NASA/TM—2016–216632

Use of the NLPQLP Sequential Quadratic Programming Algorithm to Solve Rotorcraft Aeromechanical Constrained Optimisation Problems

Jane Anne Leyland
Ames Research Center
Moffett Field, California

April 2016

NASA STI Program ... in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

- E-mail your question to help@sti.nasa.gov

- Phone the NASA STI Information Desk at 757-864-9658

- Write to:
  NASA STI Information Desk
  Mail Stop 148
  NASA Langley Research Center
  Hampton, VA 23681-2199

NASA/TM—2016–216632

Use of the NLPQLP Sequential Quadratic Programming Algorithm to Solve Rotorcraft Aeromechanical Constrained Optimisation Problems

Jane Anne Leyland
Ames Research Center
Moffett Field, California

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, CA 94035-1000

April 2016

Available from:

NASA STI Support Services
Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199
757-864-9658

National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312
webmail@ntis.gov
703-605-6000

This report is also available in electronic form at

http://ntrs.nasa.gov

Table of Contents

Nomenclature v

Summary 1

1.0 Introduction 2

2.0 Technical 3

2.1 General Non-Linear Programming Problem 5

2.2 Specific Problems Solved as Part of This Research 6

2.2.1 (6 x 4) T-Matrix NLP Control Problems 7

2.2.2 (6 x 6) T-Matrix NLP Control Problems 10

2.2.3 (24 x 8) T-Matrix NLP Control Problems 12

2.2.4 (90 x 30) T-Matrix NLP Control Problems 14

2.2.5 (90 x 60) T-Matrix NLP Control Problems 17

2.3 Synthetic Data 20

2.3.1 Definition of the Random Number Generator Function RAN(SEED) 22

2.3.2 Determination of the Synthetic T-Matrix, the "Actual" Control θ_0 – Vector, and the "Actual" Measurement Z_A – Vector 23

2.4 The Regulator Problem 25

3.0 Results and Conclusions 27

4.0 References 29

Table of Contents (cont.)

Appendix A: NLPQLP: A Fortran Implementation of a Sequential Quadratic Programming Algorithm with Distributed and Non-Monotone Line Search – User's Guide, Version 3.1

Abstract A-1

1 Introduction A-2

2 Sequential Quadratic Programming Methods A-7

Algorithm 2.1 ..... A-10

Algorithm 2.2 ..... A-11

3 Performance Evaluation A-14

3.1 The Test Environment A-14

3.2 Testing Distributed Function Calls A-17

3.3 Function Evaluations and Gradient Approximations by a Difference Formulae Under Random Noise A-18

3.4 Testing Scaled Restarts A-19

4 Program Documentation A-22

5 Examples A-29

6 Conclusions A-35

References A-36

Separate Volumes:

Appendix B: Cases Run on the Hewlett-Packard Alpha Mainframe Computer

Appendix C: Cases Run on the Mac Pro Desktop Computer

Nomenclature

C_1	Input coefficient for the first term in the equation that defines $T_{q,p}$.
C_2	Input coefficient for the second term in the equation that defines $T_{q,p}$.
C_3	Input coefficient for the first term in the equation that defines $\theta_{p_{Initial}}$.
C_4	Input coefficient for the second term in the equation that defines $\theta_{p_{Initial}}$.
C_5	Input coefficient for the first term in the equation that defines ΔZ_{A_q}.
C_6	Input coefficient for the second term in the equation that defines ΔZ_{A_q}.
$g[Z(\theta)]$	Scalar performance index function that defines the performance index J. $g[Z(\theta)]$ is a function of the plant output measurement Z – vector, which is a function of the control θ – vector. In general, these functions can be non-linear.
I_Z	Set of all $q \ni Z_q \in Z$.
I_θ	Set of all $p \ni \theta_p \in \theta$.
IMSL	International Mathematics and Statistics Library, Inc.—a commercial collection of software libraries of numerical analysis functionality that are implemented in C, Java, C#.NET, and Fortran computer programming languages.
$ISEED1_k$	Input seed argument for the $k-th$ call to the random number generator function RAN$(\bullet)$ in the first term in the equation that defines $T_{q,p}$, and updated automatically on completion of the generation of the random number. This argument should initially be set to a large odd-integer value.
$ISEED2_k$	Input seed argument for the $k-th$ call to the random number generator function RAN$(\bullet)$ in the first term in the equation that defines $\theta_{p_{Initial}}$, and updated automatically on completion of the generation of the random number. This argument should initially be set to a large odd-integer value.

Nomenclature (cont.)

$ISEED3_k$	Input seed argument for the $k-th$ call to the random number generator function RAN$(\bullet)$ in the first term in the equation that defines ΔZ_{A_q}, and updated automatically on completion of the generation of the random number. This argument should initially be set to a large odd-integer value.
J	Scalar performance index that is defined by $g[Z(\theta)]$. In general, this function can be non-linear. For the problems considered in this research, J is a scalar performance index that is a quadratic function of the plant output measurement vector (i.e., the Z – vector). In this case, Z is a linear function of the control θ – vector, and J is a quadratic function of the control θ – vector.
$JSEED1_l$	Input seed argument for the $l-th$ call to the random number generator function RAN$(\bullet)$ in the second term in the equation that defines $T_{q,p}$, and updated automatically on completion of the generation of the random number. This argument should initially be set to a large odd-integer value.
$JSEED2_l$	Input seed argument for the $l-th$ call to the random number generator function RAN$(\bullet)$ in the second term in the equation that defines $\theta_{p_{Initial}}$, and updated automatically on completion of the generation of the random number. This argument should initially be set to a large odd-integer value.
$JSEED3_l$	Input seed argument for the $l-th$ call to the random number generator function RAN$(\bullet)$ in the second term in the equation that defines ΔZ_{A_q}, and updated automatically on completion of the generation of the random number. This argument should initially be set to a large odd-integer value.
k	Index number for the input seed argument for calls to the random number generator function RAN$(\bullet)$ in the first term in the equations that defines $T_{q,p}$, $\theta_{p_{Initial}}$, and ΔZ_{A_q}, where: $k \in [1, 2, 3, \bullet, \bullet, \bullet, \bullet, \rightarrow +\infty)$.
l	Index number for the input seed argument for calls to the random number generator function RAN$(\bullet)$ in the second term in the equations that defines $T_{q,p}$, $\theta_{p_{Initial}}$, and ΔZ_{A_q}, where: $l \in [1, 2, 3, \bullet, \bullet, \bullet, \bullet, \rightarrow +\infty)$.
N_{EQ}	Number of elements (dimension) in the Equality Constraint $\phi(\theta)$ –vector.

Nomenclature (cont.)

N_{IEQ}	Number of elements (dimension) in the Inequality Constraint $\psi(\theta)$ – vector. $N_{IEQ} = N_{IEQ_1} + N_{IEQ_2}$.
N_{IEQ_1}	Number of elements in the First Inequality Constraint ${}^1\psi(\theta)$ – sub-vector.
N_{IEQ_2}	Number of elements in the Second Inequality Constraint ${}^2\psi(\theta)$ – sub-vector.
N_Z	Number of elements (dimension) in the predicted measurement Z – vector.
N_θ	Number of elements (dimension) in the control θ – vector.
NLP	Non-Linear Programming algorithm.
NLPQLP	Non-Linear Programming (NLP) algorithm that employs the Sequential Quadratic Programming (SQP) algorithm as its core algorithm.
p	Index number for the control θ – vector elements.
q	Index number for the predicted measurement Z – vector elements.
r	Index number for the Equality Constraint $\phi(\theta)$ – vector elements.
$RAN(\bullet)$	Uniform Random Number Distribution Function that yields a uniformly random real number $\in [0, 1)$.
s	Index number for the Inequality Constraint $\psi(\theta)$ – vector elements.
SQP	Sequential Quadratic Programming (SQP) algorithm.
T	System or transfer $\left(N_Z \text{ x } N_\theta\right)$ matrix either defined by direct input or synthetically determined.
$T_{q,p}$	The $(q, p) - th$ element of the system or transfer $\left(N_Z \text{ x } N_\theta\right)$ matrix.
W_Z	Diagonal $\left(N_Z \text{ x } N_Z\right)$ weighting matrix in the performance index. The default setting that is the identity matrix; can be redefined by input.

Nomenclature (cont.)

W_θ	Diagonal $\left(\mathrm{N}_\theta \text{ x } \mathrm{N}_\theta\right)$ weighting matrix in the θ term of the regulator performance index J. The default setting that is the null matrix; can be redefined by input.
$W_{\dot{\theta}}$	Diagonal $\left(\mathrm{N}_\theta \text{ x } \mathrm{N}_\theta\right)$ weighting matrix in the $\dot{\theta}$ term of the regulator performance index J. The default setting that is the null matrix; can be redefined by input.
Z	Equals $Z(\theta)$ and is used when the omission of the explicit dependence on the control θ – vector does not present any confusion, ambiguity, or vagueness.
$Z(\theta)$	Predicted measurement Z – vector $\left(\mathrm{N}_Z \text{ x } 1\right)$ evaluated during the optimisation or regulator process and is a function of the control θ – vector. In general, this function can be non-linear. For the problems analysed in this research, $Z(\theta)$ is a linear function of the control θ – vector.
Z_A	Actual measurement Z – vector $\left(\mathrm{N}_Z \text{ x } 1\right)$ that would normally be evaluated during the previous duty cycle or at a reference epoch time. Z_A is either directly input or synthetically determined.
Z_q	The $q-th$ element of the predicted measurement Z – vector.
ΔZ_A	Random component of the synthetically determined actual measurement Z_A – vector.
ΔZ_{A_q}	Random component of the $q-th$ element of the synthetically determined actual measurement Z_A – vector.
ε	Input small value constant selected to prevent $\theta_{0_{Initial}}$ from being identical on a bound (i.e., the least upper bound [l.u.b.] or the greatest lower bound [g.l.b.]).
θ	Control θ – vector $\left(\mathrm{N}_\theta \text{ x } 1\right)$.
$\dot{\theta}$	Time rate of change of the control θ – vector $\left(\mathrm{N}_\theta \text{ x } 1\right)$.
θ_0	Solution θ_{Sol} – vector $\left(\mathrm{N}_\theta \text{ x } 1\right)$ that would normally be evaluated during the previous duty cycle or at a reference epoch time. θ_0 is either directly input or synthetically determined.

Nomenclature (cont.)

θ_{MAX_p} Least upper bound (l.u.b.) for the $p-th$ element of the control θ – vector $\left(\text{N}_\theta \text{ x } 1\right)$.

θ_{MIN_p} Greatest lower bound (g.l.b.) for the $p-th$ element of the control θ – vector $\left(\text{N}_\theta \text{ x } 1\right)$.

