6

the last residual vector

i	res[i]
1	$+.1748_{10}$ -5
2	$-.2905_{10}$ -4
3	$+.2814_{10}$ -4
4	$-.3879_{10}$ -4
5	$+.3069_{10}$ -4
6	$+.3101_{10}$ -4
7	$-.2019_{10}$ -4
8	$-.3887_{10}$ -5
9	$+.1052_{10}$ -4
10	$+.1391_{10}$ -4
11	$-.5109_{10}$ -4
12	$+.2384_{10}$ -4
13	$-.1156_{10}$ -5
14	$-.2616_{10}$ -4
15	$-.5116_{10}$ -4
16	$+.2244_{10}$ -4
17	$+.6794_{10}$ -4
18	$-.1418_{10}$ -4
19	$+.2087_{10}$ -4
20	$-.1980_{10}$ -4
21	$-.3476_{10}$ -4
22	$-.2245_{10}$ -4
23	$+.1886_{10}$ -4

the calculation in peide consumed 108.57 seconds