IMM
DEPARTMENT OF MATHEMATICAL MODELLING

Technical University of Denmark
DK-2800 Lyngby – Denmark

J. No. SEP
5.1.2000
HBN/ms

SEPARABLE NONLINEAR LEAST SQUARES

Hans Bruun Nielsen

TECHNICAL REPORT

IMM-REP-2000-01

IMM

SEPARABLE NONLINEAR LEAST SQUARES

Hans Bruun Nielsen

Contents

1. Introduction

In many data fitting problems the model involves some parameters that appear linearly and some that appear nonlinearly. The problem is said to be *separable*. This property can be exploited in a least squares estimation, and a number of special purpose algorithms have been proposed, see e.g. [1], [2], [3], [5], [6]. In this report we discuss how to reformulate the problem so that we can exploit the special property with a standard programme for nonlinear least squares.

Let $(t_1, y_1), \ldots, (t_m, y_m)$ denote the data points and consider the model

$$y_i \;\simeq\; M(\mathbf{x}, \mathbf{c}, t_i) \;\equiv\; c_1 f_1(\mathbf{x}, t_i) + \cdots + c_p f_p(\mathbf{x}, t_i) \;, \tag{1.1}$$

where the $f_i : \mathbb{R}^{n+1} \mapsto \mathbb{R}$ are given functions that depend (nonlinearly) on $\mathbf{x} \in \mathbb{R}^n$ and on t. The coefficients $c_1, \ldots, c_p$ are parameters that occur linearly in the model.

Collecting all the parameters in $\mathbf{z} = \begin{bmatrix} \mathbf{x} \\ \mathbf{c} \end{bmatrix} \in \mathbb{R}^{n+p}$, the least squares estimate is defined as

$$\mathbf{z}^* = \underset{\mathbf{z} \in \mathbb{R}^{n+p}}{\operatorname{argmin}} \left\{ \phi(\mathbf{z}) \equiv \tfrac{1}{2} \mathbf{r}(\mathbf{z})^\top \mathbf{r}(\mathbf{z}) \right\} \;, \tag{1.2a}$$

where

$$\mathbf{r}(\mathbf{z}) = \mathbf{y} - \mathbf{F}(\mathbf{x})\mathbf{c} \tag{1.2b}$$

with the matrix $\mathbf{F}(\mathbf{x}) \in \mathbb{R}^{m \times p}$ defined by

$$\big(\mathbf{F}(\mathbf{x})\big)_{ij} = f_j(\mathbf{x}, t_i) \;. \tag{1.2c}$$

The least squares estimate satisfies

$$\boldsymbol{\phi}'(\mathbf{z}^*) = \mathbf{0} \;, \quad \text{where} \quad \boldsymbol{\phi}'(\mathbf{z}) = \mathbf{J}(\mathbf{z})^\top \mathbf{r}(\mathbf{z}) \;, \tag{1.3a}$$

with the Jacobian $\mathbf{J}(\mathbf{z}) \in \mathbb{R}^{m \times (n+p)}$, $(\mathbf{J})_{ij} = \dfrac{\partial r_i}{\partial z_j}$ having the form

$$\mathbf{J}(\mathbf{z}) = \left[\mathbf{J}_{\mathrm{f}}(\mathbf{z}) \;\; -\mathbf{F}(\mathbf{x}) \right] \;. \tag{1.3b}$$

The last p columns come from the "linear" parameters $\mathbf{c}$, while the first n columns in $\mathbf{J}$ are given by

$$\left(\mathbf{J}_{\mathrm{f}}(\mathbf{x})\right)_{ij} \;=\; \frac{\partial r_i}{\partial x_j}(\mathbf{z}) \;=\; -\sum_{k=1}^{p} c_k \frac{\partial f_k}{\partial x_j}(\mathbf{x}, t_i)\ . \tag{1.4a}$$

We shall use the following notation for the jth column in $\mathbf{J}_{\mathrm{f}}$,

$$\left(\mathbf{J}_{\mathrm{f}}(\mathbf{z})\right)_{:,j} = -\mathbf{F}_j'(\mathbf{x})\mathbf{c} \quad \text{with} \quad \left(\mathbf{F}_j'(\mathbf{x})\right)_{ik} \;=\; \frac{\partial f_k}{\partial x_j}(\mathbf{x}, t_i)\ . \tag{1.4b}$$