θ_p The $p-th$ element of the control θ – vector $\left(\text{N}_\theta \text{ x } 1\right)$.

$\dot{\theta}_p$ The $p-th$ element of the time rate of change of the control θ – vector $\left(\text{N}_\theta \text{ x } 1\right)$.

$\theta_{p_{Initial}}$ The $p-th$ element of the solution θ_{Sol} – vector $\left(\text{N}_\theta \text{ x } 1\right)$ that would normally be evaluated during the previous duty cycle or at a reference epoch time. θ_0 is synthetically determined.

θ_{Sol} Solution θ_{Sol} – vector $\left(\text{N}_\theta \text{ x } 1\right)$ evaluated during the optimisation or regulator process.

θ_{Sol_p} The $p-th$ element of the solution control θ_{Sol} – vector $\left(\text{N}_\theta \text{ x } 1\right)$ evaluated during the optimisation or regulator process.

$\phi(\theta)$ Equality Constraint $\phi(\theta)$ –vector $\left(\text{N}_{\text{EQ}} \text{ x } 1\right)$ function; in general, can be dependent on the θ – vector and be non-linear.

$\phi_r(\theta)$ The $r-th$ element of the Equality Constraint $\phi(\theta)$ –vector $\left(\text{N}_{\text{EQ}} \text{ x } 1\right)$ function.

$\psi(\theta)$ Complete Inequality Constraint $\psi(\theta)$ –vector $\left(\left[\text{N}_{\text{IEQ}} = \text{N}_{\text{IEQ}_1} + \text{N}_{\text{IEQ}_2}\right] \text{ x } 1\right)$ function; in general, can be dependent on the θ – vector and be non-linear. $\psi(\theta)$ is comprised of the two sub-vector functions ${}^1\psi(\theta)$ and ${}^2\psi(\theta)$. Specifically:

$$\psi(\theta) = \begin{bmatrix} {}^1\psi(\theta) \\ {}^2\psi(\theta) \end{bmatrix}$$

Nomenclature (concluded)

$^{1}\psi(\theta)$	First Inequality Constraint $^{1}\psi(\theta)-$sub-vector $\left(\mathrm{N}_{\mathrm{IEQ}_1}\ \mathrm{x}\ 1\right)$ function with elements of the First Inequality Constraint Form.
$^{2}\psi(\theta)$	Second Inequality Constraint $^{2}\psi(\theta)-$sub-vector $\left(\mathrm{N}_{\mathrm{IEQ}_2}\ \mathrm{x}\ 1\right)$ function with elements of the Second Inequality Constraint Form.
$\psi_s(\theta)$	The $s-th$ element of the complete Inequality Constraint vector $\left(\mathrm{N}_{\mathrm{IEQ}}\ \mathrm{x}\ 1\right)$ $\psi(\theta)-$function.

Summary

Optimisation of a control vector, an aerodynamic surface design, or an aircraft configuration potentially offers significant performance enhancement to rotorcraft systems. These problems typically include various types of constraints. Previous research and analysis indicated that non-linear programming methods that solve a sequence of related quadratic-programming sub-problems could be used successfully to solve these problems. Accordingly, a licence for one of the latest versions of Professor Klaus Schittkowski's very successful Sequential Quadratic Programming NLPQLP software was obtained and used to experiment with and analyse typical optimisation problems encountered in various rotorcraft wind tunnel and flight tests. Emphasis was directed toward obtaining efficiency, robustness, and speed in computation.

The NLPQLP software was installed on both a mainframe computer and a desktop computer. Stand-alone main driver codes were developed to task the NLPQLP software to solve non-linear programming (NLP) problems. The required input data was synthesised to facilitate this analysis. Solution to the classic regulator problem was included to provide verification of the NLP solutions to the unconstrained optimisation problems. The mainframe computer was used to develop the driver codes and to experiment with the various models and tune the associated input. The desktop computer was used to refine the process and to develop software for laptop computers that can be used in austere test environments including wind tunnels.

The problems solved with this analysis were of the type encountered in rotorcraft applications where there is a linear dependence (i.e., a T-Matrix plant model) of the measurement vector on the control vector. These problems ranged from a relatively simple, unconstrained 4-vector control to a relatively large, constrained 60-vector control problem. Five different control vector dimensions ranging from 4 to 60 were analysed, and solutions were obtained for each of these with no imposition of constraints, imposition of only equality constraints, imposition of only inequality constraints, and imposition of both equality and inequality constraints. The smaller 4-, 6-, and 8-dimension control vector problems are representative of actual rotorcraft control problems. The solutions to these problems were sufficiently fast to be included in real-time duty cycles. The larger 30- and 60-dimension control vector problems are representative of aerodynamic surface design or aircraft configuration problems and, although they were solved rapidly, they are more suitable to non-real-time design applications. Solutions were obtained for all problems considered, and verification of the solutions to all of the unconstrained problems was obtained by solving the regulator problem. Although tuning and some input adjustments were required to successfully solve the large constrained 60-vector control problems, the NLPQLP System proved to be an efficient and reliable method to solve these problems.

1.0 Introduction

Previous research on the feasibility and desirability of using a constrained optimisation technique to define the optimal control vector and plant model for rotorcraft was accomplished on a mainframe computer not part of actual wind tunnel and/or flight-test experiments. This research indicated that constrained optimisation methodology provides better controller performance in many cases compared to results obtained with the widely used solution to the regulator problem for this application (ref. 1), and would be useful in defining the optimal configuration and required constants for a non-linear neural-network plant model (refs. 2–5). The initial research (ref. 1) on the development, design, and use of optimal controllers, both open- and closed-loop, to optimise rotorcraft aeromechanical behaviour indicated that the general non-linear programming method coded as the NCONF/DNCONF subroutine system, which is available in the IMSL MATH/LIBRARY (ref. 6), worked very well for solving the required constrained optimisation problems and was significantly superior to methods previously used for solving problems of this type. This IMSL NCONF/DNCONF subroutine system, which dates back to 1989, was used successfully on subsequent research studies (refs. 2–5) and likewise worked very well for solving the required constrained optimisation problems.

This IMSL NCONF/DNCONF subroutine system is based on the work by Schittkowski, Gill et al., Powell, and Stoer (refs. 7–13). This method solves the general non-linear programming problem by solving a sequence of related quadratic programming sub-problems. One advantage of this method is that quadratic programming problems can be solved efficiently. A very important property of quadratic programming problems is that if the quadratic coefficient matrix in the performance index is positive definite, the problem has a unique solution, which is, of course, the global solution. This means that the sequence of solutions to the quadratic programming sub-problems will converge to the global solution of the general problem in the limit providing that the quadratic coefficient matrix in the performance index remains positive definite in the process.

To conduct a wind tunnel experiment, the optimisation code is installed on a computer within the wind tunnel or on a portable computer such as a laptop that could be brought into the wind tunnel. Research on available suitable optimisation codes revealed that Professor Klaus Schittkowski had revised and updated his sequential quadratic programming method, which is part of the IMSL MATH/LIBRARY, via several versions of the code that improved performance and eliminated errors, and that it was available by licence as a stand-alone code (i.e., not part of a library). NASA obtained a licence, and Version 3.1, dated February 2010, was installed on both the Mac Pro desktop computer and the Hewlett-Packard Alpha mainframe computer. The code that was installed on the Mac Pro desktop computer was transportable to a Mac laptop computer for use in the wind tunnel. Version 3.1 of the NLPQLP System (ref. 14 and Appendix A) was used for the study described herein.

The linear global plant model, which linearly relates the measurement vector to the control vector and is widely used for rotorcraft aeromechanical behaviour studies, was assumed for the problems that were solved as part of this research. For this research, two options could be used to define the T-Matrix: (1) use actual test data to define these elements, or (2) synthetically

define these elements using a uniformly distributed random function. The second option was used for the results presented herein.

This method was computationally implemented by writing stand-alone main driver codes assuming the linear global plant model. These drivers included the widely used solution to the regulator problem, used for comparison purposes. These codes were written to solve several typical problems of rotorcraft aeromechanical behaviour and are specific to the dimension of the T-Matrix (i.e., the dimensions of the measurement vector and the control vector) and the computer (i.e., the Hewlett-Packard Alpha mainframe or the Mac Pro desktop) that the problem was run on.

The very general non-linear programming problem that is solved by the NLPQLP System (ref. 14) is defined in section 2.1. The specific problems that were solved using the NLPQLP System for this study are defined in section 2.2. A description of the synthetic data generation process is presented in section 2.3, and the definition of, and solution to, the regulator problem are described in section 2.4.

The (6 x 4), (6 x 6), and (24 x 8) T-Matrix non-linear programming (NLP) control problems are representative of actual rotorcraft control problems. The solutions to these problems were sufficiently fast to be included in real-time duty cycles. The (90 x 30) and (90 x 60) T-Matrix NLP problems are representative of aerodynamic surface design and/or aircraft configuration problems and, although they were solved rapidly, they are more suitable to non-real-time design applications.

The results are shown in Appendix B and Appendix C, separate volumes of this report.

Listings of the command (DCL) file code and the Fortran main driver code for the Hewlett-Packard Alpha mainframe computer, and the input and output for the four sub-problems that were part of the (6 x 4), (6 x 6), (24 x 8), (90 x 30), and (90 x 60) T-Matrix NLP control problems solved using the Hewlett-Packard Alpha mainframe computer, are presented in Appendix B. A listing of the Fortran main driver code for the Mac Pro desktop computer, and the input and output for the four sub-problems that were part of the (6 x 4), (6 x 6), (24 x 8), (90 x 30), and (90 x 60) T-Matrix NLP control problems solved using the Mac Pro desktop computer, are presented in Appendix C.

2.0 Technical

This study documents the testing, experimentation, and evaluation of Version 3.1 of the NLPQLP System (ref. 14 and Appendix A) to ascertain its suitability to solve rotorcraft optimisation problems. This NLPQLP System is designed to solve a very general NLP problem by solving a sequence of related quadratic programming sub-problems. One advantage of this method is that quadratic programming problems can be solved efficiently. A very important property of quadratic programming problems is that if the quadratic coefficient matrix in the

performance index is positive definite, then the problem has a unique solution, which is, of course, the global solution. This means that the sequence of solutions to the quadratic programming sub-problems will converge to the global solution of the general problem in the limit providing that the quadratic coefficient matrix in the performance index remains positive definite in the process.

