RICARDO ALEXANDRE DECKMANN ZANARDINI

A SUCCESSIVE LEAST SQUARES METHOD FOR NONLINEAR LEAST SQUARES PROBLEMS

Dissertation presented as partial fulfilment for obtaining the degree of Master of Science. Post Graduate Program in Numerical Methods in Engineering. Sectors of Exact Science and Technology, Federal University of Paraná, Brazil.
Advisor: Prof. Dr. Jin Yun Yuan

CURITIBA
2001

RICARDO ALEXANDRE DECKMANN ZANARDINI

A SUCCESSIVE LEAST SQUARES METHOD FOR NONLINEAR LEAST SQUARES PROBLEMS

Dissertation presented as partial fulfilment for obtaining the degree of Master of Science. Post Graduate Program in Numerical Methods in Engineering. Sectors of Exact Science and Technology, Federal University of Paraná, Brazil.

Advisor: Prof. Dr. Jin Yun Yuan.

CURITIBA

2001

RICARDO ALEXANDRE DECKMANN ZANARDINI

A SUCCESSIVE LEAST SQUARES METHOD FOR NONLINEAR LEAST SQUARES PROBLEMS

Dissertação aprovada como requisito parcial para a obtenção do grau de Mestre em Ciências, M.Sc. - Área de concentração: Programação Matemática - do Programa de Pós Graduação Métodos Numéricos em Engenharia da Universidade Federal do Paraná pela comissão formada pelos professores:

Coordenador do Curso:

Prof. Dr. Maria Teresinha Arns Steiner.

Orientador:

Prof. Dr. Jin Yun Yuan.
Departamento de Matemática - UFPR

Prof. Dr. Marli Cardia
Departamento de Matemática - UFPR

Prof. Dr. Raimundo J. B. de Sampaio
Departamento de Matemática - PUCPR
Examinador externo.

Curitiba, 05 de dezembro de 2001.

PARANÁ - BRASIL

In the province of the mind,

what one believes to be true,

either is true or becomes true.

To Mônica.

To my family.

Acknowledgements

I am deeply grateful to my advisor, Prof. Dr. Jin Yun Yuan, for introducing me to this topic and for his helpful discussions and suggestions.

I would like to give my thanks to my professors Dr. Celso Carnieri, Dr. Jiahong Yin, Dr. Maria Terezinha Arns Steiner, Dr. Marli Cardia, Dr. Mildred Ballin Hecke and Dr. Rubens Robles Ortega Junior for their teachings and helps.

I would like to give my sincere thanks to my family, specially to my wife, Mônica, for their comprehension throughout my work.

I also would like my thanks to my friends Ms. Waléria Adriana Gonçalez Cecílio, Mr. Celso José Cordeiro and Mr. Cosmo Damião Santiago for their helpful discussions and collaboration during my study.

I would like to give my special thanks to CAPES for its financial support and to CESEC for its structure.

Abstract

Nonlinear least squares (NLS) problems appear in many important practical applications, for instance, signal processing and functional approximations. The majority of the methods for solving NLS problems is based on a minimization of subproblems, but they do not keep the error structure as the data structure. In this dissertation we propose a new method for solving NLS problems. We call this method Successive Nonlinear Least Squares (SNLS). Based on the idea developed recently by Yalamov and Yuan (2000), the SNLS method differs from the existing approaches because it preserves the structure of the error. The SNLS method consists of the solutions of successive least squares (LS) problems by the total least squares (TLS) formulation. The SNLS method is general NLS problem solver and is quite suitable for structured NLS problems. Some numerical tests for Toeplitz and Vandermonde matrices and parameter estimation problems are presented. Numerical results illustrate that the SNLS algorithm converges fast and provides good approximations to the exact solution of the NLS problems for our test problems.

Resumo

Problemas de mínimos quadrados não-lineares aparecem em muitas aplicações importantes. Por exemplo, processamento de sinal e aproximações funcionais. A maioria dos métodos destinados à resolução de problemas de mínimos quadrados não-lineares está baseada na minimização de subproblemas mas estes métodos não preservam a estrutura da matriz dos erros como a matriz dos dados. Nesta dissertação propomos um novo método destinado à resolução de problemas de mínimos quadrados não-lineares. Chamamos este método de método dos Mínimos Quadrados Não-lineares Sucessivos. Baseado na idéia recentemente desenvolvida por Yalamov e Yuan (2000), o método dos Mínimos Quadrados Não-lineares Sucessivos difere das abordagens existentes por que este método preserva a estrutura da matriz dos erros. O método dos Mínimos Quadrados Não-lineares Sucessivos consiste na solução de sucessivos problemas de mínimos quadrados através da formulação de mínimos quadrados totais. Este método resolve problemas gerais de mínimos quadrados não-lineares e é muito adequado à resolução de problemas de mínimos quadrados não-lineares estruturados. Alguns testes numéricos foram realizados para matrizes Toeplitz e Vandermonde e problemas de estimativa de parâmetros são apresentados. Testes numéricos mostram que o algoritmo converge

rapidamente e fornece boas aproximações para a solução exata dos nossos problemas testes.

List of Tables

List of Figures

Contents

Chapter 1

Introduction

The aim of this work is to propose a new iterative method for solving nonlinear least squares (NLS) problems

$$A(\alpha)x = b. \tag{1.1}$$

Here $A(\alpha)$ is a $m \times n$ nonlinear functional with $m \geq n$, $x, b \in \mathbf{R}^n$ and α is a $s \times 1$ parameter vector. Nonlinear least squares problems appear in many engineering applications such as signal processing [30], frequency and exponential decay estimation [1, 25, 31], engineering and optical design, optimal control, curve fitting, fitting scattered data in three-dimensional space [21], exponential data modelling problems, functional approximation and others problems where nonlinear systems of equations are involved.

In general, methods described in [5, 15, 16, 17, 18, 36] obtain the solution by

solving sequences of minimization subproblems do not keep error structure as the data structure. Rosen, Park and Glick [33] develop an algorithm called Structured Nonlinear Total Least Norm (SNTLN), an extension of the Structured Total Least Norm (STLN) [32], for solving structured nonlinear least squares problems.

In particular, the method for solving NLS problems is to minimize the residual $r(x) = Ax - b$ iteratively with the initial value of α till to achieve a desired approximation. The related approach can be done based on the following total least squares (TLS) formulation

$$\begin{aligned} &\min_x \|(E, r)\|_F \\ &\text{s.t. } (A + E)x = b + r \end{aligned} \tag{1.2}$$

which is also known as the errors-in-variables [37].

Recently the Successive Least Squares (SLS) method for obtaining an approximate solution to the overdetermined TLS problem has been proposed by Yalamov and Yuan [39]. This method is quite suitable for structured TLS problems such as Toeplitz and Hankel systems because it satisfies the error structure requirements.

The idea of methods for solving large structured TLS problems [5, 7, 8, 9, 10] motivates us to study a new method for solving large structured NLS problems by solutions of successive least squares (LS) problems. This is the main purpose of this dissertation.

1.1 Outline of the Work

This dissertation is organized as follows. In Chapter 1 some basic ideas about least square solution to linear systems and structured matrices are reviewed. In Section 1.3 we summarize some structured matrices such as Toeplitz, Hankel and Vandermonde. A special method to find the Toeplitz matrix-vector multiplication with the complexity $O(n \log_2 n)$ flops, the Fast Fourier Transform (FFT), is described in Section 1.4. The QR decomposition by Givens and Hyperbolic rotations are reminded in Sections 1.5 and 1.6, respectively.

Chapter 2 contains important issues about existing efficient methods for structured linear least squares problems, specifically, Toeplitz systems [2, 11, 12, 22]. The Fast Inverse QR Factorization described in Section 2.1 is suitable for Toeplitz systems because the consideration of the structure of the matrix A. This fact reduces the computational cost to $O(mn)$ operations to find its LS solution. The TLS formulation and its properties are treated in Section 2.3. A survey of the SLS method is given in Section 2.4.

In Chapter 3 the NLS problem, which is our contribution of this dissertation, is treated. This problem is closely related to unconstrained minimization problems in $\mathbf{R}^n$. In Section 3.2 a new method called Successive Nonlinear Least Squares (SNLS) method is proposed. This method is general solver for NLS problems and quite suitable for structured NLS problems because of the preservation of the

structure of the input data and error matrix. The SNLS method is based on ideas of the SLS method developed by Yalamov and Yuan [39]. Our work considers the case of Toeplitz, Hankel and Vandermonde matrices, and other structures can be treated similarly.

In Chapter 4 numerical tests considering overdetermined Vandermonde systems are given to verify the efficiency of the SNLS method. The results assert that the SNLS method obtains good approximated solution for our tests. In Section 4.1 we present an application of the Vandermonde systems to the exponential data modelling. In Section 4.2 we propose some modifications in the SNLS method, the MSNLS method, to improve the convergence and the accuracy of the SNLS algorithm for Vandermonde systems. In Section 4.3 the convergent behavior of α for both the SNLS and MSNLS methods are treated with different initial guess $\bar{\alpha}$. In Section 4.4 two different types of signal are considered. The first type of signal data is from that used by Osborne and Smyth [29] and the second type of function is the sum of Gaussian functions which is similar to that used in [22]. In Section 4.5 Toeplitz systems taken from [22] are considered.

In Chapter 5 we set some conclusions and suggestion for future research.

1.2 Least Square Solution to Linear Systems

A linear least squares (LS) problem consists of solving an overdetermined linear system

$$Ax = b. \tag{1.3}$$

Here $A \in \mathbf{R}^{m\times n}$, $m \geq n$, $b \in \mathbf{R}^m$, and $x \in \mathbf{R}^n$. Unless b belongs to the range of A, i.e., $b \in R(A)$, the overdetermined system has no classic solution. Then we can find Ax such that it is the best approximation in some sense to b and denote it by

$$Ax = b.$$

One common way is to find approximation x such that

$$\|Ax - b\|_2 = \min_y \|Ay - b\|_2. \tag{1.4}$$

Of course we can consider different norms in (1.4).

Since the function $f(x) = \frac{1}{2}\|Ax - b\|_2^2$ is differentiable, by KKT condition, we have the normal equations $A^TAx = A^Tb$. Since the 2-norm is preserved under the orthogonal transformation, QR factorization can be used to solve the problem efficiently.

The vector x is called a least square solution of (1.4) where Ax is as close as possible to the vector b.

The following theorem characterizes the property of the LS solution.

THEOREM 1.2.1 (LS SOLUTION PROPERTY)

$$x^* \text{ solves the LS problem (1.4)} \iff A^T(b - Ax^*) = 0.$$

Proof: One proof of this theorem is given in Björck [5], pp. 5. ♦

The vector b is decomposed into two orthogonal components:

$$b = b^* + r^* = Ax^* + r^*, \quad r^* \perp Ax^*, \quad Ax^* = b^*$$

where b^* is the orthogonal projection of b onto $R(A)$. Note that the decomposition is always unique, even when the LS solution x^* is not unique.

Moreover, from Theorem 1.2.1 note that the LS solution satisfies the normal equations

$$A^T Ax = A^T b. \tag{1.5}$$

The system (1.5) is always semidefinite positive and consistent since

$$A^T b \in R(A^T) = R(A^T A).$$

COROLLARY 1.2.2 (LS SOLUTION AND RESIDUAL) *If $rank(A) = n$ then (1.4) has a unique LS solution, given by*

$$x^* = (A^T A)^{-1} A^T b.$$

The corresponding LS correction is given by the residual:

$$r^* = b - Ax^* = b - b^*, \; b^* = P_A b$$

where $P_A = A(A^TA)^{-1}A^T$ *is the orthogonal projector onto* $R(A)$.

If $rank(A) = r < n$, the LS problem (1.4) is rank-deficient and has an infinite number of solutions. For example, if x is a minimizer and $z \in N(A)$ then $x + z$ is also a minimizer. For reasons of stability and minimal sensitivity, a unique solution having minimal 2-norm is singled out from the set of all minimizer

$$\mathcal{X} = \{x \in \mathbf{R}^n : \|Ax - b\|_2 = min\}.$$

We denote the solution by x^*. Note that in the full rank case, there is only one LS solution which has minimal 2-norm.

1.3 Structured Matrices

In many real applications, the input data of the problems has special structure such as Toeplitz, Hankel or Vandermonde. Many fast solvers take into account this structure where these special matrices appears.

1.3.1 Toeplitz Matrices

DEFINITION 1.3.1 *The matrix* $T \in \mathbf{R}^{n+1\times n+1}$ *is called Toeplitz if the matrix* T *is in the form*

$$T = \begin{pmatrix} t_0 & t_1 & t_2 & \cdots & t_n \\ t_{-1} & t_0 & t_1 & \cdots & t_{n-1} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ t_{-n} & t_{-(n-1)} & t_{-1} & \cdots & t_0 \end{pmatrix}.$$

In this case, the general term is $t_{ij} = t_{j-i}$ for some given sequence with the entries $t_{-n}, t_{-n+1}, ..., t_{-1}, t_0, t_1, ..., t_n$. A Toeplitz matrix is defined by one row and one column. Sometimes we denote the Toeplitz matrix T as a vector in the form $(t_{-n}, t_{-n+1}, ..., t_{-1}, t_0, t_1, ..., t_n)^T$.

We denote the symmetric Toeplitz matrix defined by just one row vector $(t_0, t_1, ..., t_n)^T$ as well.

1.3.2 Circulant Matrices

A special case of a Toeplitz matrices is presented here.

DEFINITION 1.3.2 *We have that an $n \times n$ matrix C_n is called circulant when*

$$C_n = \begin{pmatrix} c_0 & c_{-1} & \cdots & c_{2-n} & c_{1-n} \\ c_1 & c_0 & c_{-1} & \cdots & c_{2-n} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ c_{n-2} & \ddots & \ddots & \ddots & \vdots \\ c_{n-1} & c_{n-2} & \cdots & c_1 & c_0 \end{pmatrix}.$$

where $c_{-k} = c_{n-k}$ for $1 \leq k \leq n-1$.

1.3.3 Hankel Matrices

DEFINITION 1.3.3 *The matrix $H \in \mathbf{R}^{m \times n}$ is called Hankel if H is in the form*

$$H = \begin{pmatrix} h_0 & h_1 & h_2 & \cdots & h_n \\ h_1 & h_2 & h_3 & \cdots & h_{n+1} \\ h_2 & h_3 & h_4 & \cdots & h_{n+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h_n & h_{n+1} & h_{n+2} & \cdots & h_{2n} \end{pmatrix}.$$

The general term of Hankel matrix H is $h_{ij} = h_{j+i-2}$ for some given sequence $h_0, h_1, ..., h_{2n-1}, h_{2n}$. The entries of H are constant along the anti-diagonals.

1.3.4 Vandermonde Matrices

DEFINITION 1.3.4 *A matrix of the form*

$$V = V(\alpha_0, \alpha_1, \ldots, \alpha_n) = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha_0 & \alpha_1 & \cdots & \alpha_n \\ \vdots & \vdots & & \vdots \\ \alpha_0^n & \alpha_1^n & \cdots & \alpha_n^n \end{pmatrix}$$

is Vandermonde where $\{\alpha_k\}_{k=0}^n$ *is a sequence of* $n+1$ *distinct real numbers.*

Vandermonde matrices are related to the polynomial problem of finding a polynomial $p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0$, which interpolates the data (α_i, f_i), $i = 0, 1, \ldots, n$. The coefficient vector satisfies the linear system

$$V^T a = f$$

called a dual Vandermonde system. The primal system

$$Vx = b$$

arises when determining weights x_i in quadrature formulas when moments b_i are given.