The *Gauss–Newton method* is widely used for solving nonlinear least squares problems: Given an approximation $\mathbf{z}$ to $\mathbf{z}^*$, the next approximation as found as $\mathbf{z}{+}\mathbf{h}_{\mathrm{G}}$, where

$$\mathbf{h}_{\mathrm{G}} = \underset{\mathbf{h}\,\in\,\mathbb{R}^{n+p}}{\operatorname{argmin}} \;\left\{\Phi(\mathbf{z},\mathbf{h}) \equiv \tfrac{1}{2}\|\mathbf{r}(\mathbf{z}) + \mathbf{J}(\mathbf{z})\mathbf{h}\|_2^2\right\}\ , \tag{1.5a}$$

leading to

$$\left(\mathbf{J}(\mathbf{z})^{\top}\mathbf{J}(\mathbf{z})\right)\mathbf{h}_{\mathrm{G}} = \mathbf{J}(\mathbf{z})^{\top}\mathbf{r}(\mathbf{z})\ . \tag{1.5b}$$

This is often modified by the *Levenberg–Marquardt* strategy, where $\mathbf{h}_{\mathrm{G}}$ is replaced by $\mathbf{h}_{\mathrm{M}}$ determined by

$$\left(\mathbf{J}(\mathbf{z})^{\top}\mathbf{J}(\mathbf{z}) + \mu\mathbf{I}\right)\mathbf{h}_{\mathrm{M}} = \mathbf{J}(\mathbf{z})^{\top}\mathbf{r}(\mathbf{z})\ . \tag{1.5c}$$

The damping parameter μ is adjusted during iteration according to how well $\Phi(\mathbf{z},\mathbf{h}_{\mathrm{M}})$ approximates $\phi(\mathbf{z}{+}\mathbf{h}_{\mathrm{M}})$, see [4]. In the experiments of Section 4 we used a standard Levenberg-Marquardt programme,[1)] where the damping parameter is initialized to

$$\mu^{(0)}) = \tau \cdot \max\left\{\left(\mathbf{J}(\mathbf{z}^{(0)})^{\top}\mathbf{J}(\mathbf{z}^{(0)})\right)_{ii}\right\}\ , \tag{1.6}$$

where τ and $\mathbf{z}^{(0)}$ are indata.

1) A Matlab implementation is available at http://www.imm.dtu.dk/~hbn/Software/Marquardt.m

2. Reduced Parameter Set

Instead of treating the elements in $\mathbf{c}$ as nonlinear variables we replace (1.2b) by

$$\mathbf{r}(\mathbf{x}) = \mathbf{y} - \mathbf{F}(\mathbf{x})\mathbf{c}(\mathbf{x})\ , \tag{2.1}$$

with $\mathbf{c}(\mathbf{x})$ computed as the least squares solution to $\mathbf{F}(\mathbf{x})\mathbf{c}(\mathbf{x}) \simeq \mathbf{y}$. We can express it as the solution to the normal equations

$$\mathbf{A}(\mathbf{x})\mathbf{c}(\mathbf{x}) = \mathbf{b}(\mathbf{x})\ , \tag{2.2a}$$

where

$$\mathbf{A}(\mathbf{x}) = \mathbf{F}(\mathbf{x})^{\top}\mathbf{F}(\mathbf{x})\ , \quad \mathbf{b}(\mathbf{x}) = \mathbf{F}(\mathbf{x})^{\top}\mathbf{y}\ . \tag{2.2b}$$

With the notation from (1.4b), the corresponding Jacobian matrix $\mathbf{J}(\mathbf{x})$ is given by

$$\begin{aligned}\left(\mathbf{J}(\mathbf{x})\right)_{:,j} \;=\; \mathbf{r}_j'(\mathbf{x}) \;&=\; -\mathbf{F}_j'(\mathbf{x})\mathbf{c}(\mathbf{x}) - \mathbf{F}(\mathbf{x})\mathbf{c}_j'(\mathbf{x}) \\ &=\; \left(\mathbf{J}_{\mathrm{f}}(\mathbf{x})\right)_{:,j} + \left(\mathbf{H}(\mathbf{x})\right)_{:,j}\ .\end{aligned} \tag{2.3}$$