This very general NLP problem is described in section 2.1. The problems solved in this analysis are encountered in various rotorcraft applications where there is a linear dependence (i.e., a T-Matrix plant model) of the measurement vector (the measurement Z – vector) on the control vector (i.e., the control θ – vector). The general form of the T-Matrix NLP problems considered in this study is referred to herein as the General T-Matrix NLP control problem, and is defined in section 2.2. Specific (6 x 4) T-Matrix NLP control problems (see section 2.2.1), (6 x 6) T-Matrix NLP control problems (see section 2.2.2), (24 x 8) T-Matrix NLP control problems (see section 2.2.3), (90 x 30) T-Matrix NLP control problems (see section 2.2.4), and (90 x 60) T-Matrix NLP control problems (see section 2.2.5) were analysed and solved. Each of these problems had four sub-problems: (1) unconstrained optimisation, (2) optimisation with only equality constraints, (3) optimisation with only inequality constraints, and (4) optimisation with both equality constraints and inequality constraints.

In an actual wind tunnel or flight test experiment or optimisation application, the required input would be the actual test data, which perhaps might be reformatted for computer compatibly. To expedite the testing, experimentation, and evaluation of the NLPQLP System, the input data was synthetically determined. This synthesis process is described in section 2.3 and its sub-sections.

Finally, the classic regulator problem is solved as an appendix to the unconstrained optimisation NLP problems to provide a means to verify their solutions. The classic regulator problem is defined in section 2.4.

In this study the NLPQLP System was coded in Fortran 77 and installed on both a Hewlett-Packard Alpha Server GS1280 mainframe computer with the Open VMS Version 8.2 Operating System and a Mac Pro desktop computer with the Mac OS X, Version 10.5.8 Operating System. The VMS FORTRAN Compiler was used to compile the code on the mainframe computer, and the G95 FORTRAN Compiler was used to compile the code on the Mac Pro desktop computer. It is noted that these codes will, in general, produce different numerical results for these computer systems when synthetic input data is generated because the random number generator function algorithms used for data synthesis are different for these computer systems. The codes for both the Hewlett-Packard Alpha mainframe computer and the Mac Pro desktop computer are stand-alone codes and do not require any special software libraries. Accordingly, the associated software and codes installed on the Mac Pro desktop computer should be transportable to a Mac laptop computer for use in the wind tunnel.

2.1 General Non-Linear Programming Problem

The general optimisation problem that can be solved with the NLPQLP System is:

Determine the θ_{Sol} *– vector that solves the problem:*

$$\underset{\theta_p \in \theta}{Minimise} \quad J = g\left[Z(\theta)\right] \qquad for \quad p \in I_\theta$$

$$where \qquad Z = Z(\theta) = \left[\bullet \ \bullet \ \bullet \ \left\{Z_q \,\middle|\, q \in I_Z\right\} \ \bullet \ \bullet \ \bullet\right]^T$$

$$and \qquad \theta = \left[\bullet \ \bullet \ \bullet \ \left\{\theta_p \,\middle|\, p \in I_\theta\right\} \ \bullet \ \bullet \ \bullet\right]^T$$

$$I_Z = \left\{\bullet \ \bullet \ \bullet \ \left\{\forall \ q \ \ni \ Z_q \in Z\right\} \ \bullet \ \bullet \ \bullet\right\}$$

$$I_\theta = \left\{\bullet \ \bullet \ \bullet \ \left\{\forall \ p \ \ni \ \theta_p \in \theta\right\} \ \bullet \ \bullet \ \bullet\right\}$$

Subject to:

$$\left.\begin{array}{l} \theta_{\mathrm{MIN}_p} \le \theta_p \le \theta_{\mathrm{MAX}_p} \\ \theta_{\mathrm{MIN}_p} \in (-\infty, +\infty) \\ \theta_{\mathrm{MAX}_p} \in (-\infty, +\infty) \end{array}\right\} \left\{\begin{array}{l} \text{Direct Constraints on the Control } \theta - \text{vector} \\ \text{Elements } \theta_p \text{ for: } \quad p \in I_\theta \end{array}\right.$$

$$\phi(\theta) = 0 \quad \left.\right\} \left\{\begin{array}{l} \text{General Equality Constraint Vector Function} \\ \text{with Dimension } N_{EQ} \end{array}\right.$$

$$\psi(\theta) \ge 0 \quad \left.\right\} \left\{\begin{array}{l} \text{General Inequality Constraint Vector Function} \\ \text{with Dimension } N_{IEQ} \end{array}\right.$$

2.2 Specific Problems Solved as Part of This Research

The specific problems that were solved during this study have a general form (i.e., the General T-Matrix NLP control problem) that is a specific application of the general non-linear programming problem defined in section 2.1. In this case, the general $g[Z(\theta)]$ scalar performance index function is replaced with a quadratic function of a T-Matrix plant model. The T-Matrix itself is a linear plant model that relates the measurement Z – vector to the control θ – vector. The performance index, J, is a quadratic function of the measurement Z – vector. Limits as per the general non-linear programming problem are imposed on the elements of the control θ – vector. Both the equality $\phi(\theta)$ and inequality $\psi(\theta)$ constraint functions are non-linear functions of the elements of the control θ – vector. The elements of the equality $\phi(\theta)$ –vector constraint function are pseudo-generalisations of the elements of a vector cross product. The Inequality Constraint $\psi(\theta)$ –vector function includes two forms of inequality constraints: (1) the First Inequality Constraint ${}^{1}\psi(\theta)$ –vector sub-vector function is comprised of elements that are pseudo-generalisations of an amplitude constraint on a matched harmonic pair (i.e., the sine and cosine elements of a specific harmonic control signal) of control elements, and (2) the Second Inequality Constraint ${}^{2}\psi(\theta)$ –vector sub-vector function is comprised of elements that are pseudo-generalisations of a rate-of-change constraint on an element of the control vector. Each of these problems has four sub-problems that were analysed and solved. These sub-problems are: (1) unconstrained optimisation, (2) optimisation with only equality constraints, (3) optimisation with only inequality constraints, and (4) optimisation with both equality constraints and inequality constraints. This General T-Matrix NLP control problem with both equality and inequality constraints is defined as:

Determine the θ_{Sol} *– vector that solves the problem:*

$$\underset{\theta_p \in \theta}{Minimise} \qquad J = Z^{\mathsf{T}} W_Z Z \qquad for \quad p \in I_\theta$$

$$where \qquad Z = Z(\theta) = Z_A + \mathsf{T}(\theta - \theta_0)$$

$$Z = Z(\theta) = \left[\bullet \ \bullet \ \bullet \ \left\{ Z_q \,\middle|\, q \in I_Z \right\} \ \bullet \ \bullet \ \bullet \right]^{\mathsf{T}}$$

$$and \qquad \theta = \left[\bullet \ \bullet \ \bullet \ \left\{ \theta_p \,\middle|\, p \in I_\theta \right\} \ \bullet \ \bullet \ \bullet \right]^{\mathsf{T}}$$

$$I_Z = \left\{ \bullet \ \bullet \ \bullet \ \left\{ \forall \ q \ \ni \ Z_q \in Z \right\} \ \bullet \ \bullet \ \bullet \right\}$$

$$I_\theta = \left\{ \bullet \ \bullet \ \bullet \ \left\{ \forall \ p \ \ni \ \theta_p \in \theta \right\} \ \bullet \ \bullet \ \bullet \right\}$$

Subject to:

$$\left.\begin{array}{l} \theta_{\mathrm{MIN}p} \le \theta_p \le \theta_{\mathrm{MAX}p} \\ \theta_{\mathrm{MIN}p} \in (-\infty, +\infty) \\ \theta_{\mathrm{MAX}p} \in (-\infty, +\infty) \end{array}\right\} \left\{\begin{array}{l} \text{Direct Constraints on the Control } \theta\text{ – vector} \\ \text{Elements } \theta_p \text{ for: } \quad p \in I_\theta \end{array}\right.$$

$$\phi_r(\theta) = \theta_i \theta_l - \theta_j \theta_k = 0 \quad \left\{\begin{array}{l} \text{Form of the Equality Constraint } \phi(\theta) \text{ – vector} \\ \text{Elements } \phi_r(\theta) \text{ for: } r \in [1, \mathrm{N_{EQ}}] \text{ and} \\ i, j, k, l \in I_\theta \quad \text{and} \quad i < j < k < l \end{array}\right.$$

$$\psi_s(\theta) = \psi_{s_{\mathrm{MAX}}} - \sqrt{\theta_i^2 + \theta_j^2} \ge 0 \quad \left\{\begin{array}{l} \text{Form of the First Inequality Constraint} \\ {}^1\psi(\theta) \text{ – sub-vector Elements } \psi_s(\theta) \text{ for:} \\ s \in [1, \mathrm{N_{IEQ_1}}], \ i, j \in I_\theta \text{ and } i < j \end{array}\right.$$

$$\psi_s(\theta) = \psi_{s_{\mathrm{MAX}}} - \mathrm{Abs}(\theta_i - \theta_{0_i}) \ge 0 \quad \left\{\begin{array}{l} \text{Form of the Second Inequality Constraint} \\ {}^2\psi(\theta) \text{ – sub-vector Elements } \psi_s(\theta) \text{ for:} \\ s \in [(\mathrm{N_{IEQ_1}} + 1), (\mathrm{N_{IEQ_1}} + \mathrm{N_{IEQ_2}})] \\ \text{and} \quad i \in I_\theta \end{array}\right.$$

2.2.1 (6 x 4) T-Matrix NLP Control Problems

For the (6 x 4) T-Matrix NLP control problem, the number of elements (dimension) in the control θ – vector is four (i.e., $\mathrm{N}_\theta = 4$), and the number of elements (dimension) in the predicted measurement Z – vector is six (i.e., $\mathrm{N}_Z = 6$). Correspondingly, the system or transfer T-Matrix is (6 x 4). The number of elements (dimension) in the Equality Constraint $\phi(\theta)$ – vector is one (i.e., $\mathrm{N_{EQ}} = 1$), the number of elements in the First Inequality Constraint ${}^1\psi(\theta)$ – sub-vector function is two (i.e., $\mathrm{N_{IEQ_1}} = 2$), the number of elements in the Second Inequality Constraint ${}^2\psi(\theta)$ – sub-vector function is four (i.e., $\mathrm{N_{IEQ_2}} = 4$), and the dimension of the Inequality Constraint $\psi(\theta)$ – vector is six (i.e., $\mathrm{N_{IEQ}} = \mathrm{N_{IEQ_1}} + \mathrm{N_{IEQ_2}} = 6$). The (6 x 4) T-Matrix NLP control problem with all constraints is defined as:

Determine the θ – vector, θ_{Sol}, that solves the problem:

$$\underset{\theta_p \in \theta}{\text{Minimise}} \qquad J = Z^{\mathsf{T}} W_Z Z \qquad \text{for} \quad p \in I_\theta$$

where $$Z = Z(\theta) = Z_A + \mathsf{T}(\theta - \theta_0)$$

$$Z = Z(\theta) = \left[\bullet \;\; \bullet \;\; \bullet \;\; \left\{ Z_q \,\middle|\, q \in I_Z \right\} \;\; \bullet \;\; \bullet \;\; \bullet \right]^{\mathsf{T}}$$

and $$\theta = \left[\bullet \;\; \bullet \;\; \bullet \;\; \left\{ \theta_p \,\middle|\, p \in I_\theta \right\} \;\; \bullet \;\; \bullet \;\; \bullet \right]^{\mathsf{T}}$$

$$I_Z = \{1,\ 2,\ 3,\ 4,\ 5,\ 6\} \quad \text{where} \quad N_Z = 6$$

$$I_\theta = \{1,\ 2,\ 3,\ 4\} \quad \text{where} \quad N_\theta = 4$$

then $$Z = Z(\theta) = \left[Z_1,\ Z_2,\ Z_3,\ Z_4,\ Z_5,\ Z_6 \right]^{\mathsf{T}}$$

$$\theta = \left[\theta_1,\ \theta_2,\ \theta_3,\ \theta_4 \right]^{\mathsf{T}}$$

Subject to:

$$\left.\begin{array}{l} \theta_{\mathrm{MIN}_p} \le \theta_p \le \theta_{\mathrm{MAX}_p} \\ \theta_{\mathrm{MIN}_p} \in (-\infty, +\infty) \\ \theta_{\mathrm{MAX}_p} \in (-\infty, +\infty) \end{array}\right\} \left\{\begin{array}{l} \text{Direct Constraints on the Control} \\ \theta \text{ – vector Elements } \theta_p \text{ for:} \quad p \in I_\theta \end{array}\right.$$

$$\left[\phi_1(\theta) = \theta_1\theta_4 - \theta_2\theta_3 = 0 \right] \left\{\begin{array}{l} \text{Equality Constraint } \phi(\theta) \text{ – vector for:} \\ N_{EQ} = 1 \end{array}\right.$$

$$\left.\begin{array}{l} \left[\psi_1(\theta) = \psi_{1_{MAX}} - \sqrt{\theta_1^2 + \theta_2^2} \ge 0 \right] \\ \left[\psi_2(\theta) = \psi_{2_{MAX}} - \sqrt{\theta_3^2 + \theta_4^2} \ge 0 \right] \end{array}\right\} \left\{\begin{array}{l} \text{First Inequality Constraint} \\ {}^1\psi(\theta) \text{ – sub - vector for:} \quad N_{IEQ_1} = 2 \end{array}\right.$$

$$\left.\begin{bmatrix} \psi_3(\theta) = \psi_{3_{MAX}} - \left|\theta_1 - \theta_{0_1}\right| \geq 0 \\ \psi_4(\theta) = \psi_{4_{MAX}} - \left|\theta_2 - \theta_{0_2}\right| \geq 0 \\ \psi_5(\theta) = \psi_{5_{MAX}} - \left|\theta_3 - \theta_{0_3}\right| \geq 0 \\ \psi_6(\theta) = \psi_{6_{MAX}} - \left|\theta_4 - \theta_{0_4}\right| \geq 0 \end{bmatrix}\right\} \begin{cases} \text{Second Inequality Constraint} \\ {}^2\psi(\theta) - \text{sub-vector for:} \quad N_{IEQ_2} = 4 \end{cases}$$

The (6 x 4) T-Matrix NLP control problem has four sub-problems: (1) unconstrained optimisation, (2) optimisation with only equality constraints, (3) optimisation with only inequality constraints, and (4) optimisation with both equality constraints and inequality constraints. It was assumed that the tasking of these T-Matrix NLP control problems occurred within the framework of real-time controller duty cycles. To expedite the testing, experimentation, and evaluation of the NLPQLP System, the required input data was synthetically determined using a process designed expressly for this analysis. Specifically, a previously identified T-Matrix and an actual control θ_0 – vector/actual measurement Z_A – vector pair from a previous duty cycle were synthesised for use as input to these problems. Additionally, these already synthesised values were directly input to the (6 x 4) T-Matrix NLP control problems for comparison purposes. The classic regulator problem was solved to verify the NLP solutions to the unconstrained optimisation NLP problems. Agreement was obtained in all cases.

Listings of the command (DCL) file code and the Fortran main driver code for the Hewlett-Packard Alpha mainframe computer, and the input and output for the four sub-problems that were part of the (6 x 4) T-Matrix NLP control problems solved using the Hewlett-Packard Alpha mainframe computer, are presented in Appendix B, section B.1. A listing of the Fortran main driver code for the Mac Pro desktop computer, and the input and output for the four sub-problems that were part of the (6 x 4) T-Matrix NLP control problems solved using the Mac Pro desktop computer, are presented in Appendix C, section C.1.

These (6 x 4) T-Matrix NLP control problems are representative of actual rotorcraft control problems. The solutions to these problems were sufficiently fast to be included in real-time duty cycles.

2.2.2 (6 x 6) T-Matrix NLP Control Problems

For the (6 x 6) T-Matrix NLP control problem, the number of elements (dimension) in the control θ – vector is six (i.e., N_θ = 6), and the number of elements (dimension) in the predicted measurement Z – vector is also six (i.e., N_Z = 6). Correspondingly, the system or transfer T-Matrix is (6 x 6). The number of elements (dimension) in the Equality Constraint $\phi(\theta)$ –vector is one (i.e., N_{EQ} = 1), the number of elements in the First Inequality Constraint ${}^1\psi(\theta)$ –sub-vector is three (i.e., N_{IEQ_1} = 3), the number of elements in the Second Inequality Constraint ${}^2\psi(\theta)$ –sub-vector function is six (i.e., N_{IEQ_2} = 6), and the dimension of the Inequality Constraint $\psi(\theta)$ –vector is nine (i.e., $N_{IEQ} = N_{IEQ_1} + N_{IEQ_2}$ = 9). The (6 x 6) T-Matrix NLP control problem with all constraints is defined as:

Determine the θ – vector, θ_{Sol}, that solves the problem:

$$\underset{\theta_p \in \theta}{Minimise} \qquad J = Z^T W_Z Z \qquad for \quad p \in I_\theta$$

where $$Z = Z(\theta) = Z_A + \mathsf{T}(\theta - \theta_0)$$

$$Z = Z(\theta) = \left[\bullet \ \bullet \ \bullet \ \left\{ Z_q \,\middle|\, q \in I_Z \right\} \ \bullet \ \bullet \ \bullet \right]^T$$

and $$\theta = \left[\bullet \ \bullet \ \bullet \ \left\{ \theta_p \,\middle|\, p \in I_\theta \right\} \ \bullet \ \bullet \ \bullet \right]^T$$

$$I_Z = \{1,\ 2,\ 3,\ 4,\ 5,\ 6\} \quad \text{where} \quad N_Z = 6$$

$$I_\theta = \{1,\ 2,\ 3,\ 4,\ 5,\ 6\} \quad \text{where} \quad N_\theta = 6$$

then $$Z = Z(\theta) = \left[Z_1,\ Z_2,\ Z_3,\ Z_4,\ Z_5,\ Z_6 \right]^T$$

$$\theta = \left[\theta_1,\ \theta_2,\ \theta_3,\ \theta_4,\ \theta_5,\ \theta_6 \right]^T$$

Subject to:

$$\left.\begin{array}{l} \theta_{\text{MIN}_p} \le \theta_p \le \theta_{\text{MAX}_p} \\ \theta_{\text{MIN}_p} \in (-\infty, +\infty) \\ \theta_{\text{MAX}_p} \in (-\infty, +\infty) \end{array}\right\} \begin{array}{l} \text{Direct Constraints on the Control} \\ \theta - \text{vector Elements } \theta_p \text{ for: } \quad p \in I_\theta \end{array}$$

$$\left.\left[\phi_1(\theta) = \theta_1\theta_4 - \theta_2\theta_3 = 0\right]\right\} \begin{array}{l} \text{Equality Constraint } \phi(\theta) - \text{vector for:} \\ \text{N}_{\text{EQ}} = 1 \end{array}$$

$$\left.\left[\begin{array}{lllll} \psi_1(\theta) & = & \psi_{1_{\text{MAX}}} & - \sqrt{\theta_1^2 + \theta_2^2} & \ge 0 \\ \psi_2(\theta) & = & \psi_{2_{\text{MAX}}} & - \sqrt{\theta_3^2 + \theta_4^2} & \ge 0 \\ \psi_3(\theta) & = & \psi_{3_{\text{MAX}}} & - \sqrt{\theta_5^2 + \theta_6^2} & \ge 0 \end{array}\right]\right\} \begin{array}{l} \text{First Inequality Constraint} \\ {}^1\psi(\theta) - \text{sub-vector for:} \quad \text{N}_{\text{IEQ}_1} = 3 \end{array}$$

$$\left.\left[\begin{array}{lllll} \psi_4(\theta) & = & \psi_{4_{\text{MAX}}} & - \left|\theta_1 - \theta_{0_1}\right| & \ge 0 \\ \psi_5(\theta) & = & \psi_{5_{\text{MAX}}} & - \left|\theta_2 - \theta_{0_2}\right| & \ge 0 \\ \psi_6(\theta) & = & \psi_{6_{\text{MAX}}} & - \left|\theta_3 - \theta_{0_3}\right| & \ge 0 \\ \psi_7(\theta) & = & \psi_{7_{\text{MAX}}} & - \left|\theta_4 - \theta_{0_4}\right| & \ge 0 \\ \psi_8(\theta) & = & \psi_{8_{\text{MAX}}} & - \left|\theta_5 - \theta_{0_5}\right| & \ge 0 \\ \psi_9(\theta) & = & \psi_{9_{\text{MAX}}} & - \left|\theta_6 - \theta_{0_6}\right| & \ge 0 \end{array}\right]\right\} \begin{array}{l} \text{Second Inequality Constraint} \\ {}^2\psi(\theta) - \text{sub-vector for:} \quad \text{N}_{\text{IEQ}_2} = 6 \end{array}$$

The (6 x 6) T-Matrix NLP control problem has four sub-problems: (1) unconstrained optimisation, (2) optimisation with only equality constraints, (3) optimisation with only inequality constraints, and (4) optimisation with both equality constraints and inequality constraints. It was assumed that the tasking of these T-Matrix NLP control problems occurred within the framework of real-time controller duty cycles. To expedite the testing, experimentation, and evaluation of the NLPQLP System, the required input data was synthetically determined using a process designed expressly for this analysis. Specifically, a previously identified T-Matrix and an actual control θ_0 – vector/actual measurement Z_A – vector pair from a previous duty cycle were synthesised for use as input to these problems. Additionally, these already synthesised values were directly input to the (6 x 6) T-Matrix NLP control problems for comparison purposes. The classic regulator problem was solved to verify the NLP solutions to the unconstrained optimisation NLP problems. Agreement was obtained in all cases.