In General, Vandermonde systems are extremely ill-conditioned.

1.4 Fast Fourier Transform

The Fast Fourier Transform (FFT) is an efficient alternative to obtain a Toeplitz matrix-vector multiplication. This method consists of computing the finite Fourier

transform, with the significative reduction to the computational cost in order of $O(n \log_2 n)$ operations where n is the number of points.

The discrete Fourier coefficients $\{c_j\}_{j=0}^{n-1}$ for a function

$$f(x) = \sum_{j=0}^{n-1} c_j e^{ijx}$$

at the points $x_k = 2k\pi/n$, $k = 0, 1, \ldots, n-1$ can be computed by

$$c_j = \frac{1}{n} \sum_{k=0}^{n-1} f\left(\frac{2k\pi}{n}\right) e^{-ij2k\pi/n}, \quad j = 0, 1, \ldots, n-1. \tag{1.6}$$

If $\omega = e^{-2k\pi i/n}$ is the nth root of unity, i.e., $\omega^n = 1$, then the Fourier coefficients can be rewritten as follows

$$c_j = \frac{1}{n} \sum_{k=0}^{n-1} f_k \omega^{jk}, \quad f_k = \frac{1}{n} f(x_k).$$

Note that c_j is expressed as a polynomial of degree $n-1$ in ω^3. This can also be written as a matrix-vector multiplication

$$c = F_n f, \quad (F_n)_{jk} = \omega^{jk},$$

where $F_n \in \mathbf{R}^{n \times n}$ is the Fourier matrix.

In the usual implementation of the discrete Fourier transform, n^2 operations are required. Now, if $n = 2^r$ and set

$$k = \begin{cases} 2k_1, & \text{if } k \text{ even}, \\ 2k_1 + 1 & \text{if } k \text{ odd}, \end{cases} \qquad 0 \le k_1 \le \frac{1}{2}n - 1,$$

then the sum in (1.6) can be splitted into an even and an odd part such that

$$c_j = \sum_{k_1=0}^{\frac{1}{2}n-1} f_{2k_1}(\omega^2)^{jk_1} + \sum_{k_1=0}^{\frac{1}{2}n-1} f_{2k_1+1}(\omega^2)^{jk_1}\omega^j. \tag{1.7}$$

Let η be the quotient and j_1 the remainder when j is divided by $\frac{1}{2}n$, i.e., $j = \eta\frac{1}{2}n + j_1$. Then, since $\omega^n = 1$,

$$(\omega^2)^{jk_1} = (\omega^2)^{\eta(1/2)nk_1}(\omega^2)^{j_1k_1} = (\omega^n)^{\eta k_1}(\omega^2)^{j_1k_1} = (\omega^2)^{j_1k_1}.$$

Thus if, for $j_1 = 0, 1, \ldots, \frac{1}{2}n - 1$, we set

$$\phi(j_1) = \sum_{k_1=0}^{\frac{1}{2}n-1} f_{2k_1}(\omega^2)^{jk_1}, \quad \psi(j_1) = \sum_{k_1=0}^{\frac{1}{2}n-1} f_{2k_1+1}(\omega^2)^{jk_1},$$

where $(\omega^2)^{\frac{1}{2}n} = 1$. Then, by (1.7)

$$c_j = \phi(j_1) + \psi(j_i)\omega^j, \quad j = 0, 1, \ldots, n-1.$$

The computational effort by this way is $n \log_2 n$ operations.

Many excellent surveys of the use of the discrete Fourier transform are given by Cooley, Lewis and Welsh [13], Henrici [14] and Walker [38].

1.5 Givens Rotations

The Givens rotations introduces zeros more selectively than the Householder transformation. The matrix of Givens rotation is

$$G(i,k,\theta) = \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix}$$

where $G_{ii} = G_{kk} = c = \cos\theta$, $G_{ik} = s = \sin\theta$ and $G_{ki} = -s$ for some θ. Note that Givens rotations are orthogonal ($c^2 + s^2 = 1$ and $G(i,k,\theta)G(i,k,\theta)^T = I$). Geometrically, the matrix $G(i,k,\theta)$ rotates in θ radians a pair of coordinates axes in the (i,k) plane. For instance, if $x \in \mathbf{R}^n$ and $y = G(i,k,\theta)^T x$, then

$$y_i = \begin{cases} cx_i - sx_k & j = i, \\ sx_i + cx_k & j = k, \\ x_j & j \neq i,k. \end{cases}$$

Setting

$$c = \frac{x_i}{\sqrt{x_i^2 + x_k^2}} \quad \text{and} \quad s = \frac{-x_k}{\sqrt{x_i^2 + x_k^2}}$$

we can force y_k to be zero.

The following Givens rotation algorithm given in [20] requires 5 flops and a single square root.

ALGORITHM 1.5.1 (GIVENS ROTATIONS ALGORITHM) *Given the scalars a and b, this function computes $c = \cos(\theta)$ and $s = \sin(\theta)$ so that*

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix}^T \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r \\ o \end{pmatrix}$$

function $[c, s] = \mathbf{givens}(a, b)$

if $b = 0$

$c = 1$

$s = 1$

else

if $|b| > |a|$

$\tau = -a/b$

$s = 1/\sqrt{1 + \tau^2}$

$c = s\tau$

else

$\tau = -b/a$

$c = 1/\sqrt{1 + \tau^2}$

$s = c\tau$

end

end

end givens

1.6 Hyperbolic Rotations

In the same way of Givens rotation we can construct a hyperbolic transformation as follows

$$H = \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & -s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix}.$$

Here, $c = \cosh(\theta)$ and $s = \sinh(\theta)$. Moreover we have that $c^2 - s^2 = 1$. The hyperbolic rotations algorithm can be easily obtained.

Chapter 2

Iterative Methods for Structured LS Problems

In this chapter we remind some iterative algorithms for solving structured LS problems

$$\min_x \|Sx - b\|_2. \tag{2.1}$$

Here $S \in \mathbf{R}^{m\times n}$ is structured, $m \geq n$, $b \in \mathbf{R}^m$, and $x \in \mathbf{R}^n$.

In Section 2.1 Fast Inverse QR Factorization for solving Toeplitz systems developed by Nagy [27] is reviewed. This method makes use of the Toeplitz structure and requires only $O(mn)$ operations. Effective preconditioned conjugate gradients with circulant and noncirculant preconditioners are presented in Section 2.2. The TLS problem is given in Section 2.3. In Section 2.4 the Successive Least Squares

(SLS) method proposed by Yalamov and Yuan [39] is reviewed because it is an important algorithm for our work in next chapter. The SLS method preserves the structure of the error matrix as in the data matrix and is quite suitable for structured linear systems.

2.1 Fast Inverse QR Factorization

A fast solver of Toeplitz system

$$\min_x \|b - Tx\|_2 \tag{2.2}$$

is the Fast Inverse QR Factorization which requires only $O(mn)$ operations.

This algorithm is based on the Toeplitz structure of T. To compute the matrix R we have the following procedure:

If T is an $m \times n$ Toeplitz matrix of full column rank and R is the Cholesky factor of T^TT then T can be partitioned as

$$T = \begin{pmatrix} t_0 & u^T \\ v & T_0 \end{pmatrix} = \begin{pmatrix} T_0 & \tilde{u} \\ \tilde{v}^T & t_{m-n} \end{pmatrix} \tag{2.3}$$

where T_0 is a submatrix of T, u and $\tilde{v}$ are $n-1$ dimensional vectors, v and $\tilde{u}$ are $m-1$ dimensional vectors, and t_0 and t_{m-n} are scalars. In the same way R can be partitioned as

$$R = \begin{pmatrix} r_{11} & z^T \\ 0 & R_b \end{pmatrix} = \begin{pmatrix} R_t & \tilde{z} \\ 0^T & r_{nn} \end{pmatrix}. \tag{2.4}$$

Here z and $\tilde{z}$ are $n-1$ dimensional vectors and r_{11} and r_{nn} are scalars.

Using the partitioning (2.3) and (2.4) and then setting $R^TR = T^TT$ we have

$$\begin{pmatrix} r_{11}^2 & r_{11}z^T \\ r_{11}z & zz^T + R_b^TR_b \end{pmatrix} = \begin{pmatrix} t_0^2 + v^Tv & t_0u^T + v^TT_0 \\ t_0u + T_0^Tv & uu^T + T_0^TT_0 \end{pmatrix} \tag{2.5}$$

and

$$\begin{pmatrix} R_t^TR_t & R_t^T\tilde{z} \\ \tilde{z}^TR_t & \tilde{z}^T\tilde{z} + r_{nn}^T \end{pmatrix} = \begin{pmatrix} T_0^TT_0 + \tilde{v}\tilde{v}^T & T_0^T\tilde{u} + t_{m-n}\tilde{v} \\ \tilde{u}^TT_0 + t_{m-n}\tilde{v}^T & \tilde{u}^T\tilde{u} + t_{m-n}^2 \end{pmatrix}. \tag{2.6}$$

From (2.5) and (2.6) we have

$$zz^T + R_b^TR_b = uu^T + T_0^TT_0$$

and

$$R_t^TR_t = T_0^TT_0 + \tilde{v}\tilde{v}^T.$$

It follows that

$$R_b^TR_b = R_t^TR_t + uu^T - \tilde{v}\tilde{v}^T - zz^T. \tag{2.7}$$

From (2.5) the first row of R can be easily computed because

$$r_{11}^2 = t_0^2 \text{ and } z^T = (t_0u^T + v^TT_0)/r_{11}.$$

The expression (2.7) can be rewritten as

$$R_1^TR_1 = R_t^TR_t + uu^T \tag{2.8}$$

$$R_2^TR_2 = R_1^TR_1 - \tilde{v}\tilde{v}^T \tag{2.9}$$

and

$$R_b^TR_b = R_2^TR_2 - zz^T \tag{2.10}$$

where R_1 and R_2 are upper triangular matrices.

For the updating problem (2.8) we can find a product of Givens rotations $Q = G(n-1, n, \theta_{n-1}) \cdots G(1, n, \theta_1)$ so that

$$Q \begin{pmatrix} R_t \\ u^T \end{pmatrix} = \begin{pmatrix} R_1 \\ 0^T \end{pmatrix} \tag{2.11}$$

where $G(i, j, \theta_i)$ is a Givens rotation.

For the downdating problems (2.9) and (2.10), the Hyperbolic rotation $Y^{(1)} = H(n-1, n, \phi_{n-1}) \cdots H(1, n, \phi_1)$ and $Y^{(2)} = H(n-1, n, \rho_{n-1}) \cdots H(1, n, \rho_1)$ can be found such that

$$Y^{(1)} \begin{pmatrix} R_1 \\ \tilde{v}^T \end{pmatrix} = \begin{pmatrix} R_2 \\ 0^T \end{pmatrix} \tag{2.12}$$

and

$$Y^{(2)} \begin{pmatrix} R_2 \\ \tilde{z}^T \end{pmatrix} = \begin{pmatrix} R_b \\ 0^T \end{pmatrix}. \tag{2.13}$$

The following algorithm proposed by Nagy in [27] computes R with $mn+6n^2+O(n)$ multiplications.

ALGORITHM 2.1.1 (FAST INVERSE QR ALGORITHM) *Given the matrix T, this function computes R.*

function $R =$ **fastiqr**(T)

$R(1,1) = r_{11} = \sqrt{t_0^2 + t_1^2 + \cdots + t_{m-1}^2}$

$z = (t_0 u + T_0^T v)/r_{11}$

$R(1, 2:n) = z^T$

for $k = 1, 2, \ldots, n-1$

$r_t(k:n-1) = R(k, k:n-1)$

$[c, s] = \mathbf{givens}(r_t(k), u(k))$

$$\begin{pmatrix} r_1^T \\ u^T \end{pmatrix} = \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} r_t^T \\ u^T \end{pmatrix}$$

$[c, s] = \mathbf{hyp}(r_1(k), \tilde{v}(k))$

$$\begin{pmatrix} r_2^T \\ \tilde{v}^T \end{pmatrix} = \begin{pmatrix} c & -s \\ -s & c \end{pmatrix} \begin{pmatrix} r_1^T \\ \tilde{v}^T \end{pmatrix}$$

$[c, s] = \mathbf{hyp}(r_2(k), z(k))$

$$\begin{pmatrix} r_b^T \\ z^T \end{pmatrix} = \begin{pmatrix} c & -s \\ -s & c \end{pmatrix} \begin{pmatrix} r_2^T \\ z^T \end{pmatrix}$$

$R(k+1, k+1:n) = r_b(k:n-1)$

end

end fastiqr

2.2 CG Methods for Toeplitz Systems

Fast direct Toeplitz solvers require $O(n \log_2^2 n)$ operations. By using a preconditioned CG method, the computational effort is reduced to $O(n \log_2 n)$ operations. Fast direct Toeplitz solvers are unstable for some important Toeplitz matrices such as indefinite and certain non-Hermitian Toeplitz matrices. In this case preconditioned iterative methods must be used.

2.2.1 Toeplitz and Circulant Matrices Using Fourier Series

An $n \times n$ Toeplitz matrix T_n with the entries $t_{j,k} = t_{j-k}$, $0 \leq j,\ k < n$ can be defined for all $n \geq 1$ by the following Fourier coefficients of f

$$t_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(\theta) e^{-ik\theta} d\theta,\ k = 0, \pm 1, \pm 2, ... \tag{2.14}$$

where f is a 2π-periodic continuous real-valued function defined on $[-\pi, \pi]$. The function f is called the generating function of the sequence of Toeplitz matrices T_n. Note that when f is an even function T_n is a real symmetric matrix.

If C_n is a circulant matrix then it can be diagonalized by the Fourier matrix F_n such that

$$C_n = F_n^* \Lambda_n F_n \tag{2.15}$$

where the entries of F_n are given by

$$[F_n]_{j,k} = \frac{1}{\sqrt{n}} e^{2\pi ijk/n},\ j, k \leq n-1 \tag{2.16}$$

and Λ_n is a diagonal matrix with the entries as the eigenvalues of C_n. We can obtain the diagonal entries λ_k of Λ_k by the FFT of first column of C_n

$$\lambda_k = \sum_{j=0}^{n-1} c_j e^{2\pi ijk/n},\ k = 0, ..., n-1. \tag{2.17}$$

By FFT we can obtain the products $C_n y$ and $C_n^{-1} y$ for every vector y using (2.15) in $O(n \log_2 n)$ operations.

2.2.2 CG Method for Toeplitz Matrices

We shall apply the CG method for solving the Toeplitz system. The method just needs matrix-vector product.

At each iteration $T_n y$ can be computed by FFT embedding T_n into a $2n \times 2n$ circulant matrix, i.e.,

$$\begin{pmatrix} T_n & \dagger \\ \dagger & T_n \end{pmatrix} \begin{pmatrix} y \\ 0 \end{pmatrix} = \begin{pmatrix} T_n y \\ \dagger \end{pmatrix}. \tag{2.18}$$

The matrix-vector multiplication requires $O(2n \log_2(2n))$ operations and the total number of operations at each iteration is $O(n \log_2 n)$ operations. An extra $2n$-vector for storing the eigenvalues of the embedded circulant matrix given in (2.18) is need.