From (2.2) we get

$$\mathbf{A}_j'(\mathbf{x})\mathbf{c}(\mathbf{x}) + \mathbf{A}(\mathbf{x})\mathbf{c}_j'(\mathbf{x}) = \mathbf{b}'(\mathbf{x})\ ,$$

or (omitting the argument $\mathbf{x}$)

$$\begin{aligned}\mathbf{A}\mathbf{c}_j' &= \mathbf{b}' - \mathbf{A}_j'\mathbf{c} \\ &= \mathbf{F}_j'^{\top}\mathbf{y} - \left(\mathbf{F}_j'^{\top}\mathbf{F} + \mathbf{F}^{\top}\mathbf{F}_j'\right)\mathbf{c} \\ &= \mathbf{F}_j'^{\top}\mathbf{r} - \mathbf{F}^{\top}\mathbf{F}_j'\mathbf{c} \\ &= \mathbf{F}_j'^{\top}\mathbf{r} + \mathbf{F}^{\top}\left(\mathbf{J}_{\mathrm{f}}(\mathbf{x})\right)_{:,j}\ , \quad j=1,\ldots,n\ .\end{aligned} \tag{2.4}$$

From this system of equations we can find $\mathbf{c}_j'$ and insert it in (2.3).

In (1.5) – (1.6) the matrix $\mathbf{J}(\mathbf{z}) \in \mathbb{R}^{m\times(n+p)}$ is replaced by the smaller matrix $\mathbf{J}(\mathbf{x}) \in \mathbb{R}^{m\times n}$. Also note that the residual $\mathbf{r}(\mathbf{x})$ (2.1) is orthogonal to the range of $\mathbf{F}(\mathbf{x})$, i.e. $\mathbf{F}(\mathbf{x})^{\top}\mathbf{r}(\mathbf{x}) = \mathbf{0}$. This implies that in the right-hand side of (1.5b-c) the jth element is

$$\begin{aligned}\left(\mathbf{J}(\mathbf{x})^\top \mathbf{r}(\mathbf{x})\right)_j &= \left(\mathbf{r}_j'(\mathbf{x})\right)^\top \mathbf{r}(\mathbf{x}) \\ &= -\left(\mathbf{F}_j'(\mathbf{x})\mathbf{c}(\mathbf{x}) + \mathbf{F}(\mathbf{x})\mathbf{c}_j'(\mathbf{x})\right)^\top \mathbf{r}(\mathbf{x}) \\ &= -\left(\mathbf{F}_j'(\mathbf{x})\mathbf{c}(\mathbf{x})\right)^\top \mathbf{r}(\mathbf{x}) \ . \end{aligned} \tag{2.5}$$

This shows that the right-hand side in (1.5b-c) is independent of $\mathbf{c}_j'(\mathbf{x})$. To get the correct matrix on the left-hand side, however, we have to include the term $\mathbf{F}(\mathbf{x})\mathbf{c}_j'(\mathbf{x})$.

Some special cases that lead to simple expressions for the Jacobian warrant attention:

1° $n=1$. This is not really special, except that (2.3) has one contribution only.

2° $p=1$. In this case $\mathbf{F}(\mathbf{x}) \in \mathbb{R}^{m\times 1}$ is a vector and $\mathbf{A}(\mathbf{x})$ and $c(\mathbf{x})$ are scalars. Further,

$$A\mathbf{c}_j' \;=\; \mathbf{F}_j'^\top \mathbf{r} - \mathbf{F}^\top \mathbf{F}_j' c \;=\; \mathbf{F}_j'^\top(\mathbf{r} - \mathbf{F}c), \quad j=1,\ldots,n \ .$$