Listings of the command (DCL) file code and the Fortran main driver code for the Hewlett-Packard Alpha mainframe computer, and the input and output for the four sub-problems that were part of the (6 x 6) T-Matrix NLP control problems solved using the Hewlett-Packard Alpha mainframe computer, are presented in Appendix B.2. A listing of the Fortran main driver code for the Mac Pro desktop computer, and the input and output for the four sub-problems that were part of the (6 x 6) T-Matrix NLP control problems solved using the Mac Pro desktop computer, are presented in Appendix C.2.

These (6 x 6) T-Matrix NLP control problems are representative of actual rotorcraft control problems. The solutions to these problems were sufficiently fast to be included in real-time duty cycles.

2.2.3 (24 x 8) T-Matrix NLP Control Problems

For the (24 x 8) T-Matrix NLP control problem, the number of elements (dimension) in the control θ – vector is 8 (i.e., $N_\theta = 8$), and the number of elements (dimension) in the predicted measurement Z – vector is 24 (i.e., $N_Z = 24$). Correspondingly, the system or transfer T-Matrix is (24 x 8). The number of elements (dimension) in the Equality Constraint $\phi(\theta)$ – vector is 2 (i.e., $N_{EQ} = 2$), the number of elements in the First Inequality Constraint ${}^1\psi(\theta)$ – sub-vector is 4 (i.e., $N_{IEQ_1} = 4$), the number of elements in the Second Inequality Constraint ${}^2\psi(\theta)$ – sub-vector function is 8 (i.e., $N_{IEQ_2} = 8$), and the dimension of the Inequality Constraint $\psi(\theta)$ – vector is 12 (i.e., $N_{IEQ} = N_{IEQ_1} + N_{IEQ_2} = 12$). The (24 x 8) T-Matrix NLP control problem with all constraints is defined as:

Determine the θ – vector, θ_{Sol}, that solves the problem:

$$\underset{\theta_p \in \theta}{Minimise} \qquad J = Z^T W_Z Z \qquad for \quad p \in I_\theta$$

$$where \qquad Z = Z(\theta) = Z_A + \mathsf{T}(\theta - \theta_0)$$

$$Z = Z(\theta) = \left[\bullet \ \bullet \ \bullet \ \left\{ Z_q \,\middle|\, q \in I_Z \right\} \ \bullet \ \bullet \ \bullet \right]^T$$

$$and \qquad \theta = \left[\bullet \ \bullet \ \bullet \ \left\{ \theta_p \,\middle|\, p \in I_\theta \right\} \ \bullet \ \bullet \ \bullet \right]^T$$

$$I_Z = \{ 1, \quad 2, \quad 3, \quad \bullet \quad \bullet \quad \bullet \quad 24 \} \quad \text{where} \quad N_Z = 24$$

$$I_\theta = \{ 1, \quad 2, \quad 3, \quad \bullet \quad \bullet \quad \bullet \quad 8 \} \quad \text{where} \quad N_\theta = 8$$

then $Z = Z(\theta) = \left[Z_1,\ Z_2,\ Z_3,\ \bullet\ \bullet\ \bullet\ Z_{24} \right]^T$

$\theta = \left[\theta_1,\ \theta_2,\ \theta_3,\ \bullet\ \bullet\ \bullet\ \theta_8 \right]^T$

Subject to:

$$\left.\begin{array}{l} \theta_{MIN_p} \le \theta_p \le \theta_{MAX_p} \\ \theta_{MIN_p} \in (-\infty, +\infty) \\ \theta_{MAX_p} \in (-\infty, +\infty) \end{array}\right\} \begin{array}{l} \text{Direct Constraints on the Control} \\ \theta - \text{vector Elements } \theta_p \text{ for: } \quad p \in I_\theta \end{array}$$

$$\left[\begin{array}{lcl} \phi_1(\theta) = \theta_1\theta_4 - \theta_2\theta_3 = 0 \\ \phi_2(\theta) = \theta_5\theta_8 - \theta_6\theta_7 = 0 \end{array}\right] \left\{\begin{array}{l} \text{Equality Constraint } \phi(\theta) - \text{vector for:} \\ N_{EQ} = 2 \end{array}\right.$$

$$\left[\begin{array}{lcl} \psi_1(\theta) = \psi_{1_{MAX}} - \sqrt{\theta_1^2 + \theta_2^2} \ge 0 \\ \psi_2(\theta) = \psi_{2_{MAX}} - \sqrt{\theta_3^2 + \theta_4^2} \ge 0 \\ \psi_3(\theta) = \psi_{3_{MAX}} - \sqrt{\theta_5^2 + \theta_6^2} \ge 0 \\ \psi_4(\theta) = \psi_{4_{MAX}} - \sqrt{\theta_7^2 + \theta_8^2} \ge 0 \end{array}\right] \left\{\begin{array}{l} \text{First Inequality Constraint} \\ {}^1\psi(\theta) - \text{sub-vector for:} \quad N_{IEQ_1} = 4 \end{array}\right.$$

$$\left[\begin{array}{lcl} \psi_5(\theta) = \psi_{5_{MAX}} - \left|\theta_1 - \theta_{0_1}\right| \ge 0 \\ \psi_6(\theta) = \psi_{6_{MAX}} - \left|\theta_2 - \theta_{0_2}\right| \ge 0 \\ \psi_7(\theta) = \psi_{7_{MAX}} - \left|\theta_3 - \theta_{0_3}\right| \ge 0 \\ \bullet \\ \bullet \\ \bullet \\ \psi_{12}(\theta) = \psi_{12_{MAX}} - \left|\theta_8 - \theta_{0_8}\right| \ge 0 \end{array}\right] \left\{\begin{array}{l} \text{Second Inequality Constraint} \\ {}^2\psi(\theta) - \text{sub-vector for:} \quad N_{IEQ_2} = 8 \end{array}\right.$$

The (24 x 8) T-Matrix NLP control problem has four sub-problems: (1) unconstrained optimisation, (2) optimisation with only equality constraints, (3) optimisation with only inequality constraints, and (4) optimisation with both equality constraints and inequality constraints. It was assumed that the tasking of these T-Matrix NLP control problems occurred within the framework of real-time controller duty cycles. To expedite the testing, experimentation, and evaluation of the NLPQLP System, the required input data was synthetically determined using a process designed expressly for this analysis. Specifically, a previously identified T-Matrix and an actual control θ_0 – vector/actual measurement Z_A – vector pair from a previous duty cycle were synthesised for use as input to these problems. Additionally, these already synthesised values were directly input to the (24 x 8) T-Matrix NLP control problems for comparison purposes. The classic regulator problem was solved to verify the NLP solutions to the unconstrained optimisation NLP problems. Agreement was obtained in all cases.

Listings of the command (DCL) file code and the Fortran main driver code for the Hewlett-Packard Alpha mainframe computer, and the input and output for the four sub-problems that were part of the (24 x 8) T-Matrix NLP control problems solved using the Hewlett-Packard Alpha mainframe computer, are presented in Appendix B, section B.3. A listing of the Fortran main driver code for the Mac Pro desktop computer, and the input and output for the four sub-problems that were part of the (24 x 8) T-Matrix NLP control problems solved using the Mac Pro desktop computer, are presented in Appendix C, section C.3.

These (24 x 8) T-Matrix NLP control problems are representative of actual rotorcraft control problems. The solutions to these problems were sufficiently fast to be included in real-time duty cycles.

2.2.4 (90 x 30) T-Matrix NLP Control Problems

For these problems, the number of elements (dimension) in the control θ – vector is 30 (i.e., N_θ = 30), and the number of elements (dimension) in the predicted measurement Z – vector is 90 (i.e., N_Z = 90). Correspondingly, the system or transfer T-Matrix is (90 x 30). The number of elements (dimension) in the Equality Constraint $\phi(\theta)$ – vector is 7 (i.e., N_{EQ} = 7), the number of elements in the First Inequality Constraint $^1\psi(\theta)$ – sub-vector is 15 (i.e., N_{IEQ_1} = 15), the number of elements in the Second Inequality Constraint $^2\psi(\theta)$ – sub-vector function is 30 (i.e., N_{IEQ_2} = 30), and the dimension of the Inequality Constraint $\psi(\theta)$ – vector is 45 (i.e., $N_{IEQ} = N_{IEQ_1} + N_{IEQ_2}$ = 45). The (90 x 30) T-Matrix NLP control problem with all constraints is defined as:

Determine the θ – vector, θ_{Sol}, that solves the problem:

$$\underset{\theta_p \in \theta}{Minimise} \qquad J = Z^T W_Z Z \qquad for \quad p \in I_\theta$$

where $$Z = Z(\theta) = Z_A + \mathsf{T}(\theta - \theta_0)$$

$$Z = Z(\theta) = \left[\bullet \ \bullet \ \bullet \ \left\{ Z_q \,\middle|\, q \in I_Z \right\} \ \bullet \ \bullet \ \bullet \right]^T$$

and $$\theta = \left[\bullet \ \bullet \ \bullet \ \left\{ \theta_p \,\middle|\, p \in I_\theta \right\} \ \bullet \ \bullet \ \bullet \right]^T$$

$$I_Z = \{ 1, \quad 2, \quad 3, \quad \bullet \ \bullet \ \bullet \quad 90 \} \quad \text{where} \quad N_Z = 90$$

$$I_\theta = \{ 1, \quad 2, \quad 3, \quad \bullet \ \bullet \ \bullet \quad 30 \} \quad \text{where} \quad N_\theta = 30$$

then

$$Z = Z(\theta) = \left[Z_1, \ Z_2, \ Z_3, \ \bullet \ \bullet \ \bullet \ Z_{90} \right]^T$$

$$\theta = \left[\theta_1, \ \theta_2, \ \theta_3, \ \bullet \ \bullet \ \bullet \ \theta_{30} \right]^T$$