Then, to solve the Toeplitz system $T_n x = b$ we solve the following preconditioned system

$$P_n^{-1} T_n x = P_n^{-1} b. \tag{2.19}$$

The preconditioner P_n is a matrix constructed with $O(n \log_2 n)$ operations and the linear system $P_n v = y$ is solved in $O(n \log_2 n)$ operations.

2.2.3 Circulant Toeplitz Preconditioners

If C is an $n \times n$ circulant matrix then it can be diagonalized by (2.15) and its inversion is done in $O(n \log_2 n)$ operations by FFT of size n.

Strang [35] proposes the first circulant preconditioner $S = s_{k-l}$, $0 \leq k, l < n$ with the diagonais s_j given by

$$s_j = \begin{cases} c_j & 0 < j \leq \frac{n}{2}, \\ c_{j-n} & \frac{n}{2} < j \leq n, \\ s_{n+j} & 0 < -j < n. \end{cases} \tag{2.20}$$

This preconditioner is obtained from the central diagonals of C and reflected them around to complete the circulant requirements.

2.3 Total Least Squares Problem

Total least squares (TLS) problems appear in many engineering applications such as signal and image processing, systems identification, automatic control, deconvolution techniques, and systems response prediction [37]. The TLS problem is a general case of the LS problem based on the fact that in many cases sampling or modelling errors affect both the vector b and the matrix A.

This assumption motivates an estimative of the unknown vector x given by solving the following TLS problem

$$\begin{aligned} &\min_x \|(E, r)\|_F \\ &\text{s.t. } (A + E)x = b + r \end{aligned} \tag{2.21}$$

where $A, E \in \mathbf{R}^{m\times n}$, $m \geq n$, $b, r \in \mathbf{R}^m$, and $x \in \mathbf{R}^n$, $\|\cdot\|_F$ denotes the Frobenius norm and $b+r \in R(A+E)$. The Frobenius norm of an $m \times n$ matrix A is defined by

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2} = \sqrt{\mathrm{tr}(A^T A)}$$

where "tr" denotes the trace.

The vector x^* that solve this problem is called total least squares (TLS) solution.

In many cases the solution of the TLS problem is close to the solution of the correspondent LS problem.

2.4 Successive Least Squares Method

In this section we review the Successive Least Squares (SLS) method proposed by Yalamov and Yuan [39] for solving structured TLS problems. This method is based on solution of successive LS problems to satisfy the requirement of error matrix structure. Of course, the method can also be applied to the general TLS problems.

2.4.1 Mathematical Background

The TLS equation

$$(A+E)x = b+r \tag{2.22}$$

is nonlinear whith respect to the unknowns E, x and r. The unknowns can be separated into two groups. For instance, we have one group for x and another one for E and r. If x is constant the problem becomes linear with respect to the unknowns E and r. In many cases the LS solution is close enough to the TLS solution because LS problems are a special case of TLS problems. Thus the initial value x_0 is chosen for solving the LS problem $Ax_0 = b + r_0$. This initial value is also proposed in [6, 32].

Considering x as a constant at each step, and E and r variables, we rewrite the TLS problem (2.21) as

$$\begin{aligned} &\min \left\| \begin{array}{c} r \\ \alpha \end{array} \right\|_2 \\ &\text{s.t. } Ax + X\alpha = b + r. \end{aligned} \tag{2.23}$$

To obtain the matrix X and the vector α we consider the following equality

$$X\alpha = Ex. \tag{2.24}$$

This choice depends on the structure of the matrix E.

To illustrate this choice a few examples are cited here.

EXAMPLE 2.4.1 *If E is an unstructured matrix then we have*

$$X = \begin{pmatrix} x_1 \cdots x_n & & & \\ & x_1 \cdots x_n & & \\ & & \ddots & \\ & & & x_1 \cdots x_n \end{pmatrix}$$

and

$$\alpha = vec(E) = (e_{11}, \ldots, e_{1n}, e_{21}, \ldots, e_{2n}, \cdots, e_{m1}, \ldots, e_{mn})^T.$$

Here $X \in \mathbf{R}^{m \times mn}$ *and* $\alpha \in \mathbf{R}^{mn}$.

EXAMPLE 2.4.2 *If* E *is sparse,* X *and* α *are also sparse. Their sparsity pattern depends on the sparsity pattern of* E.

EXAMPLE 2.4.3 *General Toeplitz matrices* E *generates the following results*

$$X = \begin{pmatrix} x_n & x_{n-1} & \cdots & x_1 & & 0 \\ & x_n & \cdots & x_2 & x_1 & \\ & & \ddots & & \ddots & \ddots \\ 0 & & & x_n & \cdots & x_2 & x_1 \end{pmatrix}$$

and

$$\alpha = (e_{n-1}, \ldots, e_1, e_0, e_{-1}, \ldots, e_{-m+1})^T$$

where $X \in \mathbf{R}^{m \times (m+n-1)}$ *and* $\alpha \in \mathbf{R}^{m+n-1}$.

For simplicity, the problem (2.23) can be rewritten as

$$\min \left\| \begin{array}{c} r \\ \alpha \end{array} \right\|_2$$

$$\text{s.t. } (-I, X) \begin{pmatrix} r \\ \alpha \end{pmatrix} = b - Ax. \tag{2.25}$$

Here (2.25) is an underdetermined linear system with the identity matrix $I \in \mathbf{R}^{m\times m}$ and $X \in \mathbf{R}^{m\times k}$ where k depends on the structure of E.

By analysis of Björck [6], the solution of (2.25) is given by $r = -y$ and $\alpha = X^T y$ where y is the solution of the system

$$(I + XX^T)y = s. \tag{2.26}$$

This form is a better alternative to solve the LS problem (2.25) because the matrix $M = I + XX^T$ is computed easily in practical applications. Moreover, M is symmetric positive definite (s.p.d.) and its smallest eigenvalue is not less than 1.

To obtain the solution of the linear system (2.26) there are many alternatives such as iterative methods, direct solution by some banded solver or direct solution by a super-fast Toeplitz solver [2]. Since M has very nice properties the iterative solution using CG methods with an appropriate preconditioner (Section 2.2) is quite suitable to solve this system.

2.4.2 Successive Least Squares Algorithm

The SLS method is based on the solution of successive LS problems. The initial value x_0 is given solving the related LS problem $Ax_0 = b + r_0$. When x is a known vector we compute E and r. Obtaining E, x and r can be computed by solving a LS problem with the matrix $A+E$ and the vector b. The SLS algorithm is proposed

based on these notes.

ALGORITHM 2.4.1 (SUCCESSIVE LEAST SQUARES ALGORITHM) *Given A and b this algorithm computes the TLS solution x.*

function x =**sls**(A, b)

Solve the LS problem $Ax_0 = b$

$r_0 = -s_0 = Ax_0 - b$

for $k = 1, 2, \ldots$ *until convergence*

Solve the LS problem $(-I, X_k) \begin{pmatrix} r_k \\ \\ \alpha_k \end{pmatrix} = s_{k-1}$

Define E_k *such that* $X_{k-1}\alpha_k = E_k x_{k-1}$

Solve the LS problem $(A + E_k)x_k = b$

$s_k = b - Ax_k$

$r_k = E_k x_k - s_k$

end

end sls

For Toeplitz systems the matrix M is fully defined by its first row $m^{(1)}$ whose entries are defined by the following matrix-vector product

$$\begin{pmatrix} x_n & \cdots & x_1 \\ & \ddots & \vdots \\ 0 & & x_n \end{pmatrix} \begin{pmatrix} x_n \\ \vdots \\ x_1 \end{pmatrix}. \tag{2.27}$$

Note that this matrix is also Toeplitz. So, this multiplication can be done in $O(n \log_2 n)$ flops by FFT. When an iterative method is used to solve the s.p.d. Toeplitz system $My = s$ the matrix M is not definite explicitly.

The matrix-vector multiplication $\alpha = X^T y$ is done in $O((m+n)\log_2(m+n))$ flops.

Unfortunately, for general matrices E the overdetermined linear system

$$(A + E_k)x_k = b \tag{2.28}$$

can not be solved efficiently. This occur because E_k, in general, is not of low rank. The Fast Inverse QR Iteration (Section 2.1) may be used to solve the system (2.28) in $4mn + O(n^2)$ flops.

Finally, the computation of s and r that involves products with Toeplitz matrices are done in $O((m+n)\log_2(m+n))$ flops.

Summarizing, the total number of flops per iteration step is

$$4mn + O((m+n)\log_2(m+n))$$

and a few vectors of length not greater than $m+n$ are stored.

By numerical experiments given in Yalamov and Yuan [39], the method can obtain the TLS solution which is quite different from the LS solution. Other methods do not keep the structure of the error matrix in this case.

Chapter 3

The SNLS Method for NLS Problems

In this chapter we propose a new way for solving overdetermined NLS problems

$$A(\alpha)x = b. \tag{3.1}$$

Here $A(\alpha)$ is an $m \times n$ nonlinear functional of an $s \times 1$ parameter vector α with $m \geq n$, $x \in \mathbf{R}^n$ and $b \in \mathbf{R}^m$.

This way is based on the TLS formulation (2.21) combined with the successive LS method. In Section 3.1 the NLS problem is presented. An important source of the NLS problems is the fitting data to a mathematical model such as exponential fitting problems. In Section 3.2 we derive the SNLS method and its numerical algorithm. The SNLS method is quite suitable for structured NLS problems because

the SNLS algorithm preserves the structure of the error matrix. Here we just consider the case of Toeplitz, Hankel and Vandermonde structures. Since the SNLS method is designed for general structure problems, other structures can be treated similarly. Numerical experiments will be presented in next chapter.

3.1 Nonlinear LS Problems

The unconstrained Nonlinear LS problem consists of finding a global minimizer of the sum of squares of m nonlinear functions

$$\min_{x \in \mathbf{R}^n} \phi(x), \quad \phi(x) = \frac{1}{2}\|f(x)\|_2^2 = \frac{1}{2} f(x)^T f(x) \tag{3.2}$$

where $f : \mathbf{R}^n \to \mathbf{R}^m$, $m \geq n$. This is a special case of the general optimization problem in $\mathbf{R}^n$ that is solved by unconstrained methods such as described in [5, 4, 16, 18, 34].

An important application of the NLS problems is the fitting data to a mathematical model. Given a set of observed points (t_i, y_i), $i = 1, \ldots, m$, the purpose is to fit this points to a model function $y = g(t, x)$ If $r_i(x)$ represents the error in the prediction model for the ith observation,

$$r_i(x) = y_i - g(t_i, x), \quad i = 1, \ldots, m,$$

the objective is to minimize the l-norm of the vector $r(x)$. This problem is on the form of the problem (3.2) with $f(x) = r(x)$.

For example, the following exponential fitting problem is to determine the parameter vector x which gives the best fit to m observed points (t_i, y_i), $i = 1, \ldots, m$, $m \geq 5$, in the expression

$$y(t, x) = x_1 + x_2 e^{x_4 t} + x_3 e^{x_5 t}$$

This problem is linear with respect to x_1, x_2 and x_3 but nonlinear in x_4 and x_5. In this case, the parameter vector x can be partitioned as $x^T = (y^T, z^T)$ and is called the separable NLS problem.

3.2 Successive LS method for Nonlinear LS Problems

The idea of the SLS algorithm proposed by Yalamov and Yuan [39] motivates us to set up some similar method for solving structured NLS problems because of preservation of error structure.

3.2.1 Successive Nonlinear Least Squares Method

From (3.1) the residual vector r is defined by

$$r(\alpha, x) = A(\alpha)x - b.$$

In many applications $A(\alpha)$ has some special structure, for example, Toeplitz, Hankel or Vandermonde. The related problem is called structured NLS problem.

The main problem is the error in $A(\alpha)$ also with the same structure as that of $A(\alpha)$. Note that the problem (3.1) is linear in x and nonlinear in α and called separable NLS problem. In this work we limit our attention to the separable case investigated in the papers [3, 19, 23, 26, 28].

We follow ideas from Rosen, Park and Glick [33] and Yalamov and Yuan [39] to consider the equivalent form for the problem as follows

$$\begin{aligned} &\min_x \|(\Delta A(\alpha), r)\|_F \\ &\text{s.t. } (A(\alpha) + \Delta A(\alpha))x = b + r. \end{aligned} \tag{3.3}$$

The matrix $A(\alpha)$ contains errors that are stored in the matrix $\Delta A(\alpha)$. The error matrix $\Delta A(\alpha)$ has the same structure as $A(\alpha)$ and $\Delta\alpha$ represents the corresponding error in α.

For the nonlinear formulation, in the equality (2.24) X is replaced by $J(\alpha, x)$. Then we have

$$\Delta A(\alpha)x = J(\alpha, x)\Delta\alpha. \tag{3.4}$$

Here, $J(\alpha, x)$ is the Jacobian of $A(\alpha)x$ with respect to α. Let $a_j(\alpha)$ represent the jth column of $A(\alpha)$,

$$J(\alpha, x) = \nabla_\alpha(A(\alpha)x) = \sum_{j=1}^{n} x_j \nabla_\alpha a_j(\alpha).$$

Therefore, the problem (3.3) can be stated as follows

$$\begin{aligned} &\min_x \left\| \begin{array}{c} r \\ \Delta\alpha \end{array} \right\|_2 \\ &\text{s.t. } A(\alpha)x + J(\alpha, x)\Delta\alpha = b + r, \end{aligned} \tag{3.5}$$

that is,

$$\begin{aligned} &\min_x \left\| \begin{array}{c} r \\ \Delta\alpha \end{array} \right\|_2 \\ &\text{s.t. } [-I, J(\alpha, x)] \begin{pmatrix} r \\ \Delta\alpha \end{pmatrix} = b - A(\alpha)x. \end{aligned} \tag{3.6}$$

Here (3.6) is a underdetermined linear system with $I \in \mathbf{R}^{m\times m}$ and $J(\alpha, x) \in \mathbf{R}^{m\times s}$ where I denotes the identity matrix.

The solution of (3.6) is the solution of the following linear system

$$My = s \tag{3.7}$$

where $M = (I + J(\alpha, x)J(\alpha, x)^T)$ and $s = b - A(\alpha)x$. Here $r = -y$ and $\alpha = J(\alpha, x)^T y$.

The SNLS algorithm consists of the following steps. Computing an initial value x from $A(\bar{\alpha})x = b$ for some given initial value $\bar{\alpha}$; solving (3.7) with $\bar{\alpha}$ given and x computed to obtain the approximation of the unknowns r and $\Delta\alpha$; updating $\bar{\alpha}$; computing the new vectors x and r by solving a LS problem with updated matrix $A(\alpha)$; repeating these steps.

3.2.2 Successive Nonlinear Least Squares Algorithm

It follows from the analysis in previous sections that there is the SNLS algorithm as follows.

ALGORITHM 3.2.1 (SNLS ALGORITHM) *Given* $A(\alpha)$, b, $\nabla_\alpha(A(\alpha)x)$ *and* $\bar{\alpha}$ *this algorithm computes the NLS solution* x *and the parameter vector* α.