Introducing the matrix $\mathbf{F}' = [\mathbf{F}_1' \;\cdots\; \mathbf{F}_n']$ and the vector $\mathbf{c}'$ with $(\mathbf{c}')_j = c_j'$ we get

$$\mathbf{c}' = \frac{1}{A}\mathbf{F}'^\top(\mathbf{r} - c\mathbf{F}) \ , \qquad \mathbf{J} = -c\mathbf{F}' - \mathbf{F}(\mathbf{c}')^\top \ . \tag{2.6}$$

3° $p=n$ and $f_j(\mathbf{x},t) = f_j(x_j,t)$, i.e. each of the functions f_j depends only on x_j and t. In this case all columns in $\mathbf{F}_j'(\mathbf{x})$ are zero except for $\left(\mathbf{F}_j'(\mathbf{x})\right)_{:,j}$. Therefore $\left(\mathbf{F}_j'^\top\mathbf{r}\right)_j$ is the only nonzero element in $\mathbf{F}_j'^\top\mathbf{r}$, and the other contribution to the right-hand side in (2.4) is $\mathbf{F}^\top(c_j(\mathbf{F}_j')_{:,j})$. We let

$$\mathbf{F}' = [(\mathbf{F}_1')_{:,1} \;\cdots\; (\mathbf{F}_n')_{:,n}] \ , \quad \mathbf{C}' = [\mathbf{c}_1' \;\cdots\; \mathbf{c}_n'] \ , \tag{2.7a}$$

and for any vector $\mathbf{u}$ we define the diagonal matrix

$$[\mathbf{u}] = \mathrm{diag}(\mathbf{u}) \ , \quad \text{i.e.} \quad ([\mathbf{u}])_{ij} = \begin{cases} u_i & \text{if } i=j \\ 0 & \text{otherwise} \end{cases} \ . \tag{2.7b}$$

Then we can write (2.4) as a matrix equation

$$\mathbf{AC}' = [\mathbf{F}'^\top\mathbf{r}] - \mathbf{F}^\top\mathbf{G} \quad \text{with} \quad \mathbf{G} = \mathbf{F}'[\mathrm{c}] \ (= -\mathbf{J}_\mathrm{f}\ ) \ , \tag{2.7c}$$

and (2.3) takes the form

$$\mathbf{J} = -\mathbf{G} - \mathbf{FC}' \ . \tag{2.7d}$$

3. Computational Aspects

For reasons of accuracy we recommend that $\mathbf{c}(\mathbf{x})$ defined in (2.1) be computed via *orthogonal transformation*: Determine $\widetilde{\mathbf{Q}} = \widetilde{\mathbf{Q}}(\mathbf{x}) \in \mathbb{R}^{m\times p}$ with orthonormal columns so that

$$\widetilde{\mathbf{Q}}^\top\mathbf{F} = \mathbf{R} \ , \tag{3.1a}$$

where $\mathbf{R} = \mathbf{R}(\mathbf{x}) \in \mathbb{R}^{p\times p}$ is upper triangular. Then $\mathbf{c} = \mathbf{c}(\mathbf{x})$ is determined by back substitution in

$$\mathbf{Rc} = \widetilde{\mathbf{y}} \equiv \widetilde{\mathbf{Q}}^\top\mathbf{y} \ . \tag{3.1b}$$

It can be shown that

$$\mathbf{A} = \mathbf{F}^\top\mathbf{F} = \mathbf{R}^\top\mathbf{R} \ , \tag{3.2}$$

and this factorization of $\mathbf{A}$ can be used in the solution of (2.4). Note that $\mathbf{R}$ is computed without having to suffer the loss of accuracy associated with forming the normal equations matrix.

As regards computational effort per iteration step, let us consider small problems with dense matrices. Both formulations need evaluation of the matrices $\mathbf{F}(\mathbf{x})$, $\left\{\mathbf{F}_j'(\mathbf{x})\right\}_{j=1}^n$ and $\mathbf{J}_\mathrm{f}(\mathbf{x})$. The extra work involved to compute $\mathbf{c}(\mathbf{x})$ by (3.1) is about $2p^2(m-\frac{1}{3}p)$ flops, and each $\mathbf{c}_j'(\mathbf{x})$ costs about $2p(2m+p)$ flops. Thus, the extra work involved in computing $\mathbf{J}(\mathbf{x})$ (2.3) instead of $\mathbf{J}(\mathbf{z})$ (1.3) is about $2p\big(m(p+3n) + p(n-\frac{1}{3}p)\big)$ flops.