Subject to:

$$\left.\begin{array}{l} \theta_{MIN_p} \le \theta_p \le \theta_{MAX_p} \\ \theta_{MIN_p} \in (-\infty, +\infty) \\ \theta_{MAX_p} \in (-\infty, +\infty) \end{array}\right\} \left\{\begin{array}{l} \text{Direct Constraints on the Control} \\ \theta - \text{vector Elements } \theta_p \text{ for:} \quad p \in I_\theta \end{array}\right.$$

$$\left.\begin{bmatrix} \phi_1(\theta) = \theta_1\theta_4 - \theta_2\theta_3 = 0 \\ \phi_2(\theta) = \theta_5\theta_8 - \theta_6\theta_7 = 0 \\ \phi_3(\theta) = \theta_9\theta_{12} - \theta_{10}\theta_{11} = 0 \\ \phi_4(\theta) = \theta_{13}\theta_{16} - \theta_{14}\theta_{15} = 0 \\ \phi_5(\theta) = \theta_{17}\theta_{20} - \theta_{18}\theta_{19} = 0 \\ \phi_6(\theta) = \theta_{21}\theta_{24} - \theta_{22}\theta_{23} = 0 \\ \phi_7(\theta) = \theta_{25}\theta_{28} - \theta_{26}\theta_{27} = 0 \end{bmatrix}\right\} \left\{\begin{array}{l} \text{Equality Constraint } \phi(\theta) - \text{vector for :} \\ N_{EQ} = 7 \end{array}\right.$$

$$\left.\begin{bmatrix} \psi_1(\theta) = \psi_{1_{MAX}} - \sqrt{\theta_1^2 + \theta_2^2} \geq 0 \\ \psi_2(\theta) = \psi_{2_{MAX}} - \sqrt{\theta_3^2 + \theta_4^2} \geq 0 \\ \psi_3(\theta) = \psi_{3_{MAX}} - \sqrt{\theta_5^2 + \theta_6^2} \geq 0 \\ \bullet \\ \bullet \\ \bullet \\ \psi_{15}(\theta) = \psi_{15_{MAX}} - \sqrt{\theta_{29}^2 + \theta_{30}^2} \geq 0 \end{bmatrix}\right\} \left\{\begin{array}{l} \text{First Inequality Constraint} \\ {}^1\psi(\theta) - \text{sub - vector for :} \quad N_{IEQ_1} = 15 \end{array}\right.$$

$$\left.\begin{bmatrix} \psi_{16}(\theta) = \psi_{16_{MAX}} - \left|\theta_1 - \theta_{0_1}\right| \geq 0 \\ \psi_{17}(\theta) = \psi_{17_{MAX}} - \left|\theta_2 - \theta_{0_2}\right| \geq 0 \\ \psi_{18}(\theta) = \psi_{18_{MAX}} - \left|\theta_3 - \theta_{0_3}\right| \geq 0 \\ \bullet \\ \bullet \\ \bullet \\ \psi_{45}(\theta) = \psi_{45_{MAX}} - \left|\theta_{30} - \theta_{0_{30}}\right| \geq 0 \end{bmatrix}\right\} \left\{\begin{array}{l} \text{Second Inequality Constraint} \\ {}^2\psi(\theta) - \text{sub - vector for :} \quad N_{IEQ_2} = 30 \end{array}\right.$$

The (90 x 30) T-Matrix NLP control problem has four sub-problems: (1) unconstrained optimisation, (2) optimisation with only equality constraints, (3) optimisation with only inequality constraints, and (4) optimisation with both equality constraints and inequality constraints. It was assumed that the tasking of these T-Matrix NLP control problems occurred within the framework of real-time controller duty cycles. To expedite the testing, experimentation, and evaluation of the NLPQLP System, the required input data was synthetically determined using a process designed expressly for this analysis. Specifically, a previously identified T-Matrix and an actual control θ_0 – vector/actual measurement Z_A – vector pair from a previous duty cycle were synthesised for use as input to these problems. Additionally, these already synthesised values were directly input to the (90 x 30) T-Matrix NLP control problems for comparison purposes. The classic regulator problem was solved to verify the NLP solutions to the unconstrained optimisation NLP problems. Agreement was obtained in all cases.

Listings of the command (DCL) file code and the Fortran main driver code for the Hewlett-Packard Alpha mainframe computer, and the input and output for the four sub-problems that were part of the (90 x 30) T-Matrix NLP control problems solved using the Hewlett-Packard Alpha mainframe computer, are presented in Appendix B, section B.4. A listing of the Fortran main driver code for the Mac Pro desktop computer, and the input and output for the four sub-problems that were part of the (90 x 30) T-Matrix NLP control problems solved using the Mac Pro desktop computer, are presented in Appendix C, section C.4.

These (90 x 30) T-Matrix NLP control problems are representative of aerodynamic surface design and/or aircraft configuration problems and, although they were solved rapidly, they are more suitable to non-real-time design applications.

2.2.5 (90 x 60) T-Matrix NLP Control Problems

For these problems, the number of elements (dimension) in the control θ – vector is 60 (i.e., N_θ = 60), and the number of elements (dimension) in the predicted measurement Z – vector is 90 (i.e., N_Z = 90). Correspondingly, the system or transfer T-Matrix is (90 x 60). The number of elements (dimension) in the Equality Constraint $\phi(\theta)$ –vector is 15 (i.e., N_{EQ} = 15), the number of elements in the First Inequality Constraint ${}^1\psi(\theta)$ – sub-vector is 30 (i.e., N_{IEQ_1} = 30), the number of elements in the Second Inequality Constraint ${}^2\psi(\theta)$ –sub-vector function is 60 (i.e., N_{IEQ_2} = 60), and the dimension of the Inequality Constraint $\psi(\theta)$ – vector is 90 (i.e., N_{IEQ} = N_{IEQ_1} + N_{IEQ_2} = 90). The (90 x 60) T-Matrix NLP control problem with all constraints is defined as:

Determine the θ – vector, θ_{Sol}, that solves the problem:

$$\underset{\theta_p \in \theta}{\textit{Minimise}} \qquad J = Z^{\mathsf{T}} W_Z \, Z \qquad \textit{for} \quad p \in I_\theta$$

where

$$Z = Z(\theta) = Z_A + \mathsf{T}\left(\theta - \theta_0\right)$$

$$Z = Z(\theta) = \left[\bullet \;\; \bullet \;\; \bullet \;\; \left\{ Z_q \,\middle|\, q \in I_Z \right\} \;\; \bullet \;\; \bullet \;\; \bullet \right]^{\mathsf{T}}$$

and

$$\theta = \left[\bullet \;\; \bullet \;\; \bullet \;\; \left\{ \theta_p \,\middle|\, p \in I_\theta \right\} \;\; \bullet \;\; \bullet \;\; \bullet \right]^{\mathsf{T}}$$

$$I_Z = \{ 1, \quad 2, \quad 3, \quad \bullet \;\; \bullet \;\; \bullet \;\; 90 \} \qquad \text{where} \quad N_Z = 90$$

$$I_\theta = \{ 1, \quad 2, \quad 3, \quad \bullet \;\; \bullet \;\; \bullet \;\; 60 \} \qquad \text{where} \quad N_\theta = 60$$

then

$$Z = Z(\theta) = \left[Z_1, \;\; Z_2, \;\; Z_3, \;\; \bullet \;\; \bullet \;\; \bullet \;\; Z_{90} \right]^{\mathsf{T}}$$

$$\theta = \left[\theta_1, \;\; \theta_2, \;\; \theta_3, \;\; \bullet \;\; \bullet \;\; \bullet \;\; \theta_{60} \right]^{\mathsf{T}}$$

Subject to:

$$\left.\begin{array}{l} \theta_{MIN_p} \le \theta_p \le \theta_{MAX_p} \\ \theta_{MIN_p} \in \left(-\infty, +\infty\right) \\ \theta_{MAX_p} \in \left(-\infty, +\infty\right) \end{array}\right\} \begin{array}{l} \text{Direct Constraints on the Control} \\ \theta - \text{vector Elements } \theta_p \text{ for: } \quad p \in I_\theta \end{array}$$

$$\left[\begin{array}{lcl} \phi_1(\theta) = \theta_1\theta_4 - \theta_2\theta_3 = 0 \\ \phi_2(\theta) = \theta_5\theta_8 - \theta_6\theta_7 = 0 \\ \phi_3(\theta) = \theta_9\theta_{12} - \theta_{10}\theta_{11} = 0 \\ \vdots \\ \phi_{15}(\theta) = \theta_{57}\theta_{60} - \theta_{58}\theta_{59} = 0 \end{array}\right]$$

Equality Constraint $\phi(\theta)$ – vector for : $N_{EQ} = 15$

$$\left[\begin{array}{lcl} \psi_1(\theta) = \psi_{1_{MAX}} - \sqrt{\theta_1^2 + \theta_2^2} \geq 0 \\ \psi_2(\theta) = \psi_{2_{MAX}} - \sqrt{\theta_3^2 + \theta_4^2} \geq 0 \\ \psi_3(\theta) = \psi_{3_{MAX}} - \sqrt{\theta_5^2 + \theta_6^2} \geq 0 \\ \vdots \\ \psi_{30}(\theta) = \psi_{30_{MAX}} - \sqrt{\theta_{59}^2 + \theta_{60}^2} \geq 0 \end{array}\right]$$

First Inequality Constraint

${}^1\psi(\theta)$ – sub - vector for : $N_{IEQ_1} = 30$

$$\left[\begin{array}{lcl} \psi_{31}(\theta) = \psi_{31_{MAX}} - \left|\theta_1 - \theta_{0_1}\right| \geq 0 \\ \psi_{32}(\theta) = \psi_{32_{MAX}} - \left|\theta_2 - \theta_{0_2}\right| \geq 0 \\ \psi_{33}(\theta) = \psi_{33_{MAX}} - \left|\theta_3 - \theta_{0_3}\right| \geq 0 \\ \vdots \\ \psi_{90}(\theta) = \psi_{90_{MAX}} - \left|\theta_{60} - \theta_{0_{60}}\right| \geq 0 \end{array}\right]$$

Second Inequality Constraint

${}^2\psi(\theta)$ – sub - vector for : $N_{IEQ_2} = 60$

The (90 x 60) T-Matrix NLP control problem has four sub-problems: (1) unconstrained optimisation, (2) optimisation with only equality constraints, (3) optimisation with only inequality constraints, and (4) optimisation with both equality constraints and inequality constraints. It was assumed that the tasking of these T-Matrix NLP control problems occurred within the framework of real-time controller duty cycles. To expedite the testing, experimentation, and evaluation of the NLPQLP System, the required input data was synthetically determined using a process designed expressly for this analysis. Specifically, a previously identified T-Matrix and an actual control θ_0– vector/actual measurement Z_A– vector pair from a previous duty cycle were synthesised for use as input to these problems. Additionally, these already synthesised values were directly input to the (90 x 60) T-Matrix NLP control problems for comparison purposes. The classic regulator problem was solved to verify the NLP solutions to the unconstrained optimisation NLP problems. Agreement was obtained in all cases.