Set $\alpha = \bar{\alpha}$

Compute $A(\alpha)$

Solve $A(\alpha)x = b$

Compute $J(\alpha, x)$

Set $r = A(\alpha)x - b$, $s = -r$

Repeat

Solve $[I + J(\alpha, x)J(\alpha, x)^T]y = s$

Set $\Delta\alpha = J(\alpha, x)^T y$, $\alpha = \alpha + \Delta\alpha$

Compute $A(\alpha)$

Solve $A(\alpha)x = b$

Compute $J(\alpha, x)$

Set $s = b - A(\alpha)x$, $r = -s$

until convergence

One way to find the solution of the second linear system is to compute the QR factorization of the matrix $A(\alpha)$ as follows

$$A(\alpha)x = b \quad \Rightarrow \quad QRx = b.$$

From the normal equations we have

$$Q^T QRx = Q^T b$$

that results in

$$Rx = Q^T b.$$

Multiplying both the sides by R^T we have the following equality

$$R^T Rx = R^T Q^T b.$$

How

$$R = \begin{pmatrix} \bar{R} \\ 0 \end{pmatrix}$$

then

$$(\bar{R}^T \quad 0) \begin{pmatrix} \bar{R} \\ 0 \end{pmatrix} x = (\bar{R}^T \quad 0)\, Q^T b.$$

Calling $\bar{b} = Q^T b$ and $\bar{b} = \begin{pmatrix} \bar{b}_1 \\ \bar{b}_2 \end{pmatrix}$ we have

$$\bar{R}^T \bar{R}x = \bar{R}^T \bar{b}_1$$

that is equivalent to

$$\bar{R}x = \bar{b}_1. \tag{3.8}$$

The triangular system (3.8) can be easily solved by a back substitution method.

Chapter 4

Numerical Experiments

In this chapter we present some numerical experiments by using in MATLAB 6.0 to verify the efficiency of the SNLS method. We also present the modified SNLS (MSNLS) method. The modification is done to reduce the computational cost and to improve the convergence and the accuracy of the SNLS algorithm for Vandermonde system.

It follows from our numerical tests that both the SNLS and MSNLS algorithms converge fast with reasonable tolerance and provide good approximation to the desired solution when the exact problem is perturbed by several uniformly distributed random errors in α, or in both α and b. Also the MSNLS algorithm is better than the SNLS algorithm in the sense of the convergence rate and accuracy of the approximated solution.

In Section 4.1 we present an application of the Vandermonde systems to the

exponential data modelling. In Section 4.2 we give an overview about the MSNLS method. In Section 4.3 the performance of both the SNLS and MSNLS methods is compared with LS, TLS and SNTLN algorithms for a given problem. In Section 4.4 we consider two types of signal to verify the ability of the SNLS method to converge to the exact parameter vector α_e from different choices of initial parameter estimates $\bar{\alpha}$. Toeplitz systems are considered in Section 4.5.

4.1 Vandermonde System Applied to the Exponential Data Modelling

Vandermonde structure frequently appears in many nonlinear applications such as exponential data modelling problems. For m given uniformly sample data points y_i, the fitting model function is

$$y_i \approx \sum_{j=1}^{n} x_j a_j^i = \sum_{j=1}^{n} (a_j e^{\sqrt{-1}\phi_j}) e^{(-d_j + 2\pi\sqrt{-1} f_j) i \Delta t},$$

$i = 0, \ldots, m$, where n is the model order and Δt is the constant sampling interval. The objective of this problem is to obtain an estimative of the frequencies f_j, damping factors d_j, amplitudes a_j and phases ϕ_j, $j = 1, \ldots, n$. Frequencies and damping factors are obtained from methods based on TLS where Toeplitz structure is considered. The linear parameters x_j which contains the amplitudes a_j and phases ϕ_j, $1 \leq j \leq n$, are estimated from solving the overdetermined Vandermonde

system

$$A(\alpha)x = b \tag{4.1}$$

where

$$A(\alpha) = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \cdots & \alpha_n^2 \\ \vdots & \vdots & & \vdots \\ \alpha_1^{m-1} & \alpha_2^{m-1} & \cdots & \alpha_n^{m-1} \end{pmatrix}, m \geq n, \quad \alpha = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{pmatrix}, \quad b = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{m-1} \end{pmatrix}.$$

Since the problem (4.1) is nonlinear with respect to α, a good initial estimated parameter vector $\bar{\alpha}$ is need. There is an perturbation h in the initial parameter vector $\bar{\alpha}$ such that

$$\min \left\| \begin{matrix} r \\ h \end{matrix} \right\|_2$$

$$\text{s.t. } A(\alpha)x = A(\bar{\alpha} + h)x = b + r.$$

4.2 Modified Successive Nonlinear Least Squares Algorithm

In this section we can solve the NLS problem by the approach proposed by Yalamov and Yuan [39]. Our objective is to reduce the computational cost, and to improve the convergence rate and the accuracy of the SNLS algorithm for Vandermonde structures. Of course, this modification can be used for other structures.

Considering the NLS problem (3.3), we choose the matrices X and Γ in such a way that

$$\Delta A(\alpha)x = X\Gamma\Delta\alpha, \tag{4.2}$$

where both X and Γ have nice structure for structured problems. For example, if $A(\alpha)$ is Vandermonde, then

$$X = \begin{pmatrix} x_1 \cdots x_n & & 0 \\ & x_1 \cdots x_n & \\ & & \ddots \\ 0 & & x_1 \cdots x_n \end{pmatrix}$$

and

$$\Gamma = \begin{pmatrix} \text{zeros}(s) \\ I_{s\times s} \\ 2\text{diag}(\alpha_i) \\ \vdots \\ (m-1)\text{diag}(\alpha_i^{m-2}) \end{pmatrix}$$

where $X \in \mathbf{R}^{m\times mn}$ and $\Gamma \in \mathbf{R}^{mn\times s}$, $i = 1, \ldots, s$.

Now we consider the problem

$$\min \left\| \begin{matrix} r \\ \Gamma\Delta\alpha \end{matrix} \right\|_2 \tag{4.3}$$

$$\text{s.t. } A(\alpha)x + X\Gamma\Delta\alpha = b + r.$$

Then (4.3) is equivalent to

$$\begin{aligned} &\min \left\| \begin{array}{c} r \\ \\ \Gamma\Delta\alpha \end{array} \right\|_2 \\ &\text{s.t. } (-I, X) \begin{pmatrix} r \\ \\ \Gamma\Delta\alpha \end{pmatrix} = b - A(\alpha)x. \end{aligned} \tag{4.4}$$

From Yalamov and Yuan [39], one way to find the LS solution of the problem (4.4) is to solve

$$(I + XX^T)y = s \tag{4.5}$$

where $s = b - A(\alpha)x$. Note that (4.5) is very easily solvable system (in general, it is a diagonal system). From (4.5) it follows that $r = -y$ and

$$\Gamma\Delta\alpha = X^T y. \tag{4.6}$$

Since (4.6) may not be a consistent system, we consider its least squares solution as follows:

$$\Gamma^T\Gamma\Delta\alpha = \Gamma^T X^T y, \tag{4.7}$$

otherwise, we obtain $\Delta\alpha$ just by its first s equations. Of course, we can consider other type of solutions for $\Delta\alpha$ from (4.6). Note that $\Gamma^T X^T = (X\Gamma)^T = J(\alpha, x)^T$.

Then (4.7) can be solved by

$$D\Delta\alpha = J(\alpha, x)y \tag{4.8}$$

where $D = \Gamma^T\Gamma$.

For Vandermonde systems, for instance, $I + XX^T = (1 + \|x\|_2^2)I$ and D is an $n \times n$ diagonal matrix whose entries are given by $d_{ii} = \sum_{j=1}^{m} j^2 \alpha_i^{2j-2}$. The second linear system is solved by QR decomposition as described in Section 3.2.

4.2.1 Modified Successive Nonlinear Least Squares Algorithm

ALGORITHM 4.2.1 (MSNLS ALGORITHM) *Given $A(\alpha)$, b, $\nabla_\alpha A(\alpha)$ and $\bar{\alpha}$ this algorithm computes the NLS solution x and the parameter vector α.*

Set $\alpha = \bar{\alpha}$

Compute $A(\alpha)$

Solve $A(\alpha)x = b$

Compute $J(\alpha, x)$

Set $r = A(\alpha)x - b$, $s = -r$

Repeat

Set $y = \frac{s}{1+\|x\|_2^2}$

Set $\Delta\alpha = diag\left(\frac{1}{\sum_{j=1}^{m} j^2 \alpha_i^{2j-2}}\right) J(\alpha, x)^T y$

Set $\alpha = \alpha + \Delta\alpha$

Compute $A(\alpha)$

Solve $A(\alpha)x = b$

Compute $J(\alpha, x)$

Set $s = b - A(\alpha)x$, $r = -s$

until convergence

4.3 Performance of Both the SNLS and MSNLS Methods for Vandermonde Overdetermined Systems

In this section we make some numerical tests to verify the efficiency of both the SNLS and MSNLS methods for reconstructing a given exact problem. The test problem is the same as that proposed by Rosen, Park and Glick in [33].

Given an initial value of $A(\alpha)$, we assume that there exists a exact Vandermonde matrix $A(\alpha_e)$ and vector b_e such that

$$A(\alpha_e)x_e = b_e$$

for some exact solution x_e. Then, the overdetermined system has a solution x_e with zero residual for $A(\alpha_e)$ and b_e. In practical applications, since the data are perturbed by noise, the perturbed Vandermonde matrix $A(\alpha_e + \delta_\alpha) = A(\bar{\alpha})$ and the perturbed vector b_p are assumed to be known. Random perturbations δ_α are generated in the vector parameter α to give a Vandermonde matrix $A(\alpha + \delta_\alpha)$.

The components of δ_α are uniformly distributed random variables in the interval $[-\gamma, \gamma]$.

The matrix $A(\alpha)$ and vectors r and x_p are computed via LS, TLS, SNTLS2, SNLS and MSNLS, satisfying the perturbed system

$$A(\alpha)x_p = A(\bar{\alpha} + h)x_p = b_p - r,$$

where h is the error for a parameter vector α. For LS, $A(\alpha) = A(\bar{\alpha})$ since the matrix is not perturbed. For TLS, $A(\alpha) = A(\bar{\alpha}) + E$ for some error matrix E since TLS does not preserves the structure or to consider the nonlinear dependence of A on α into account in computing the solution.

In the test we consider $A(\alpha_e)$ a 15×3 Vandermonde matrix such that

$$\alpha_e = \begin{pmatrix} e^{-0.1+2\pi\sqrt{-1}*0.5} \\ e^{-0.2+2\pi\sqrt{-1}*0.4} \\ e^{-0.3+2\pi\sqrt{-1}*0.3} \end{pmatrix}, \quad x_e = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad b_e = A(\alpha_e)x_e.$$

Each data obtained represents the average of 100 solutions computed with different random values in the interval $[-\gamma, \gamma]$. For both SNLS and MSNLS methods the number of iterations was limited to 10 while for the SNTLN2 algorithm the number of iterations was limited to 20. When the convergence test is not satisfied, that is, $\|\Delta\alpha\| > 1.0e-6$ or $\|\Delta x\| > 1.0e-6$, the result given by our methods in the 10th iteration is taken as the final solution.

The results of LS, TLS and SNTLN2 are obtained from [33]. The SNTLN2* method was implemented in MATLAB by us. Unfortunately we do not have notice of

how their minimization subproblem was solved. In our implementation we use a CG method for solving the minimization subproblem without a suitable preconditioner. Then, for the tests, our results are not so good as that the SNTLN2 method.

4.3.1 Numerical Results When b is Unperturbed

In Table 4.1 the test results are given for the problem where only the parameter vector α is perturbed by uniformly distributed random variables in the interval $[-\gamma, \gamma]$. For $\gamma = 1.0e-8$ and $\gamma = 1.0e-6$ both the SNLS and MSNLS methods converges in one step. For other values of γ, its iterations was terminated after 10 iterations. The reason for this fact is that we did not apply the efficient preconditioned CG method described in Section 2.2 to solve very ill-conditioned Vandermonde System (4.1). But for all tests, the relative error with respect to the vector x obtained by the MSNLS method is better than that by LS, TLS, SNTLN2* and SNLS methods.

The CPU time, in seconds, of both MSNLS and SNTLN2* methods is given in Table 4.2.

Figures 4.1 and 4.2 display the relative error computed by both MSNLS and SNTLN2* methods with respect to x and α for $\varepsilon = 1.0e-16$. In this case the convergence of the SNTLN2* method is still better than the convergence of the MSNLS method.

γ	LS	TLS	SNTLN2	SNTLN2*	SNLS	MSNLS
1.0e-8	4.8e-8	4.8e-8	4.9e-15	4.5e-8	2.3e-8	1.9e-8
1.0e-6	4.5e-6	4.5e-6	2.2e-16	2.9e-16	2.2e-6	2.0e-6
1.0e-4	4.9e-4	4.9e-4	1.7e-14	3.5e-13	9.5e-4	1.9e-4
1.0e-3	5.0e-3	5.0e-3	3.5e-16	3.4	9.3e-3	1.8e-3
1.0e-2	4.5e-2	4.6e-2	2.1e-14	3.0	1.2e-1	1.6e-2
1.0e-1	4.2e-1	4.2e-1	5.1e-2	1.3	3.8	1.2e-1

Table 4.1: Solution error $\frac{\|x_p - x_e\|_2}{\|x_e\|_2}$ of x_p computed by the LS, TLS, SNTLN2, SNTLN2*, SNLS and MSNLS methods when b is unperturbed.

In Figures 4.3 and 4.4 the comparison between both the MSNLS and SNTLN2* methods with respect to x and α, respectively, is shown. Here the MSNLS method converge fast when compared to the SNTLN2* method.

4.3.2 Numerical Results When b is Perturbed

For the perturbation in vector b_e, two different cases are analyzed. First, b_e is affected by uniformly distributed random variables in the interval [-1.0e-8,1.0e-8]. In the second case, b_e is perturbed by uniformly distributed random variables in the interval $[-\gamma, \gamma]$, like α_e. The objective of these tests is to verify how both the SNLS and MSNLS methods recovers the correct solution when α_e and b_e are affected by the different kinds of errors.

γ	SNTLN2*	MSNLS
1.0e-8	3.8e-1	1.8e-1
1.0e-6	8.2e-1	6.0e-2
1.0e-4	1.1	1.9
1.0e-3	5.3	1.8
1.0e-2	5.3	1.9
1.0e-1	5.0	1.9

Table 4.2: CPU time of both SNTLN2* and MSNLS methods when b is unperturbed.

Numerical Results When b is Perturbed by Uniformly Distributed Random Variables in the Interval [-1.0e-8,1.0e-8]

In Table 4.3 the results obtained for b perturbed by uniformly distributed random variables in the interval [-1.0e-8,1.0e-8] are given. Both the SNLS and MSNLS methods converges fast for $\gamma = 1.0e-8$ and $\gamma = 1.0e-6$ but they not satisfies the stopping criterion for all tests when other values of γ are considered. For all tests the relative error in x of the MSNLS method is better than that of LS, TLS and SNLS.

The CPU time of both MSNLS and SNTLN2* methods when b is perturbed by uniformly distributed random variables in the interval [-1.0e-8,1.0e-8] is given in Table 4.4.