Some work is saved, however, in the solution of (1.5c). If we form the matrix and solve the system via Cholesky factorization, then the cost of computing $\mathbf{h}_{\mathrm{M}}$ is about $(n+p)^2(m + \frac{1}{3}(n+p))$ flops. With the reduced parameter set we replace $n+p$ by n, and the overall extra cost per iteration step is about $p(m(p+4n) + pn - n^2 - p^2)$ flops. In many cases this will be small compared to the effort involved in the computation of the mp elements in $\mathbf{F}(\mathbf{x})$ and mn elements in $\mathbf{J}_{\mathrm{f}}(\mathbf{x})$. Also, in the special cases 2° and 3° above, the extra work is somewhat smaller.

4. Examples

We illustrate the effect of exploiting separability by three data fitting problems. Figure 1a shows data points and the least squares fit with the model

$$M(\mathbf{x}, \mathbf{c}, t) = c_1 e^{x_1 t} + c_2 e^{x_2 t} \ . \tag{4.1}$$

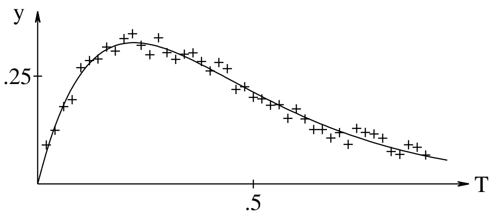

Figure 1a. *Data points (marked by +) and model $M(\mathbf{x}^*, \mathbf{c}^*, t)$ (marked by full line)*

This is a problem of the special form 3° above. We can use (2.7) with $\mathbf{F}$ and $\mathbf{F}'$ given by

$$\big(\mathbf{F}(\mathbf{x})\big)_{i,:} = [e^{x_1 t_i} \quad e^{x_2 t_i}] \ ,$$

$$\big(\mathbf{F}'(\mathbf{x})\big)_{i,:} = t_i \cdot [e^{x_1 t_i} \quad e^{x_2 t_i}] \ .$$

This means that $\mathbf{F}' = [\mathbf{t}]\mathbf{F}$.

We have used the Levenberg-Marquardt programme of footnote 1 to find an approximate solution $\mathbf{z}$ so that $\|\phi'(\mathbf{z})\|_\infty \le 10^{-8}$. The data are artificial, generated so that $\mathbf{x}^* = [\,-4, \ -5\,]^\top$, $\mathbf{c}^* = [\,4, \ -4\,]^\top$. We used the starting point $\mathbf{x}^{(0)} = [\,-3, \ -6\,]^\top$, and for the starting vector for the "full" version we took the least squares solution to $\mathbf{F}(\mathbf{x}^{(0)})\mathbf{c} \simeq \mathbf{y}$. Finally, we used $\tau = 10^{-3}$ in (1.6).

The "full" (4 parameter) and reduced (2 parameter) version of the problem needed 66 and 6 iteration steps, respectively. The two iteration processes are illustrated below.

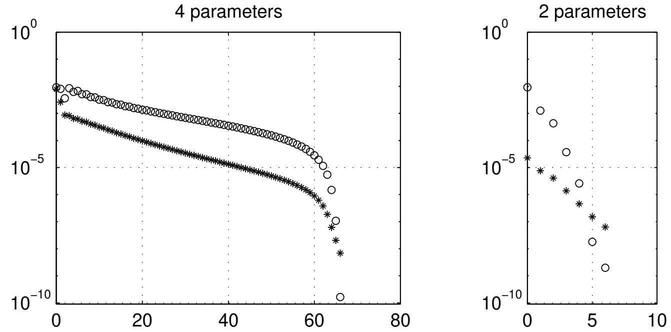

Figure 1b. *Behaviour of* $\|\phi'(\mathbf{z})\|_\infty$ (o) *and* μ (+)