Listings of the command (DCL) file code and the Fortran main driver code for the Hewlett-Packard Alpha mainframe computer, and the input and output for the four sub-problems that were part of the (90 x 60) T-Matrix NLP control problems solved using the Hewlett-Packard Alpha mainframe computer, are presented in Appendix B, section B.5. A listing of the Fortran main driver code for the Mac Pro desktop computer, and the input and output for the four sub-problems that were part of the (90 x 60) T-Matrix NLP control problems solved using the Mac Pro desktop computer, are presented in Appendix C, section C.5.

These (90 x 60) T-Matrix NLP control problems are representative of aerodynamic surface design and/or aircraft configuration problems and, although they were solved rapidly, they are more suitable to non-real-time design applications.

2.3 Synthetic Data

The specific problems that were solved during this study were representative of actual problems to be solved during experimentation and/or testing of various rotorcraft configurations. It is assumed that the tasking of these T-Matrix NLP control problems occurs within the framework of real-time controller duty cycles. Tasking of one of these T-Matrix NLP control problems during the current duty cycle requires: (1) a previously identified T-Matrix, and (2) the actual control θ_0– vector and the actual measurement Z_A– vector pair determined during a previous duty cycle or at a reference epoch time. Preparations for such experimentation and/or testing requires the definition of the specific NLP problem (i.e., the performance index, the dimension and elements of the control vector and the measurement vector, and all constraints) to be solved. The main driver program, which defines the problem to be solved and tasks the NLPQLP System to solve it, must be coded, verified for proper functioning, and tuned-up so that it reliably and efficiently solves the required problems during actual tests. This, of course, must precede the test and the generation of actual associated test data.

In an actual wind tunnel or flight test, the identification and/or acquisition of the required data would normally be tasked during a previous duty cycle or at a reference epoch time and would be transmitted to the NLP program host computer for the solution of the NLP problem. The solution control θ_{Sol}– vector would then be transmitted back to the controller during the current duty cycle. It is necessary to have input/output (I/O) interface/compatibility between the controller and the NLP host computer in order to successfully transmit usable data between the controller and the NLP host computer. This issue is not addressed in this document.

Although it might be possible to use similar test data from another test for verification and tuning purposes, in general this is cumbersome and can be unnecessarily time consuming. In addition, there could be I/O interface/compatibility issues associated with use of test data from another test. Synthetic determination of the T-Matrix, the actual control θ_0– vector, and the actual measurement Z_A– vector data avoids problems associated with use of test data from another test and provides a simple, rapid method to obtain the required data during verification and tuning.

A synthesis procedure was designed to provide the T-Matrix and a previous duty cycle "actual" control θ_0– vector/"actual" measurement Z_A– vector pair required for the verification of the NLP main driver codes and subsequent tuning of the operation of these codes to solve the type of problems expected during the test. In order to develop a realistic test of the main driver codes and the NLPQLP System, a small degree of randomness was included in the synthetic modelling. Uniformly distributed pseudo-random numbers were selected for this synthesis process (see section 2.3.1).

First, the T-Matrix is determined by defining its elements to be pseudo-random numbers between –1.0 inclusive and +1.0 exclusive (see section 2.3.2). Scaling coefficients are provided to ensure that the norm to the T-Matrix is within acceptable limits. Next, the "actual" control θ_0– vector is likewise determined by defining its elements to be pseudo-random numbers between –1.0 inclusive and +1.0 exclusive (see section 2.3.2). In this case, the elements are constrained to be within specified limits à la "external limiting." Although the definition of the T-Matrix and the "actual" control θ_0– vector was accomplished using pseudo-random numbers, these elements did not have to be defined randomly and could have been directly input or generated otherwise. These elements were generated randomly for convenience. The essential requirements for these elements are that the scaling and limits be reasonable and the resulting values are realistic. The degree of randomness to the model is introduced in the definition of the "actual" measurement Z_A– vector. The "actual" measurement Z_A– vector is defined by adding uniformly distributed pseudo-random numbers to the $T\theta_0$ product used in the definition of the Z_A– vector (see section 2.3.2).

2.3.1 Definition of the Random Number Generator Function RAN(SEED)

The random number generator function, RAN(•), employed for data synthesis from both the Hewlett-Packard Alpha mainframe computer and Mac Pro desktop computer, produces a real uniformly pseudo-random number between 0.0 inclusive and 1.0 exclusive $\left(\text{i.e., } \text{RAN}\left(\bullet\right) \in [0,1)\right)$. Because the algorithm employed by RAN(•) on the Hewlett-Packard Alpha mainframe computer is a VMS System subroutine, it is not necessary to provide this subroutine for the main driver code on that computer. In order to provide nearly identical main driver codes, and because the corresponding G95 intrinsic uniform pseudo-random number generator function is named RAND(•), a Fortran code for a RAN(•) subroutine that calls RAND(•) is provided in addition to the main driver code for the Mac Pro desktop computer. It is, however, necessary to define this RAN(•) with EXTERNAL and REAL statements in the Mac Pro main driver codes. The provision of the EXTERNAL and REAL statements in the Mac Pro main driver codes is the only difference from the corresponding main driver codes for the Hewlett-Packard Alpha mainframe computer. These RAN(•) codes employ different algorithms to generate the pseudo-random numbers, and correspondingly will, in general, produce different numerical values.

Both of these RAN(•) random number generators require provision of a seed in the calling argument. This argument (i.e., the seed) should initially be set to a large odd-integer value for the first call to RAN(•). Its value will be updated during this call to RAN(•) to be used in the next call to RAN(•).

2.3.2 Determination of the Synthetic T-Matrix, the "Actual" Control θ_0– Vector, and the "Actual" Measurement Z_A–Vector

The methodology employed to synthetically determine the T-Matrix, the actual control θ_0– vector, and the actual measurement Z_A– vector data is:

First, synthesise the T-Matrix *according to:*

$$\mathsf{T}_{q,p} = \mathsf{C}_1\left[2 \bullet \mathsf{RAN}\left(\mathsf{ISEED1}_k\right) - 1\right] \; + \; \mathsf{C}_2\left[2 \bullet \mathsf{RAN}\left(\mathsf{JSEED1}_l\right) - 1\right]$$

where

k = *is the index number of the input seed for the first call to* $\mathsf{RAN}\left(\bullet\right)$.

l = *is the index number of the input seed for the* second *call to* $\mathsf{RAN}\left(\bullet\right)$.

and

$$\mathsf{T} = \begin{bmatrix} \bullet & \bullet & \bullet \\ \bullet & \mathsf{T}_{q,p} & \bullet \\ \bullet & \bullet & \bullet \end{bmatrix} \; is \; \left(\mathsf{N_Z} \times \mathsf{N}_\theta\right) \quad \forall \; q \in I_Z \quad and \quad \forall \; p \in I_\theta$$

Next, synthesise the "actual" θ_0 – Vector *according to*:

$$\theta_{p_{Initial}} = \mathsf{C}_3\left[2 \bullet \mathsf{RAN}\left(\mathsf{ISEED2}_k\right) - 1\right] \; + \; \mathsf{C}_4\left[2 \bullet \mathsf{RAN}\left(\mathsf{JSEED2}_l\right) - 1\right] \qquad \forall \quad p \in I_\theta$$

Subject to:

$$\left.\begin{array}{l} \theta_{\mathsf{MIN}_p} \le \theta_{p_{Initial}} \le \theta_{\mathsf{MAX}_p} \\ \theta_{\mathsf{MIN}_p} \in \left(-\infty, +\infty\right) \\ \theta_{\mathsf{MAX}_p} \in \left(-\infty, +\infty\right) \end{array}\right\} \left\{\begin{array}{l} \text{Direct Constraints on the Control} \\ \theta - \text{vector Elements } \theta_p \text{ for:} \quad p \in I_\theta \end{array}\right.$$

if $\theta_{p_{Initial}} \le \theta_{\mathsf{MIN}_p} + \varepsilon$ *then* $\theta_{p_{Initial}} = \theta_{\mathsf{MIN}_p} + \varepsilon$

if $\theta_{p_{Initial}} \ge \theta_{\mathsf{MAX}_p} - \varepsilon$ *then* $\theta_{p_{Initial}} = \theta_{\mathsf{MIN}_p} - \varepsilon$

otherwise there is no change to the value of $\theta_{p_{Initial}}$ *as determined by the random equation above.*

then

$$\theta_0 = \left[\bullet \ \bullet \ \bullet \ \left\{ \theta_{p_{Initial}} \,\middle|\, p \in I_\theta \right\} \ \bullet \ \bullet \ \bullet \right]^{\mathsf{T}}$$

Finally, synthesise the "actual" Z_{A} – Vector according to:

$$\Delta Z_{\mathsf{A}_q} = \mathsf{C}_5 \left[2 \bullet \mathsf{RAN}\left(\mathsf{ISEED3}_k \right) - 1 \right] + \mathsf{C}_6 \left[2 \bullet \mathsf{RAN}\left(\mathsf{JSEED3}_l \right) - 1 \right]$$
$$\forall \ q \in I_{\mathsf{Z}}$$

then

$$\Delta Z_{\mathsf{A}} = \left[\bullet \ \bullet \ \bullet \ \left\{ \Delta Z_{\mathsf{A}_q} \,\middle|\, q \in I_{\mathsf{Z}} \right\} \ \bullet \ \bullet \ \bullet \right]^{\mathsf{T}}$$

and

$$Z_{\mathsf{A}} = \mathsf{T}\,\theta_0 + \Delta Z_{\mathsf{A}}$$

where

$\mathsf{RAN}(\bullet)$ *is the Uniform Random Number Distribution Function which yields a uniformly random real number* $\in [0,1)$

$$I_{\mathsf{Z}} = \left\{ \bullet \ \bullet \ \bullet \ \left\{ \forall \ q \ \ni \ Z_q \in Z \right\} \ \bullet \ \bullet \ \bullet \right\}$$

$$I_\theta = \left\{ \bullet \ \bullet \ \bullet \ \left\{ \forall \ p \ \ni \ \theta_p \in \theta \right\} \ \bullet \ \bullet \ \bullet \right\}$$

2.4 The Regulator Problem

Unlike the non-linear programming problems that are constrained/unconstrained optimisation problems, the regulator problem is a steady-state problem; its solution process seeks maintenance of a steady-state condition with minimal control and, in some cases, minimal control rate of change. Because the regulator problem solution is analytically explicit and known, its solution process is fast and has a relatively small computational load compared to the constrained/unconstrained optimisation solution processes. In some cases, "External Limiting" constraints (i.e., direct maximum limits on control vector elements imposed *after* the analytic explicit solution is obtained) are imposed on control vector elements. Correspondingly, early attempts (i.e., circa the 1950s) to solve constrained/unconstrained optimisation control problems formulated these problems as regulator problems because the algorithms for constrained/ unconstrained optimisation solution processes and computer technology were in their early stages of development and not sufficiently reliable or efficient for this purpose. As computer technology advances occurred and efficient, reliable optimisation techniques were developed, use of numerical optimisation techniques became feasible for actual test applications.