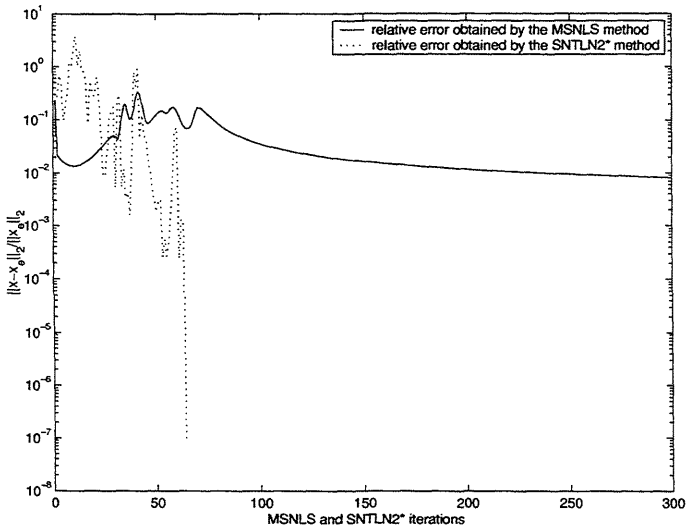

Figure 4.1: Convergence of both the MSNLS and SNTLN2* methods with respect to x when b is unperturbed, $\varepsilon = 1.0\text{e} - 16$ and $\gamma = 1.0\text{e} - 1$.

For $\varepsilon = 1.0\text{e} - 16$ the convergence of the SNTLN2* method is still better than the convergence of the MSNLS method. This fact is shown in Figures 4.7 and 4.8.

The comparison between both the MSNLS and SNTLN2* methods with respect to x and α, respectively, is shown in Figures 4.7 and 4.8.

In accord with Figures 4.7 and 4.8 the convergence of the MSNLS method is better than that the SNTLN2* method for a small tolerance.

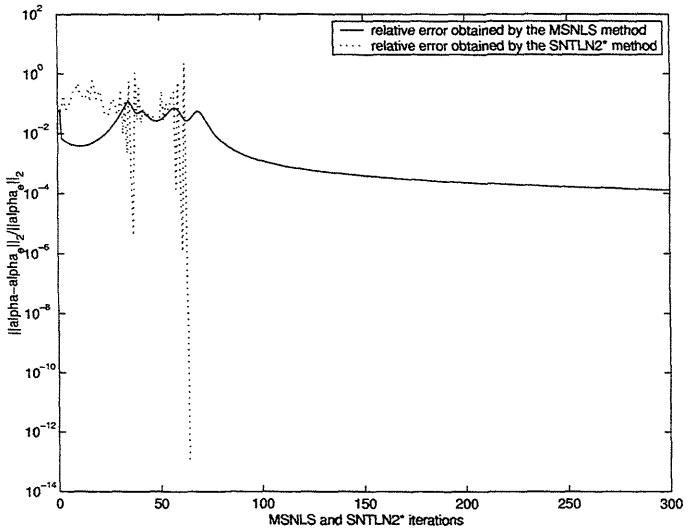

Figure 4.2: Convergence of both the MSNLS and SNTLN2* methods with respect to α when b is unperturbed, $\varepsilon = 1.0\text{e} - 16$ and $\gamma = 1.0\text{e} - 1$.

Numerical Results When b is Perturbed by Uniformly Distributed Random Variables in the Interval $[-\gamma, \gamma]$

When b is perturbed by uniformly distributed random variables in the interval $[-\gamma, \gamma]$, the accuracy of the MSNLS method is better than that of LS, TLS, SNTLN2, SNTLN2* and SNLS. The results are given in Table 4.5. Unfortunately, the convergence test of both SNLS and MSNLS methods is not satisfied within 10 iterations for $\gamma = (1.0\text{e} - 4, \ldots, 1.0\text{e} - 1)$ for all tests.

In Table 4.6 the CPU time of both MSNLS and SNTLN2* methods when b is perturbed by uniformly distributed random variables in the interval $[-\gamma, \gamma]$ is presented.

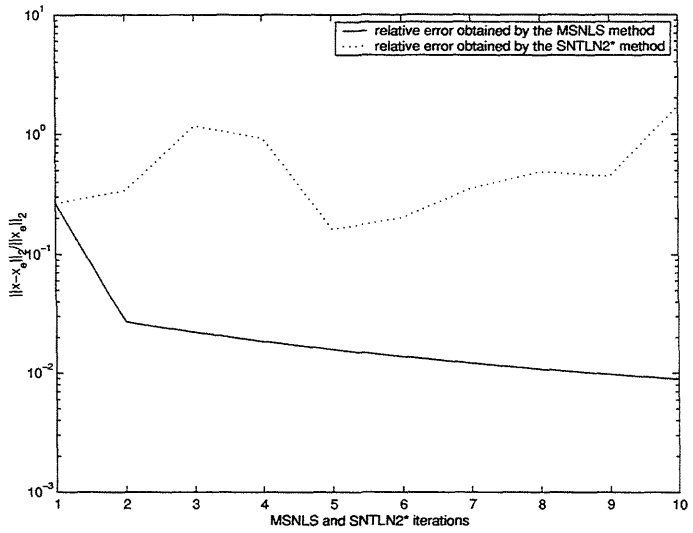

Figure 4.3: Comparison between both the MSNLS and SNTLN2* methods with respect to x when b is unperturbed and $\gamma = 1.0\text{e} - 1$.

When $\varepsilon = 1.0\text{e} - 16$ the SNTLN2* method converges fast and provides good approximation to the desired solutions.

From Figures 4.11 and 4.12 the convergence of the MSNLS method is better than that SNTLN2* method.

4.4 Parameter Estimation Problems

To verify the capacity of the SNLS method we consider the following parameter estimation problem:

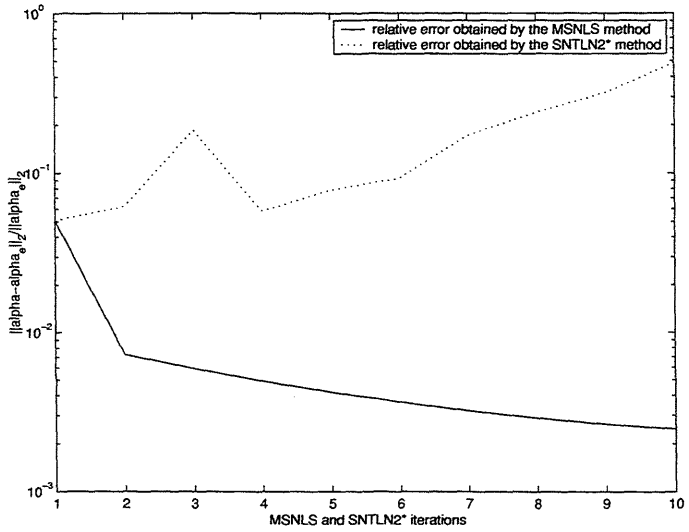

Figure 4.4: Comparison between both the MSNLS and SNTLN2* methods with respect to α when b is unperturbed and $\gamma = 1.0\text{e} - 1$.

Suppose that a measured signal $f(t)$ at m observed values of t has the form

$$f_i = f(t_i) + \eta_i, \quad i = 1, \ldots, m \tag{4.9}$$

where η_i represent noise or error in the measurement and $f(t)$ is one of the following forms

$$1.\ f(t) = \sum_{j=1}^{n} x_j e^{-a_j t}$$

or

$$2.\ f(t) = \sum_{j=1}^{n} x_j e^{-(t-\alpha_j)^2/\sigma^2}.$$

We shall use the SNLS method to obtain the approximation x_p to the exact solution x_e. The data for the first type of signal was obtained from Osborne and

γ	LS	TLS	SNTLN2	SNTLN2*	SNLS	MSNLS
1.0e-8	4.5e-8	4.5e-8	2.5e-8	4.6e-8	2.4e-8	1.9e-8
1.0e-6	5.0e-6	5.0e-6	2.5e-8	1.3e-8	2.5e-6	2.0e-6
1.0e-4	5.0e-4	5.0e-4	2.7e-8	1.3e-8	9.1e-4	1.9e-4
1.0e-3	4.2e-3	4.2e-3	2.7e-8	3.1	9.2e-3	1.8e-3
1.0e-2	4.9e-2	4.9e-2	2.4e-8	2.6	1.3e-1	1.6e-2
1.0e-1	4.7e-1	4.7e-1	1.1e-1	1.8	4.2	1.2e-1

Table 4.3: Solution error $\frac{\|x_p - x_e\|_2}{\|x_e\|_2}$ of x_p computed by the LS, TLS, SNTLN2, SNTLN2*, SNLS and MSNLS methods when b is perturbed by uniformly distributed random variables in the interval [-1.0e-8,1.0e-8].

Smyth [29], and the second type is the sum of Gaussian functions which is similar to that in [24].

For the first case we take 30 points t_i in the interval $t \in [0, 1]$ and set the values $\alpha_e = (0 \quad 4 \quad 7)^T$ and $x_e = (0.5 \quad 2 \quad -1.5)^T$. Then, we have $m = 30$, $n = 3$, $s = 3$. This signal is used to measure the effect of errors in the data vector f_i on the computed parameter estimate α_p. Uniformly distributed random errors η_i, $i = 1, \ldots, m$ in the interval $[-\epsilon, \epsilon]$ were added to $f(t_i)$ to give f_i as in 4.9.

For the second case we take $\sigma^2 = 0.05$, $\alpha_e = (0.1 \quad 0.3 \quad 0.5 \quad 0.9)^T$, $x_e = (1.0 \quad 0.5 \quad 2.0 \quad 0.25)^T$ and 64 values of t_i equally spaced in the interval [0,1]. So we have $m = 64$, $n = 4$, and $s = 4$. The error in $f(t_i)$ is given by uniformly

γ	SNTLN2*	MSNLS
1.0e-8	2.1e-1	1.5e-1
1.0e-6	8.5e-1	2.2e-1
1.0e-4	9.7e-1	1.8
1.0e-3	5.4	2.0
1.0e-2	5.1	2.0
1.0e-1	4.9	1.9

Table 4.4: CPU time of both SNTLN2* and MSNLS methods when b is perturbed by uniformly distributed random variables in the interval [-1.0e-8,1.0e-8].

distributed random variables η_i, $i = 1, \ldots, m$ in the interval $[-\epsilon, \epsilon]$.

The ability of the SNLS method for different initial parameter estimates $\bar{\alpha}$ was tested with the two types of signals. The initial parameter estimation is

$$\bar{\alpha} = \alpha_e + \delta_\alpha$$

where the components of δ_α are uniformly distributed random variables in the interval $[-\gamma, \gamma]$. For all cases, uniformly distributed random variables η_i, $|\eta_i| \leq 1.0e-7$ are added to f_i such that

$$f_i = f(t_i) + \eta_i \quad i = 1, \ldots, m.$$

The number of iterations is limited to 100 for six values of γ. Here we consider $\gamma = (0, 0.01, 0.02, 0.03, 0.05, 0.07)$.

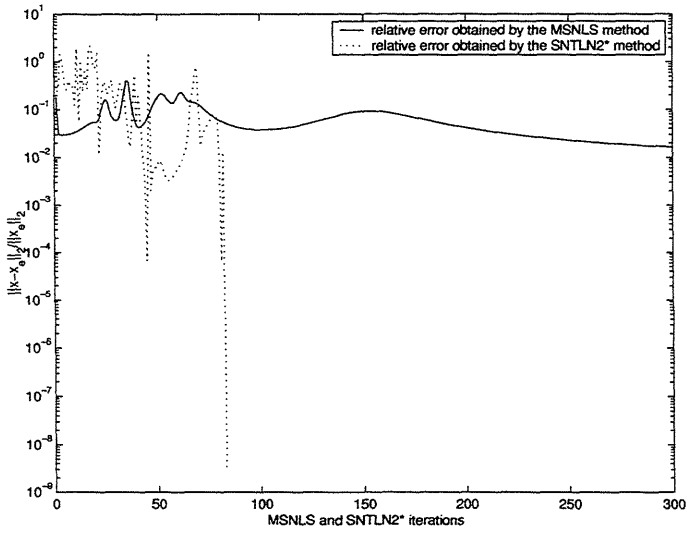

Figure 4.5: Convergence of both the MSNLS and SNTLN2* methods with respect to x when b is perturbed by uniformly distributed random variables in the interval [-1.0e-8,1.0e-8], $\varepsilon = 1.0e-16$ and $\gamma = 1.0e-1$.

4.4.1 Numerical Results for the First Type of Signal

First we consider numerical tests for the first case. Uniformly distributed random errors in the interval [-1.0e-7,1.0e-7] are added in f_i. Numerical tests are also done with a single outlier at one of the points t_i. For no outliers, the results of the SNLS method are given in Table 4.7. Since the norm of the error in both x and α are minimized, different digits of correction were considered. Table 4.8 displays that the number of iterations decreases significatively for different digits of corrections.

The effect of outliers is verified by adding 1 outlier (± 0.1) in a random position of the vector f. In Table 4.9 we give the average result of the obtained results for

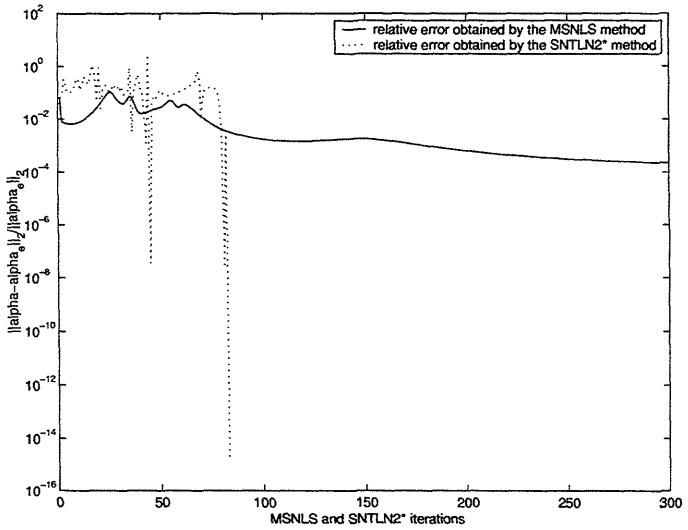

Figure 4.6: Convergence of both the MSNLS and SNTLN2* methods with respect to α when b is perturbed by uniformly distributed random variables in the interval [-1.0e-8,1.0e-8], $\varepsilon = 1.0e-16$ and $\gamma = 1.0e-1$.

different values of γ and 100 test problems. The number of iterations and the approximated solution of x and α are given in Table 4.10

4.4.2 Numerical Results for the Second Type of Signal

The obtained results are compared with the tests of both the SNTLN1 (SNTLS with L_1 norm) and SNTLN2 (SNTLS with L_2 norm) algorithms presented by Rosen, Park and Glick in [33]. The results of 20 solutions, each with different random values of δ_α when no outliers was considered are given in Table 4.11. If $\gamma = 0.02$, the SNLS method converges for all cases, as both SNTLN1 and SNTLN2.