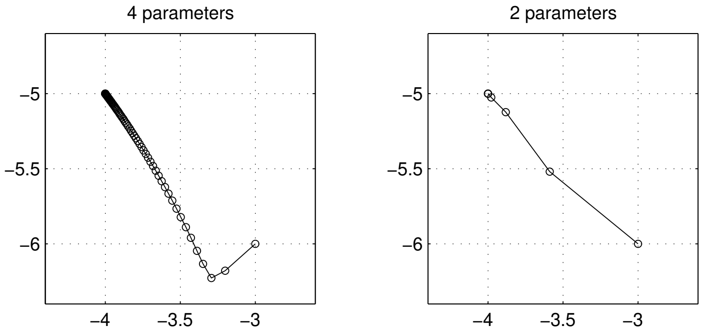

Figure 1c. *Behaviour of* $\mathbf{x}$

Data fitting with models of the form (4.1) have many applications and are known to be hard to solve, especially when the components of $\mathbf{x}$ are close. For the problem of Figure 1a the exploitation of separability has almost miraculous effect.

Next, consider the socalled *Meyer's problem*, illustrated below,

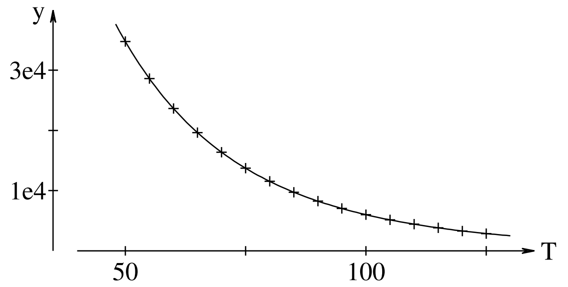

Figure 2a. *Meyer's problem*

The model is

$$M(\mathbf{x}, \mathbf{c}, t) = c_1 \exp\left(\frac{x_1}{x_2 + t}\right) \quad ,$$

i.e. a problem of the form 2° above. We introduce the vectors $\mathbf{a}$ and $\mathbf{b}$ with

$$a_i = x_2 + t_i, \quad b_i = x_1/a_i \ .$$

Then the quantities in (2.6) are given by

$$(\mathbf{F})_i = F_i = e^{b_i}, \quad (\mathbf{F}')_{i,:} = (F_i/a_i)[1,\ -b_i] \ .$$

The Meyer problem is widely used for testing nonlinear least squares software. The solution (to 6 digits accuracy) is

$$\mathbf{z}^* = [6181.35,\ 345.224,\ 0.00560964]^\top \ .$$

With the starting point

$$\mathbf{z}^{(0)} = [4000,\ 250,\ 0.0020]^\top], \quad \tau = 1 \ ,$$

and $\mathbf{z}^{(0)} = \mathbf{z}^{(0)}_{1:2}$ the Levenberg-Marquardt programme stopped after 181 (28) iteration steps with the 3 (2) parameter formulations of the problem. The behaviour of the nonlinear parameters is shown below. As in Figure 1c we see that the approach to $\mathbf{x}^*$ is much faster when we exploit separability.

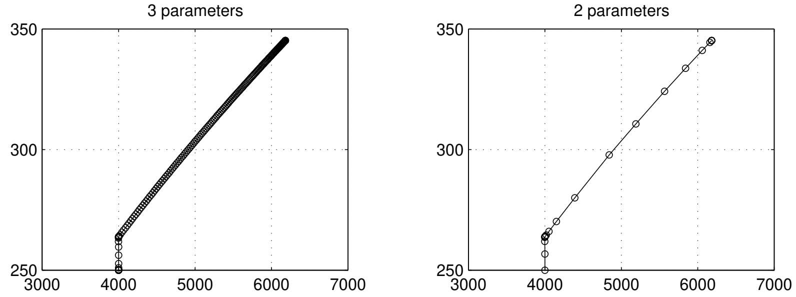

Figure 2b. *Behaviour of* $\mathbf{x}$

Finally, the data in Figure 3a are from a computer simulation of the flow past a submerged cylinder.[2)]

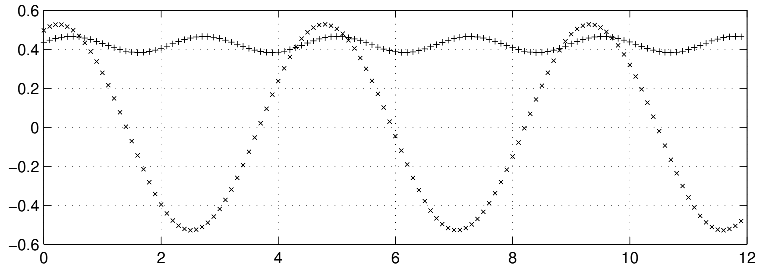

Figure 3a. *Streamwise* (+) *and crosswise* ($\mathbf{x}$) *fluctuations*

For each curve want to fit a model of the form (1.1) with $p = 3$, $n = 4$,

---

[2)] This is from an ongoing project by *Stefan Mayer, IMM, DTU.*

$$M(\widetilde{\mathbf{x}},\widetilde{\mathbf{c}},t) = \widetilde{c}_1+\widetilde{c}_2\cos(2\pi\widetilde{x}_1 t + \widetilde{x}_3) + \widetilde{c}_3\cos(2\pi\widetilde{x}_2 t + \widetilde{x}_4)\ . \tag{4.2}$$

Alternatively, we can formulate the model as

$$M(\mathbf{x},\mathbf{c},t) = c_1+c_2\cos(x_1 t) + c_3\sin(x_1 t) + c_4\cos(x_2 t) + c_5\sin(x_2 t)\ , \tag{4.3}$$

thus increasing the number of "linear parameters to $p=5$, while reducing the number of nonlinear parameters to $n=2$. There are simple relationships between $(\mathbf{x},\mathbf{c})$ and $(\widetilde{\mathbf{x}},\widetilde{\mathbf{c}})$. Note that (4.3) is preferable for a reformulation as in Section 2.

In terms of the parameters in (4.3) the least squares fits to the data in Figure 3a are

$$\text{Streamwise:}\quad \mathbf{x}^* = \begin{bmatrix}\texttt{2.77117}\\ \texttt{5.54244}\end{bmatrix}\quad \mathbf{c}^* = \begin{bmatrix}\texttt{4.25055e-1}\\ \texttt{1.05247e-2}\\ \texttt{4.00968e-2}\\ \texttt{5.36684e-4}\\ \texttt{4.02148e-4}\end{bmatrix}$$

$$\text{Crosswise:}\quad \mathbf{x}^* = \begin{bmatrix}\texttt{1.38560}\\ \texttt{4.15660}\end{bmatrix}\quad \mathbf{c}^* = \begin{bmatrix}\texttt{1.78620e-5}\\ \texttt{4.94180e-1}\\ \texttt{1.90192e-1}\\ \texttt{2.84519e-3}\\ \texttt{-3.43487e-3}\end{bmatrix}$$

With a good starting guess the formulation (4.3) and the corresponding reduced parameter formulation perform almost equally well. To investigate this further, we have made an experiment with the crosswise data, where the starting point was chosen as

$$\mathbf{z}^{(0)} = \operatorname{diag}(1+\delta\,\boldsymbol{\varrho})\mathbf{z}^*\ , \tag{4.4}$$

for varying δ. For each δ the vector $\boldsymbol{\varrho}\in\mathbb{R}^7$ was found by a random number generator[3)] followed by normalization so that $\|\boldsymbol{\varrho}\|_\infty = 1$. The solution was found by the Levenberg-Marquardt programme of footnote 1, with $\tau=10^{-3}$, the stopping criterion $\|\boldsymbol{\phi}'\|_\infty \le 10^{-8}$, and allowing at most 50 iteration steps. For the reduced formulation we used $\mathbf{x}^{(0)} = \mathbf{z}^{(0)}_{1:2}$, and for (4.2) the simple relations were used to find $\widetilde{\mathbf{z}}^{(0)}$ from $\mathbf{z}^{(0)}$.