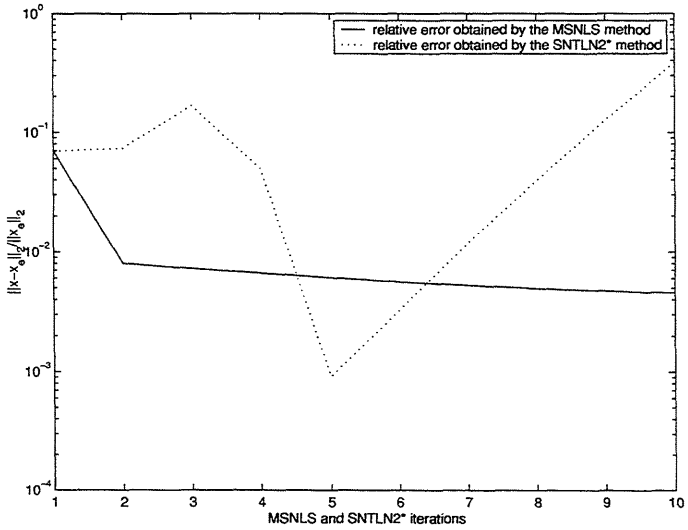

Figure 4.7: Comparison between both the MSNLS and SNTLN2* methods with respect to x when b is perturbed by uniformly distributed random variables in the interval [-1.0e-8,1.0e-8] and $\gamma = 1.0e-1$.

For $\gamma = 0.07$ the percentage of convergence of the SNLS method drops to 80% while the percentage of convergence of both SNTLN1 and SNTLN2 drops to 75%.

In order to investigate the effect of outliers, we consider 10 and 25 outliers. The values of γ are the same as that used above. The value of each outlier is ± 0.1. These values are added to f_i (in addition to η_i) at randomly selected positions of the 64 points. The SNLS algorithm, as the SNTLN2 algorithm, did not satisfy the convergence criterion in many cases.

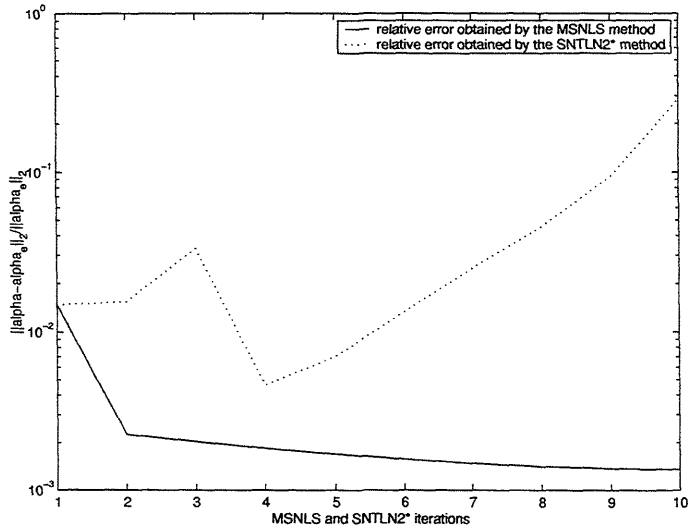

Figure 4.8: Comparison between both the MSNLS and SNTLN2* methods with respect to α when b is perturbed by uniformly distributed random variables in the interval [-1.0e-8,1.0e-8] and $\gamma = 1.0e-1$.

4.5 Toeplitz Systems

In this section we present some numerical tests considering a Toeplitz system. The problem is take from [22]. The test problem is constructed so that both $A(\alpha_e)$ and b_e are known. Random perturbations δ_α on α_e and δ_b on b_e was added to give a Toeplitz matrix $A(\alpha_e + \delta_\alpha)$ and $b = b_e + \delta_b$. The components of δ_α are uniformly distributed random variables in the interval $[-\gamma, \gamma]$ and δ_b is randomly generated and scaled so that $\|e\|_2 = 0.01\|b\|_2$.

γ	LS	TLS	SNTLN2	SNTLN2*	SNLS	MSNLS
1.0e-8	4.6e-8	4.6e-8	2.5e-8	5.1e-8	2.2e-8	1.8e-8
1.0e-6	4.6e-6	4.6e-6	2.5e-6	1.3e-6	2.4e-6	2.2e-6
1.0e-4	4.6e-4	4.6e-4	2.3e-4	1.3e-4	8.8e-4	1.8e-4
1.0e-3	4.7e-3	4.7e-3	2.5e-3	2.9	9.4e-3	1.7e-3
1.0e-2	4.3e-2	4.3e-2	2.7e-2	2.2	1.1e-1	1.6e-2
1.0e-1	5.1e-1	5.1e-1	3.4e-1	1.5	4.3	1.1e-1

Table 4.5: Solution error $\frac{\|x_p - x_e\|_2}{\|x_e\|_2}$ of x_p computed by the LS, TLS, SNTLN2, SNTLN2*, SNLS and MSNLS methods when b is perturbed by uniformly distributed random variables in the interval $[-\gamma, \gamma]$.

In the test, the first column of the Toeplitz matrix $A(\alpha)$ is given by

$$col : a_{i,1} = \begin{cases} \frac{1}{\sqrt{2\pi\alpha^2}} e^{\frac{-(\omega-i+1)^2}{2\alpha^2}} & , i = 1, 2, \ldots, 2\omega + 1 \\ 0 & , \text{otherwise} \end{cases}$$

and its row is given by

$$row = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \end{bmatrix}.$$

Here $A(\alpha) \in \mathbf{R}^{n \times n}$, $n = 64$, $\alpha_e = 1.25$ and $\omega = 8$. The exact solution x_e is not known.

The vector b is given as follows

$$b = \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix}^T + e.$$

γ	SNTLN2*	MSNLS
1.0e-8	2.6e-1	4.9e-2
1.0e-6	6.7e-1	2.2e-1
1.0e-4	9.4e-1	1.9
1.0e-3	5.3	1.8
1.0e-2	5.1	1.9
1.0e-1	4.9	1.9

Table 4.6: CPU time of both SNTLN2* and MSNLS methods when b is perturbed by uniformly distributed random variables in the interval $[-\gamma, \gamma]$.

Our purpose is to obtain the exact parameter α_e via SNLS. The obtained results represents the average of 100 solutions, each with different random values in the given interval. The number of iterations was limited in 20 iterations. When the stopping criterion

$$\|\Delta\alpha\| \leq \varepsilon \quad \text{and} \quad \|\Delta x\| \leq \varepsilon, \quad \varepsilon = 1.0e-6,$$

is not satisfied, the result obtained at the 20th iteration is taken as the final solution. In our tests the stopping criterion is satisfied, at most, in 5 iterations. The results are given in Table 4.12. This fact shows that the SNLS algorithm can be fast enough, especially if some fast or super fast methods are applied for the basic iteration step.

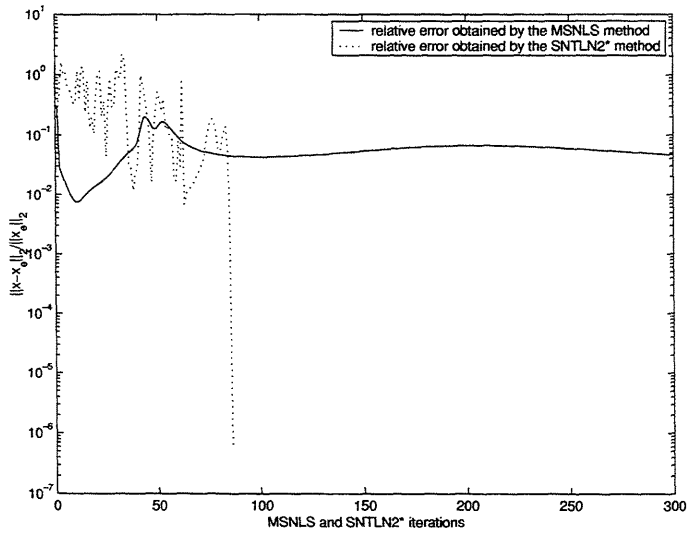

Figure 4.9: Convergence of both the MSNLS and SNTLN2* methods with respect to x when b is perturbed by uniformly distributed random variables in the interval $[\gamma, \gamma]$, $\varepsilon = 1.0\text{e} - 16$ and $\gamma = 1.0\text{e} - 1$.

γ	$\frac{\|x_p - x_e\|_2}{\|x_e\|_2}$	$\frac{\|\alpha_p - \alpha_e\|_2}{\|\alpha_e\|_2}$	$\|\Delta x\|$	$\|\Delta\alpha\|$	# it.
0	5.4e-8	5.9e-9	8.3e-8	8.2e-8	1
0.01	1.1e-3	9.5e-4	9.2e-7	9.1e-7	40
0.02	2.4e-3	1.9e-3	9.1e-7	9.1e-7	44
0.03	3.6e-3	2.9e-3	9.2e-7	9.2e-7	46
0.05	5.2e-3	4.2e-3	9.2e-7	9.1e-7	49
0.07	7.4e-3	6.5e-3	9.1e-7	9.1e-7	51

Table 4.7: Convergence of the SNLS method for 100 tests of the first type of signal when no outliers was considered.

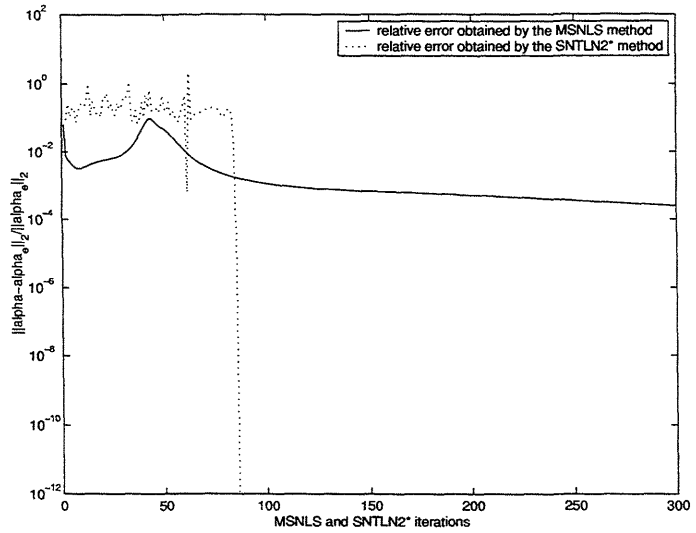

Figure 4.10: Convergence of both the MSNLS and SNTLN2* methods with respect to α when b is perturbed by uniformly distributed random variables in the interval $[\gamma, \gamma]$, $\varepsilon = 1.0\text{e} - 16$ and $\gamma = 1.0\text{e} - 1$.

ε	$\frac{\|x_p - x_e\|_2}{\|x_e\|_2}$	$\frac{\|\alpha_p - \alpha_e\|_2}{\|\alpha_e\|_2}$	$\|\Delta x\|$	$\|\Delta\alpha\|$	# it.
1.0e-6	7.4e-3	6.5e-3	9.1e-7	9.1e-7	51
1.0e-4	8.3e-3	6.5e-3	9.1e-5	9.1e-5	24
1.0e-3	8.4e-3	6.9e-3	8.8e-4	8.7e-4	11
1.0e-2	1.6e-2	7.7e-3	6.0e-3	5.9e-3	2
1.0e-1	1.7e-2	7.9e-3	7.1e-3	7.0e-3	1

Table 4.8: Convergence of the SNLS method for 100 tests of the first type of signal when no outliers was considered for different digits of correction ε and $\gamma = 0.07$.

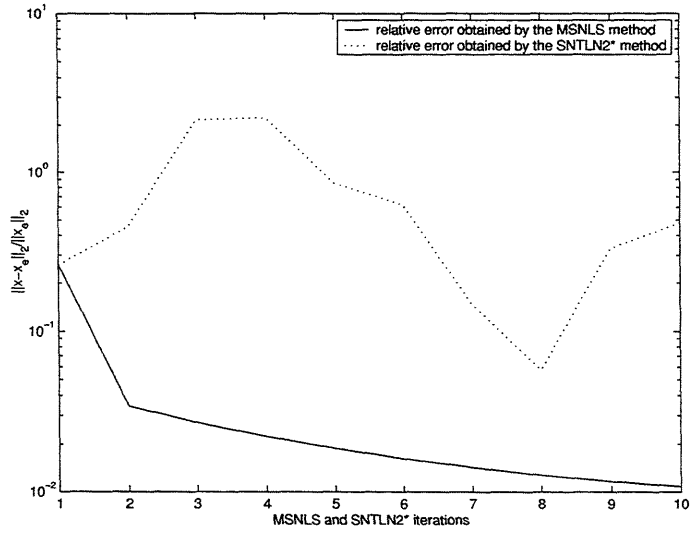

Figure 4.11: Comparison between both the MSNLS and SNTLN2* methods with respect to x when b is perturbed by uniformly distributed random variables in the interval $[\gamma, \gamma]$ and $\gamma = 1.0\text{e} - 1$.

γ	$\frac{\|x_p - x_e\|_2}{\|x_e\|_2}$	$\frac{\|\alpha_p - \alpha_e\|_2}{\|\alpha_e\|_2}$	$\|\Delta x\|$	$\|\Delta\alpha\|$	# it.
0	6.0e-2	7.2e-3	9.1e-7	9.1e-7	58
0.01	6.1e-2	7.3e-3	9.1e-7	9.1e-7	57
0.02	6.9e-2	9.2e-3	9.1e-7	9.1e-7	57
0.03	8.5e-2	1.3e-2	9.3e-7	9.2e-7	59
0.05	7.8e-2	1.3e-2	9.2e-7	9.2e-7	57
0.07	7.1e-2	1.3e-2	9.2e-7	9.1e-7	57

Table 4.9: Convergence of the SNLS method for 100 tests of the first type of signal when 1 outlier was added in a random component of f_i.

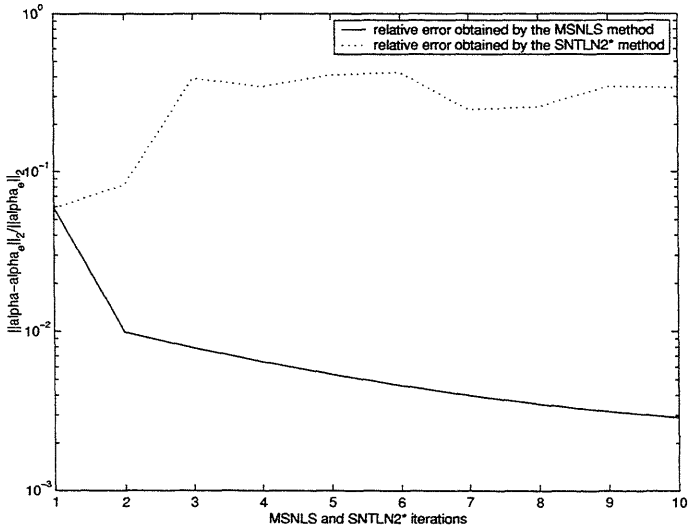

Figure 4.12: Comparison between both the MSNLS and SNTLN2* methods with respect to α when b is perturbed by uniformly distributed random variables in the interval $[\gamma, \gamma]$ and $\gamma = 1.0\text{e} - 1$.

ε	$\frac{\|x_p - x_e\|_2}{\|x_e\|_2}$	$\frac{\|\alpha_p - \alpha_e\|_2}{\|\alpha_e\|_2}$	$\|\Delta x\|$	$\|\Delta\alpha\|$	# it.
1.0e-6	7.1e-2	1.3e-2	9.2e-7	9.1e-7	57
1.0e-4	7.8e-2	1.4e-2	9.1e-5	9.1e-5	29
1.0e-3	7.7e-2	1.4e-2	8.9e-4	8.9e-4	16
1.0e-2	5.9e-2	1.1e-2	7.7e-3	7.7e-3	4
1.0e-1	4.0e-2	9.2e-3	1.5e-2	1.4e-2	1

Table 4.10: Convergence of the SNLS method for 100 tests of the first type of signal for different digits of correction ε and $\gamma = 0.07$ when 1 outlier was added in a random component of f_i.

γ	$\frac{\|x_p-x_e\|_2}{\|x_e\|_2}$	$\frac{\|\alpha_p-\alpha_e\|_2}{\|\alpha_e\|_2}$	$\|\Delta x\|$	$\|\Delta\alpha\|$	# it.
0	5.4e-8	5.9e-9	8.3e-8	8.2e-8	1
0.01	1.1e-3	9.5e-4	9.2e-7	9.1e-7	40
0.02	2.4e-3	1.9e-3	9.1e-7	9.1e-7	44
0.03	3.6e-3	2.9e-3	9.2e-7	9.2e-7	46
0.05	5.2e-3	4.2e-3	9.2e-7	9.1e-7	49
0.07	7.4e-3	6.5e-3	9.1e-7	9.1e-7	51

Table 4.11: Convergence of the SNLS method for 100 tests of the second type of signal when no outliers was considered.

γ	$\frac{\|\alpha_p-\alpha_e\|_2}{\|\alpha_e\|_2}$	$\|\Delta\alpha\|$	# it.
1.0e-8	3.9e-9	3.4e-23	3
1.0e-6	4.3e-7	4.1e-7	4
1.0e-4	4.0e-5	1.9e-23	5
1.0e-3	3.7e-4	1.0e-22	3
1.0e-2	3.9e-3	1.2e-23	4
1.0e-1	3.8e-2	2.7e-21	5

Table 4.12: Convergence of the SNLS method for 100 tests when a Toeplitz system is considered.

Chapter 5

Conclusions

In this dissertation we propose a new way to solve structured NLS problems based on the TLS formulation. We consider an initial estimate to the parameter vector α and an initial value to the vector x. The errors in both α and x are minimized iteratively by successive solutions of two LS problems at each iteration. The first system is a symmetric and positive definite linear system. Its solution can be obtained by CG methods with a suitable preconditioner. If $A(\alpha)$ has some structure, the SNLS method keeps the same structure for the errors and does not require the matrix M explicitly when some fast iterative method is used. Then, a few vectors with length not greater than $m + n$ are required. The solution of the second linear system depends on the structure of the data matrix, Toeplitz, Hankel or Vandermonde. One of the advantages of the SNLS method is that the structure of the error matrix is preserved. Another one is the low computational cost at each

iteration. Another version of the SNLS method, the MSNLS method, is proposed to solve Vandermonde systems and it can be extended to other structures. This version is based on solution of two diagonal systems for solving the first linear system. The computational cost is reduced and the related modification provides a still better accuracy for the method. Numerical experiments confirm that both the SNLS and MSNLS methods converges to good approximation to the desired solution. This fact shows that our methods can be fast enough, especially if some fast or super fast methods are applied to solve the two related linear systems. Even though the numerical performance of our methods is not as good as that of the SNTLN2 algorithm our methods are all certainly better than the LS, TLS and SNTLN2* methods. In agreement with Björk, the SNLS method converges fast and is quite suitable when a small desired tolerance is required.

In future, we are trying to consider different algebraic manipulations and direct and iterative methods to obtain the solution of both the underdetermined and the overdetermined linear systems and to verify the effect of this choice in the obtained results.

Appendix A

Matlab Codes

```
function msnls(gamma,opt)

%MSNLS This code reconstruct an exact Vandermonde system

%A(alpha_e)x_e=b_e by the MSNLS method where alpha_e and b_e

%are perturbed by distributed random variables in the interval

%[-gamma,gamma].

TotalOfTests=100;

RelativeErrorAlpha=0;

RelativeErrorX=0;

TotalTime=0;

% Initializing the optional plot

if opt==1

      TotalOfTests=1;
```

```
end
for t=1:TotalOfTests
      % Initial values
      MaxIt=10;
      tol=1.0e-6;
      ErrAlpha=2*tol;
      ErrX=2*tol;
      k=0;
      % Exact Vandermonde test problem
      [V,x_e,alpha_e,b,m,n,ss]=test1;
      % Perturbation on the initial parameter vector alpha
      [alpha_p]=pertalpha(alpha_e,gamma,ss);
      % Perturbation on the exact vector b by distributed
      % random variables in the interval [-1.0e-8,1.0e-8]
      %[b]=pert1b(b,m);
      % Perturbation on the exact vector b by distributed
      % random variables in the interval [-gamma,gamma]
      [b]=pert2b(b,gamma,m);
      % Initial computations
      alpha=alpha_p;
      V=vand(alpha,m,n);
```

```
% Initial least square solution
[Q,R]=qr(V);
R=R(1:n,:);
for c=1:n
     bb(c)=Q(:,c)'*b;
end
bb=bb(:);
% Back substitution
x_old=back(R,bb,n);
x=x_old;
JVx=jacobv(alpha,x,m,n);
r=V*x_old-b;
s=-r;
nrm=norm(s);
% Initial plot data (optional)
if opt==1
     VecErrorAlpha(1)=norm(alpha_e-alpha)/norm(alpha_e);
     VecErrorX(1)=norm(x_e-x_old)/norm(x_e);
end
Time=cputime;
% Main computations
```

```
while (ErrAlpha>tol|ErrX>tol)&k<MaxIt
    % Computing the matrix X
    for i=1:m
    X(i,i*n-(n-1):i*n)=x';
    end
    y=s/(1+x'*x);
    v=X'*y;
    % Computing the matrix D
    for i=1:m
        D(i*n-(n-1):i*n,:)=diag((i)*alpha.^(i-1));
    end
    DD=(D'*D);
    JVxy=JVx'*y;
    for h=1:n
        DeltaAlpha(h)=JVxy(h)/DD(h,h);
    end
    DeltaAlpha=DeltaAlpha(:);
    alpha=alpha+DeltaAlpha;
    % Updating the matrix V
    V=vand(alpha,m,n);
    % Least square solution to the overdetermined system
```

```
[Q,R]=qr(V);
R=R(1:n,:);
bb=Q'*b;
bb=bb(1:n);
% Back substitution
x=back(R,bb,n);
% Updating the Jacobian matrix with respect to new x
JVx=jacobv(alpha,x,m,n);
s=b-V*x;
r=-s;
% Computing the stop criterion
ErrAlpha=norm(DeltaAlpha);
ErrX=norm(x-x_old);
% Plot data
if opt==1
     VecErrorAlpha(k+2)=ErrAlpha/norm(alpha_e);
     VecErrorX(k+2)=ErrX/norm(x_e);
end
% Updating
x_old=x;
k=k+1;
```

```
end
% Plotting the error (optional)
if opt==1
     semilogy(VecErrorAlpha,'-');
     hold on;
     semilogy(VecErrorX,':');
     title('Modified Successive Nonlinear Least
     Squares Method');
     xlabel('MSNLS iterations');
     ylabel('||x-x_e||_2/||x_e||_2 and
     ||alpha-alpha_e||_2/||alpha_e||_2');
     legend('relative error in the vector alpha',
     'relative error in the vector x');
end
Time=cputime-Time;
TotalTime=TotalTime+Time;
RelativeFinalErrorAlpha=norm(alpha-alpha_e)/norm(alpha_e);
RelativeErrorAlpha=RelativeErrorAlpha+
RelativeFinalErrorAlpha;
RelativeFinalErrorX=norm(x-x_e)/norm(x_e);
RelativeErrorX=RelativeErrorX+RelativeFinalErrorX;
```

```
end
% Close the current figure
if opt==1
      hold off
end
TotalTime
AverageOfRelativeErrorAlpha=RelativeErrorAlpha/TotalOfTests
AverageOfRelativeErrorX=RelativeErrorX/TotalOfTests


function snls(gamma,opt1,opt2)
%SNLS - A Successive Least Squares Method for Nonlinear
%Least Squares Problems
TotalOfTests=100;
RelativeErrorAlpha=0;
RelativeErrorX=0;
TotalTime=0;
% Initializing the optional plot
if opt2==1
      TotalOfTests=1;
end
for t=1:TotalOfTests
```

```
% Initial values

MaxIt=10;

tol=1.0e-6;

ErrAlpha=2*tol;

ErrX=2*tol;

k=0;

if opt1==1

     % Vandermonde test problem

     [V,x_e,alpha_e,b,m,n,ss]=test1;

elseif opt1==2

% First type of signal

     [V,x_e,alpha_e,b,m,n,ss]=test2;

elseif opt1==3


     [V,x_e,alpha_e,b,m,n,ss]=test2;

elseif opt1==4


     [V,x_e,alpha_e,b,m,n,ss,omega]=test4;

end

% Perturbation on the initial parameter vector alpha

[alpha_p]=pertalpha(alpha_e,gamma,ss);
```

```
      % Perturbation on the exact vector b by distributed
      % random variables in the interval [-1.0e-8,1.0e-8]
      % (for the Vandermonde test problem)
      %[b]=pert1b(b,m);
      % Perturbation on the exact vector b by distributed
      % random variables in the interval [-gamma,gamma]
      % (for the Vandermonde test problem)
[b]=pert2b(b,gamma,m);
      % Perturbation in vector b (for the Toeplitz test
      % problem)
      %e=randn(n,1);
      %b=b+0.01*norm(b)/norm(e);
      % Initial computations
      alpha=alpha_p;
      if opt1==1
           % Vandermonde test problem
           V=vand(alpha,m,n);
      elseif opt1==2

           V=matrix1(alpha,m,n);
      elseif opt1==3
```

```
    % Second type of signal
    V=matrix2(alpha,m,n);
elseif opt1==4
    % Toeplitz test problem
    V=toep(alpha,omega,n);
end
% Initial least square solution
[Q,R]=qr(V);
R=R(1:n,:);
for c=1:n
    bb(c)=Q(:,c)'*b;
end
bb=bb(:);
% Back substitution
x_old=back(R,bb,n);
x=x_old;
if opt1==1
    % Vandermonde test problem
    JVx=jacobv(alpha,x,m,n);
elseif opt1==2
    % First type of signal
```

```
    JVx=jacobm1(alpha,x,m,n);
elseif opt1==3
    % Second type of signal
    JVx=jacobm2(alpha,x,m,n);
elseif opt1==4
    % Toeplitz test problem
    JVx=jacobt(alpha,omega,x,m,n);
end
r=V*x_old-b;
s=-r;
nrm=norm(s);
% Initial plot data (optional)
if opt2==1
    VecErrorAlpha(1)=norm(alpha_e-alpha)/norm(alpha_e);
    VecErrorX(1)=norm(x_e-x_old)/norm(x_e);
end
Time=cputime;
% Main computations
while (ErrAlpha>tol|ErrX>tol)&k<MaxIt
    % Computing the matrix M
    M=eye(m)+JVx*JVx';
```

```
y=cg(M,s);
DeltaAlpha=JVx'*y;
alpha=DeltaAlpha+alpha;
% Updating the matrix V
if opt1==1
     % Vandermonde test problem
     V=vand(alpha,m,n);
elseif opt1==2
     % First type of signal
     V=matrix1(alpha,m,n);
elseif opt1==3
     % Second type of signal
     V=matrix2(alpha,m,n);
elseif opt1==4
     % Toeplitz test problem
     V=toep(alpha,omega,n);
end
% Least square solution to the overdetermined system
[Q,R]=qr(V);
R=R(1:n,:);
bb=Q'*b;
```

```
bb=bb(1:n);
% Back substitution
x=back(R,bb,n);
s=b-V*x;
% Updating the Jacobian matrix with respect to the
% new x
if opt1==1
     % Vandermonde test problem
     JVx=jacobv(alpha,x,m,n);
elseif opt1==2
     % First type of signal
     JVx=jacobm1(alpha,x,m,n);
elseif opt1==3
% Second type of signal
JVx=jacobm2(alpha,x,m,n);
elseif opt1==4
     % Toeplitz test problem
     JVx=jacobt(alpha,omega,x,m,n);
end
r=JVx*alpha-s;
% Computing the stop criterion
```

```
    ErrAlpha=norm(DeltaAlpha);
    ErrX=norm(x-x_old);
    % Plot data
    if opt2==1
        VecErrorAlpha(k+2)=ErrAlpha/norm(alpha_e);
        VecErrorX(k+2)=ErrX/norm(x_e);
    end
    % Updating
    x_old=x;
    k=k+1;
end
% Plotting the error (optional)
if opt2==1
    semilogy(VecErrorAlpha,'-');
    hold on;
    semilogy(VecErrorX,':');
    title('Successive Nonlinear Least Squares Method');
    xlabel('SNLS iterations');
    ylabel('||x-x_e||_2/||x_e||_2 and
    ||alpha-alpha_e||_2/||alpha_e||_2');
    legend('relative error in the vector alpha',
```

```
        'relative error in the vector x');
    end
    Time=cputime-Time;
    TotalTime=TotalTime+Time;
    RelativeFinalErrorAlpha=norm(alpha-alpha_e)/norm(alpha_e);
    RelativeErrorAlpha=RelativeErrorAlpha+
    RelativeFinalErrorAlpha;
    RelativeFinalErrorX=norm(x-x_e)/norm(x_e);
    RelativeErrorX=RelativeErrorX+RelativeFinalErrorX;
end
% Close the current figure
if opt2==1
    hold off
end
TotalTime
AverageOfRelativeErrorAlpha=RelativeErrorAlpha/TotalOfTests
AverageOfRelativeErrorX=RelativeErrorX/TotalOfTests


function sntln(gamma)
%SNTLN This code reconstruct an exact Vandermonde system
%A(alpha_e)x_e=b_e by the SNTLN algorithm where alpha_e and
```

```
%b_e are perturbed by uniformly distributed random variables
%in the interval [-gamma,gamma].
TotalOfTests=100;
RelativeErrorAlpha=0;
RelativeErrorX=0;
TotalTime=0;
% Initializing the optional plot
if opt==1
      TotalOfTests=1;
end
for t=1:TotalOfTests
      %Initial values
      maxit=20;
      tol=1.0e-6;
      DeltaAlpha=2*tol;
      DeltaX=2*tol;
      k=0;
      % Exact Vandermonde test problem
      [V,x_e,alpha_e,b,m,n,ss]=test1;
      D=zeros(n);
      for i=1:n
```

```
    D(i,i)=1.0e-8;
end
% Perturbation on the initial parameter vector alpha
[alpha_p]=pertalpha(alpha_e,gamma,n);
% Perturbation on the exact vector b by distributed
% random variables in the interval [-1.0e-8,1.0e-8]
%[b]=pert1b(b,m);
% Perturbation on the exact vector b by distributed
% random variables in the interval [-gamma,gamma]
[b]=pert2b(b,gamma,m);
% Initial computations
alpha=alpha_p;
V=vand(alpha,m,n);
% Initial least square solution
[Q,R]=qr(V);
x_old=pcg((R'*R),(R'*Q')*b);
x=x_old;
JVx=jacobv(alpha,x,m,n);
r=V*x-b;
% Initial plot data (optional)
if opt==1
```

```
    VecErrorAlpha(1)=norm(alpha_e-alpha)/norm(alpha_e);
    VecErrorX(1)=norm(x_e-x_old)/norm(x_e);
end
Time=cputime;
% Main computations
while (norm(DeltaAlpha)>tol|norm(DeltaX)>tol)&k<maxit
    DAlpha=1.0e-8*(alpha-alpha_p);
    A=[V,JVx;zeros(n),D];
    bb=[r;DAlpha];
    % Least square solution to the overdetermined
    % system
    [Q,R]=qr(A);
    y=pcg((R'*R),(R'*Q')*bb);
    DeltaX=y(1:n);
    DeltaX=DeltaX(:);
    x=x+DeltaX;
    DeltaAlpha=y(n+1:2*n);
    DeltaAlpha=DeltaAlpha(:);
    alpha=alpha+DeltaAlpha;
    % Updating the matrix V
    V=vand(alpha,m,n);
```

```
    % Updating the Jacobian matrix with respect to the
    % new x
    JVx=jacobv(alpha,x,m,n);
    r=b-V*x;
    % Plot data
    if opt==1
        VecErrorAlpha(k+2)=norm(DeltaAlpha)/norm(alpha_e);
        VecErrorX(k+2)=norm(DeltaX)/norm(x_e);
    end
    % Updating
    k=k+1;
end
% Plotting the error (optional)
if opt==1
    semilogy(VecErrorAlpha,'-');
    hold on;
    semilogy(VecErrorX,':');
    title('Structured Nonlinear Total Least
    Norm Method
    Squares Method');
    xlabel('SNTLN iterations');
```

```
        ylabel('||x-x_e||_2/||x_e||_2 and
        ||alpha-alpha_e||_2/||alpha_e||_2');
        legend('relative error in the vector alpha',
        'relative error in the vector x');
    end
    Time=cputime-Time;
    TotalTime=TotalTime+Time;
    RelativeFinalErrorAlpha=norm(alpha-alpha_e)/norm(alpha_e);
    RelativeErrorAlpha=RelativeErrorAlpha+
    RelativeFinalErrorAlpha;
    RelativeFinalErrorX=norm(x-x_e)/norm(x_e);
    RelativeErrorX=RelativeErrorX+RelativeFinalErrorX;
end
% Close the current figure
if opt==1
    hold off
end
TotalTime
AverageOfRelativeErrorAlpha=RelativeErrorAlpha/TotalOfTests
AverageOfRelativeErrorX=RelativeErrorX/TotalOfTests
```

```
function b=back(R,b,n)

b(n)=b(n)/R(n,n);

for c=n-1:-1:1

      b(c)=(b(c)-R(c,c+1:n)*b(c+1:n))/R(c,c);

end


function x=cg(A,b)

% Conjugate Gradiente Method

%

%This function solves symmetric and positive definite linear

%systems Ax=b.

%

eps=1.0e-6;

maxit=100;

k=0;

[m,n]=size(A);

x=zeros(n,1);

r=b;

rho(1)=norm(r)^2;

while sqrt(rho)>eps*norm(b)&k<maxit

      k=k+1;
```

```
    if k==1
        p=r;
    else
        beta=rho(k)/rho(k-1);
        p=r+beta*p;
    end
    w=A*p;
    alpha=rho(k)/(p'*w);
    x=x+alpha*p;
    r=r-alpha*w;
    rho(k+1)=norm(r)^2;
end

function JVx=jacobm1(alpha,x,m,n)
tt=linspace(0,1,m);
alpha=alpha(:);
JVx=zeros(n,m);
for i=2:m
     JVx(:,i)=-tt(i)*exp(-alpha.*tt(i));
end
for i=1:n
```

```
      JVx(i,:)=JVx(i,:)*x(i);
end
JVx=JVx';

function JVx=jacobm2(alpha,x,m,n)
tt=linspace(0,1,m);
alpha=alpha(:);
JVx=zeros(n,m);
for i=2:m
      JVx(:,i)=40*(tt(i)-alpha).*exp(-20*(tt(i)-alpha).^2);
end
for i=1:n
      JVx(i,:)=JVx(i,:)*x(i);
end
JVx=JVx';

function JVx=jacobt(alpha,omega,x,m,n)
JVx=zeros(m,1);
c=zeros(n,1);
r=zeros(1,n);
for i=1:2*omega+1
```

```
      c(i)=(-1/2*(sqrt(2)*exp(-(omega-i+1)^2/2*alpha^2))*...
           pi*alpha/(pi^(3/2)*alpha^3)+1/2*(sqrt(2)*...
           (omega-i+1)^2*exp(-(omega-i+1)^2/2*alpha^2))*...
           pi*alpha/(sqrt(pi)*alpha^4))*x(i);
end
r(1,1)=c(1,1);
JVx=toeplitz(c,r);
JVx=JVx*x;


function JVx=jacobv(alpha,x,m,n)
%JACOBVAND This subroutine computes the m-by-n jacobian of
%V(alpha)x with respect to alpha
alpha=alpha(:);
JVx=zeros(n,m);
for i=2:m
      JVx(:,i)=(i-1)*alpha.^(i-2);
end
for i=1:n
     JVx(i,:)=JVx(i,:)*x(i);
end
JVx=JVx';
```

```
function V=matrix1(alpha,m,n)

tt=linspace(0,1,m);

alpha=alpha(:);

V=ones(n,m);

for i=1:m

      V(:,i)=exp(-alpha.*(tt(i)));

end

V=V';


function V=matrix2(alpha,m,n)

tt=linspace(0,1,m);

alpha=alpha(:);

V=ones(n,m);

for i=1:m

      V(:,i)=exp(-20*(tt(i)-alpha).^2);

end

V=V';


function b=outliers(b,m)

% OUTLIERS

for i=1:0
```

```
     r=randint(1,1,[1,m]);
    b(r)=b(r)+0.1;
end

function [b]=pert1b(b,m)
%PERT1B This algorithm generates a perturbation on each
%component of the vector b by uniformly distributed random
%variables in the interval [-1.0e-8,1.0e-8].
delta2=1.0e-8*rand(m,1);
b=b+delta2;

function [b]=pert2b(b,gamma,m)
%PERT2B This algorithm generates a perturbation on each
%component of the vector b by uniformly distributed random
%variables in the interval [-gamma,gamma].
delta2=gamma*rand(m,1);
b=b+delta2;

function [alpha_p]=pertalpha(alpha_e,gamma,ss)
%PERTALPHA This algorithm generates a perturbation on the
%parameter vector alpha
```

```
delta1=gamma*rand(ss,1);

alpha_p=alpha_e+delta1;


function [V,x_e,alpha_e,b,m,n,ss]=test1

%TEST1 This algorithm generates the exact Vandermonde test

%problem obtained from Rosen, Park and Glick [33]

m=15;

alpha_e=[exp(-.1+2*pi*sqrt(-1)*.5)

         exp(-.2+2*pi*sqrt(-1)*.4)

         exp(-.3+2*pi*sqrt(-1)*.3)];

ss=length(alpha_e);

n=ss;

x_e=ones(n,1);

alpha=alpha_e;

V=vand(alpha,m,n);

b=V*x_e;


function [V,x_e,alpha_e,b,m,n,ss]=test2

%TEST2 This algorithm generates the exact first type of

%signal obtained from Osborne and Smith [29]

m=30;
```

```
alpha_e=[0; 4; 7];

ss=length(alpha_e);

n=ss;

x_e=[0.5; 2; -1.5];

alpha=alpha_e;

V=matrix1(alpha,m,n);

b=V*x_e;


function [V,x_e,alpha_e,b,m,n,ss]=test3

%TEST3 This algorithm generates the exact second type of

%signal obtained from Osborne and Smith [29]

m=64;

alpha_e=[0.1;0.3;0.5;0.9];

ss=length(alpha_e);

n=ss;

x_e=[1.0;0.5;2.0;0.25];

alpha=alpha_e;

V=matrix2(alpha,m,n);

b=V*x_e;


function [V,x_e,alpha_e,b,m,n,ss,omega]=test4
```

```
m=64;

n=m;

omega=8;

alpha_e=1.25;

ss=1;

alpha=alpha_e;

V=toep(alpha,omega,n);

b=ones(n,1);

x_e=V;


function V=toep(alpha,omega,n)

c=zeros(n,1);

r=zeros(1,n);

for i=1:2*omega+1

     c(i)=(1/sqrt(2*pi*alpha^2))*exp(-(omega-i+1)^2/(2*alpha^2));

end

r(1,1)=c(1,1);

V=toeplitz(c,r);


function V=vand(alpha,m,n)

%VAND This subroutine generates a m-by-n Vandermonde matrix.
```

```
alpha=alpha(:);

V=ones(n,m);

for i=1:m

     V(:,i)=alpha.^(i-1);

end V=V';
```