Figure 3b illustrates the sensitivity on the starting guess. It is seen that generally the reduced formulation needs slightly fewer iteration steps than formulation (4.3) and that the $n=4$ formulation (4.2) is considerably slower.

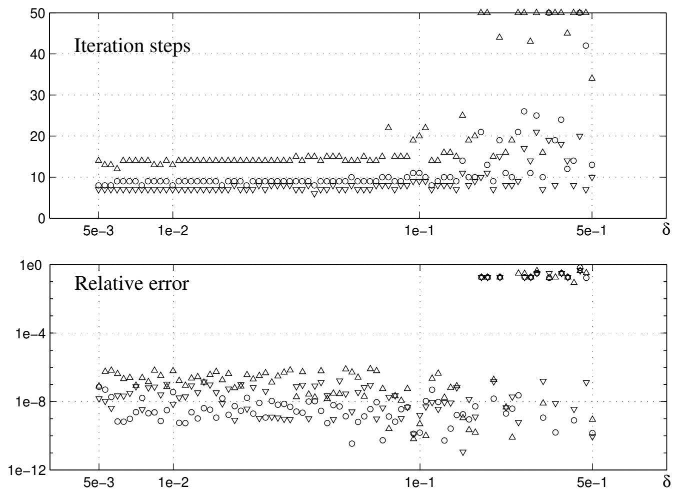

Figure 3b. *Starting guess given by (4.4).*
Formulation (4.2): $\triangle$, *(4.3):* $\circ$, *Reduced:* $\triangledown$

For large perturbations of the starting guess ($\delta \gtrsim 0.2$) the iteration may land in a local minimum of $\phi(\mathbf{z})$. The reduced formulation

3) We used MATLAB's `randn` initialized with `randn('seed', 12345678)` at the start of the experiment.

seems to be slightly more robust than both “fully nonlinear” formulations. More specifically, we can give the following statistics for the 81 problems in the experiments:

	(4.3)	(4.2)	Reduced
Wrong minimum	14	11	10
Average no. of it. steps in succesful runs	25.5	13.8	9.8

5. Conclusion

If a nonlinear least squares problem is separable, then this property should be exploited. The code describing the reduced problem will be only slightly more complicated, and often the nescessary number of iteration steps will decrease markedly, so that the total computational work decreases.

In the last of the above examples we saw that halving the number of nonlinear parameters from $n = 4$ in (4.2) to $n = 2$ in (4.3) halves the number of iteration steps needed. Although we do not suggest that the number of steps is directly proportional to n, it is to be expected that a nonlinear problem normally is easier to solve if a reformulation reduces n.

The conclusion is based on very few examples, and we have only tried one solution algorithm. However, we cannot see, why this should not be true in general, and this is confirmed by the papers in the list of references.

References

[1] L. Kaufman (1975): *A Variable Projection Method for Solving Separable Nonlinear Least Squares Problems.* BIT **15**, 49 – 57.

[2] L. Kaufman and D. Gay (1991): *Tradeoffs in Algorithms for Separable and Block Separable Nonlinear Least Squares.* IMACS 91 Proceedings of the 13th World Congress on Computational and Applied Mathematics, Ed. R. Vichnevetsky and J. J. H. Miller, pp. 157 – 158, Criterion Press, Dublin.

[3] L. Kaufman. G. Sylvester and M. Wright (1994): *Structured Linear Least-Squares Problems in System Identification and Separable Nonlinear Data Fitting.* SIAM J. OPT. **4** (4), 847 – 871.

[4] H.B. Nielsen (1999): *Damping Parameter in Marquardt's Method.* IMM, DTU. Report IMM-REP-1999-05. Available at http://www.imm.dtu.dk/~hbn/publ/TR9905.ps.Z

[5] A. Ruhe and P.Å. Wedin (1980): *Algorithms for Separable Nonlinear Least Squares Problems.* SIAM REVIEW **22** (3), 318 – 337.

[6] N. Sagara and M. Fukushima (1984): *A Continuation Method for Solving Separable Nonlinear Least Squares.* JOURN. COMP. AND APPL. MATH. **10** (2), 157 – 161.