Bibliography

[1] T. J. Abatzoglou, J. M. Mendel and G. A. Harada, The Constrained Total Least Squares Technique and Its Application to Harmonic Superresolution, *IEEE Trans. Signal Processing*, 39(1991), pp. 1070–1087.

[2] G. Ammar and W. B. Gragg, Superfast Solution of Real Positive Definite Toeplitz Systems, *SIAM J. Matrix Anal. Appl.*, 9 (1988), pp. 61–76.

[3] D. Bates and M. Lindstrom, Nonlinear Least Squares With Conditionally Linear Parameters, in Proceedings of the Statistical Computing Section, American Statistical Association, Washington, DC, 1986, pp. 152–157.

[4] D. M. Bates and D. G. Watts, *Nonlinear Regression Analysis and Its Applications*, John Wiley&Sons, Inc., New York, 1988.

[5] Å. Björck, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, 1996.

[6] Å. Björck, Newton and Rayleigh Quotient Methods for Total Least Squares Problems, in: *Recent Advances in Total Least Squares and Errors-in-Variables Modeling*, SIAM, Philadelphia, 1997.

[7] R. H. Chan, J. G. Nagy and R. J. Plemmons, Displacement Preconditioner for Toeplitz Least Squares Iterations, *Electron. Trans. Numer. Anal.*, 30(1994), pp. 44–56.

[8] R. H. Chan, J. G. Nagy and R. J. Plemmons, FFT-Based Preconditioner for Toeplitz-Block Least Squares Problems, *SIAM J. Numer. Anal.*, 30(1993), pp. 1740–1768.

[9] R. H. Chan, J. G. Nagy and R. J. Plemmons, Circulant Preconditioned Toeplitz Least Squares Iterations, *SIAM J. Matrix. Anal. Appl.*, 15(1994), pp. 80–97.

[10] R. H. Chan, J. G. Nagy and R. J. Plemmons, Generalization of Strang's Preconditioner with Applications to Toeplitz Least Squares Problems, *Numer. Linear Algebra Appl.*, 3(1996), pp. 45–64.

[11] R. H. Chan and M. K. Ng, Conjugate Gradient Methods for Toeplitz Systems, *SIAM Review*, 38 (1996), pp. 427–482.

[12] R. H. Chan and M. K. Ng, Toeplitz Preconditioners for Hermitian Toeplitz Systems, *Linear Algebra Appl.*, 190 (1993), pp. 181–208.

[13] J. W. Cooley, P. A. W. Lewis and P. D. Welsh, *The Fast Fourier Transform and its Aplication*, IEEE Trans. Education, E-12 (1969), pp. 27–34.

[14] P. Henrici, Fast Fourier Transform in Computational Complex Analysis, *SIAM Review*, 21 (1979), pp. 481–527.

[15] J. E. Dennis, *Nonlinear Least Squares and Equations*, in The State of the Art in Numerical Analysis, D. A. H. Jacobs, ed., Academic Press, New York, 1977, pp. 269–312.

[16] J. E. Dennis and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice Hall, Englewood Cliffs, NJ, 1983.

[17] C. Fraley, Algorithms for Nonlinear Least Squares, Tech. Report STAN-CLaSSiC-88-22, Center for Large Scale Scientific Computation, Stanford University, CA, 1988.

[18] P. E. Gill, W. Murray and M. H. Wright, *Practical Optimization*, Academic Press, London and New York, 1981.

[19] G. H. Golub and V. Pereyra, The Differentiation of Pseudo-Inverses and Nonlinear Least Squares Problems whose Variables Separate, *SIAM J. Numer. Anal.*, 10(1973), pp. 413–432.

[20] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., John Hopkins University Press, Baltimore, 1996.

[21] T. A. Grandine, Generating Surface Lofts to Scattered data, Engineering Computing and Analysis Technical Report ECA-TR-157, Boeing Computer Services, Seattle, WA, 1991.

[22] J. Kamm and J. G. Nagy, A Total Least Squares Method for Toeplitz Systems of Equations, *BIT*, 38(1998), pp. 560–582.

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

[24] L. Kaufman and G. Sylvester, Separable Nonlinear Least Squares with Multiple Right-Hand-Sides, *SIAM J. Matrix Anal. Appl.*, 13(1992), pp. 68–89.

[25] R. Kumaresan and D. W. Tufts, Estimating the Parameters of Exponentially Damped Sinusoids and Pole-zero Modeling in Noise, *IEEE Trans. Acoust. Speech Signal Proc.*, 30(1982), pp. 833–840.

[26] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*, Prentice-Hall, Englewood Clifs, NJ, 1974.

[27] J. G. Nagy, Fast Inverse QR Factorization for Toeplitz Matrices, *SIAM J. Sci. Stat. Comput.*, 14 (1993), pp. 1174–1193.

[28] M. R. Osborne, Some Special Nonlinear Least Squares Problems, *SIAM J. Numer. Anal.*, 12 (1975), pp. 571–592.

[29] M. R. Osborne and G. HK. Smyth, A Modified Prony Algorithm for Exponential Function Fitting, *SIAM J. Sci. Comput.*, 16 (1995), pp. 119-138.

[30] H. Park, J. B. Rosen and J. Glick, Structure Preserving Total Least Norm Method and Application to Parameter Estimation, *in Proceedings of the 1995 International Conference on Acoustics, Speech and Signal Processing, Vol. 2*, 1995, pp. 11451–1144.

[31] M. A. Rahman and K. B. Yu, Total Least Squares Approach for Frequency Estimation Using Linear Prediction, *IEEE Trans. Acous. Speech Signal Proc.*, 35(1987), pp. 1141–1454.

[32] J. B. Rosen, H. Park, and J. Glick, Total Least Norm Formulation and Solution for Structured Problems, *SIAM J. Matrix Anal. Appl.*, 17(1996), pp. 110–126.

[33] J. B. Rosen, H. Park, and J. Glick, Structured Total Least Norm for Nonlinear Problems, *SIAM J. Matrix Anal. Appl.*, 20 (1998), pp. 14–30.

[34] G. A. F. Seber and C. J. Wild, *Nonlinear Regression*, John Wiley&Sons, Inc., New York, 1989.

[35] G. Strang, A Proposal for Toeplitz Matrix Calculations, *Stud. Appl. Math.*, 74 (1986), pp. 171–176.

[36] W. Sun, R. J. B. Sampaio, J. Y. Yuan, Quasi-Newton Trust Region Algorithm for Non-Smooth Least Squares Problems, *Appl. Math. and Comput.*, 105 (1999), pp. 183–194.

[37] S. Van Huffel and J. Vandewalle, *The Total Least Squares Problem: Computational Aspects and Analysis*, SIAM, Philadelphia, 1991.

[38] J. S. Walker, *Fast Fourier Transform*, 2nd ed., CRC Press, New York, 1996.

[39] P. Y. Yalamov and J. Y. Yuan, A Successive Least Squares Method for Structured Total Least Squares, *Journal of Computational Mathematics*, (submitted), 2000.