A control vector metric, and a rate of change of the control vector metric if required, are adjoined to a steady-state excursion metric to form the performance index for the regulator problem. By appropriately defining the steady-state excursion metric, which is the first term in the performance index, and carefully tuning the weighting coefficients for all the terms in the performance index, a pseudo-optimal solution can be obtained that satisfies, or nearly satisfies, any required constraints. This tuning must, of course, be accomplished before actual test applications.

As in the case of the General T-Matrix NLP control problem described in section 2.2, a T-Matrix linear plant model that relates the measurement Z – vector to the control θ – vector is assumed for the regulator problem described below. The first-term performance index is the steady-state excursion metric and a quadratic function of a T-Matrix plant model, and correspondingly a quadratic function of the control θ – vector. The second term in the performance index is simply a weighted control θ – vector quadratic. If required, the third term in the performance index is a weighted time rate of change of the control θ – vector quadratic. The regulator problem is:

Determine the θ – vector, θ_{Sol}, that solves the problem:

$$\underset{\theta_p \in \theta}{Minimise} \qquad J = Z^{\mathsf{T}} W_Z Z + \theta^{\mathsf{T}} W_\theta \theta + \dot{\theta}^{\mathsf{T}} W_{\dot{\theta}} \dot{\theta} \qquad for \quad p \in I_\theta$$

where

$$Z = Z(\theta) = Z_A + \mathsf{T}(\theta - \theta_0)$$

$$Z = Z(\theta) = \left[\bullet \quad \bullet \quad \bullet \quad \left\{ Z_q \,\middle|\, q \in I_Z \right\} \quad \bullet \quad \bullet \quad \bullet \right]^{\mathsf{T}}$$

and

$$\theta = \left[\bullet \quad \bullet \quad \bullet \quad \left\{ \theta_p \,\middle|\, p \in I_\theta \right\} \quad \bullet \quad \bullet \quad \bullet \right]^{\mathsf{T}}$$

$$\dot{\theta} = \left[\bullet \quad \bullet \quad \bullet \quad \left\{ \dot{\theta}_p \,\middle|\, p \in I_\theta \right\} \quad \bullet \quad \bullet \quad \bullet \right]^{\mathsf{T}}$$

$$I_Z = \left\{ \bullet \quad \bullet \quad \bullet \quad \left\{ \forall \; q \; \ni \; Z_q \in Z \right\} \quad \bullet \quad \bullet \quad \bullet \right\}$$

$$I_\theta = \left\{ \bullet \quad \bullet \quad \bullet \quad \left\{ \forall \; p \; \ni \; \theta_p \in \theta \right\} \quad \bullet \quad \bullet \quad \bullet \right\}$$

Subject to NO constraints per se during the regulator problem solution process to determine θ_{Sol}.

The Explicit Solution θ_{Sol} – vector is:

$$\theta_{Sol} = \left(D\, W_{\dot{\theta}} + D\, \mathsf{T}^{\mathsf{T}} W_Z \mathsf{T} \right) \theta_0 - \alpha D\, \mathsf{T}^{\mathsf{T}} W_Z Z_A$$

where

$$D = \left(\mathsf{T}^{\mathsf{T}} W_Z \mathsf{T} + W_{\dot{\theta}} + W_\theta \right)^{-1}$$

and

$$\alpha \in [0, 1]$$

In some cases, "External Limiting" constraints are imposed on θ_{Sol} after the explicit analytic solution for θ_{Sol} is determined. Specifically:

$$\left.\begin{array}{l} \theta_{\mathrm{MIN}_p} \leq \theta_{Solp} \leq \theta_{\mathrm{MAX}_p} \\ \theta_{\mathrm{MIN}_p} \in (-\infty, +\infty) \\ \theta_{\mathrm{MAX}_p} \in (-\infty, +\infty) \end{array}\right\} \left\{\begin{array}{l} \text{Direct Constraints on the Control} \\ \theta_{Sol}\text{ – vector Elements } \theta_{Solp} \text{ for: } p \in I_\theta \end{array}\right.$$

if $\theta_{Sol_p} \leq \theta_{\mathrm{MIN}_p}$ *then set* $\theta_{Sol_p} = \theta_{\mathrm{MIN}_p}$

if $\theta_{Sol_p} \geq \theta_{\mathrm{MAX}_p}$ *then set* $\theta_{Sol_p} = \theta_{\mathrm{MIN}_p}$

otherwise there is no change to the value of θ_{Sol_p} *as determined by the equation for the explicit solution for* θ_{Sol_p}.

then

$$\theta_{Sol} = \left[\bullet \quad \bullet \quad \bullet \quad \left\{ \theta_{Sol_p} \,\middle|\, p \in I_\theta \right\} \quad \bullet \quad \bullet \quad \bullet \right]^{\mathrm{T}}$$

3.0 Results and Conclusions

The newly acquired Version 3.1 of the NLPQLP System was used to solve several typical constrained and unconstrained optimisation problems of the type encountered in various rotorcraft wind tunnel and flight tests on both the Hewlett-Packard Alpha mainframe computer and the Mac Pro desktop computer. The associated software and codes installed on the Mac Pro desktop computer should be transportable to a Mac laptop computer for use in a wind tunnel. A linear dependence (i.e., a T-Matrix plant model) of the measurement vector (the measurement Z – vector) on the control vector (i.e., the control θ – vector) was assumed. These problems ranged from a relatively simple, unconstrained 4-vector control problem, to a relatively large, constrained 60-vector control problem. Solutions were obtained for all problems considered. Although tuning and some input adjustments were required to successfully solve the large, constrained 60-vector control problem, the NLPQLP System proved to be an efficient and reliable method to solve these problems.

The problems solved in this analysis included: (6 x 4) T-Matrix NLP control problems, (6 x 6) T-Matrix NLP control problems, (24 x 8) T-Matrix NLP control problems, (90 x 30) T-Matrix NLP control problems, and (90 x 60) T-Matrix NLP control problems. Each of these problems had four sub-problems: (1) unconstrained optimisation, (2) optimisation with only equality constraints, (3) optimisation with only inequality constraints, and (4) optimisation with both equality constraints and inequality constraints. It was assumed that the tasking of these T-Matrix NLP control problems occurred within the framework of real-time controller duty cycles. To expedite the testing, experimentation, and evaluation of the NLPQLP System, the required input data was synthetically determined using a process designed expressly for this analysis. Specifically, a previously identified T-Matrix and an actual control θ_0– vector/actual measurement Z_A– vector pair from a previous duty cycle, or at a reference epoch time, were synthesised for use as input to these problems. Additionally, these already synthesised values were directly input to the (6 x 4), (6 x 6), and (24 x 8) T-Matrix NLP control problems for comparison purposes. The classic regulator problem was solved to verify the NLP solutions to the unconstrained optimisation NLP problems. Agreement was obtained in all cases.

The (6 x 4), (6 x 6), and (24 x 8) T-Matrix NLP control problems are representative of actual rotorcraft control problems. The solutions to these problems were sufficiently fast to be included in real-time duty cycles. The (90 x 30) and (90 x 60) T-Matrix NLP problems are representative of aerodynamic surface design and/or aircraft configuration problems and, although they were solved rapidly, they are more suitable to non-real-time design applications.

The results are shown in Appendix B and Appendix C, separate volumes of this report.

Listings of the command (DCL) file code and the Fortran main driver code for the Hewlett-Packard Alpha mainframe computer, and the input and output for the four sub-problems that were part of the (6 x 4), (6 x 6), (24 x 8), (90 x 30), and (90 x 60) T-Matrix NLP control problems solved using the Hewlett-Packard Alpha mainframe computer, are presented in Appendix B. A listing of the Fortran main driver code for the Mac Pro desktop computer, and the input and output for the four sub-problems that were part of the (6 x 4), (6 x 6), (24 x 8), (90 x 30), and (90 x 60) T-Matrix NLP control problems solved using the Mac Pro desktop computer, are presented in Appendix C.

4.0 References

1. Leyland, J. A.: A Higher Harmonic Optimal Controller to Optimise Rotorcraft Aeromechanical Behaviour. NASA Technical Memorandum 110390, Mar. 1996.

2. Leyland, J. A.: A Closed-Loop Optimal Neural-Network Controller to Optimise Rotorcraft Aeromechanical Behaviour, Volume 1, Theory and Methodology. NASA TM-2001-209622, Mar. 2001.

3. Leyland, J. A.: A Closed-Loop Optimal Neural-Network Controller to Optimise Rotorcraft Aeromechanical Behaviour, Volume 2, Output from Two Sample Cases. NASA TM-2001-209623, Mar. 2001.

4. Leyland, J. A.: A General Regularisation Functional to Enhance the Update Convergence of a Neural-Network Rotorcraft Behaviour Model, Volume 1, Theory and Methodology. NASA TM-2002-211843, Oct. 2002.

5. Leyland, J. A.: A General Regularisation Functional to Enhance the Update Convergence of a Neural-Network Rotorcraft Behaviour Model, Volume 2, Output From Two Sample Cases. NASA TM-2002-211844, Oct. 2002.

6. Anon.: IMSL MATH/LIBRARY User's Manual, FORTRAN Subroutines for Mathematical Applications, Volume 3, Chapter 8: Nonlinearly Constrained Minimization Using Finite Difference Gradients. Version 1.1, MALB-USM-UNBND-EN8901-1.1, pp. 895–902, Jan. 1989.

7. Schittkowski. K.: Non-linear Programming Codes. *Lecture Notes in Economics and Mathematics*, **183**, Springer-Verlag, Berlin, Germany, 1980.

8. Schittkowski, K.: On the Convergence of a Sequential Quadratic Programming Method With an Augmented Lagrangian Line Search Function. *Mathematik Operationsforschung und Statistik, Serie Optimization*, **14**, pp. 197–216, 1983.

9. Schittkowski, K.: NLPQL: a FORTRAN Subroutine solving Constrained Non-linear Programming Problems. (Clyde L. Monma, Ed.), *Annals of Operations Research*, **5**, pp. 485–500, 1986.

10. Gill, P. E.; Murray, W.; Saunders, M. A.; and Wright, M. H.: Model Building and Practical Aspects of Non-Linear Programming. *Computational Mathematical Programming*, (K. Schittkowski, Ed.), NATO ASI Series, **15**, Springer-Verlag, Berlin, Germany, 1985.

11. Powell, M. J. D.: A Fast Algorithm for Non-linearly Constrained Optimisation Calculations. *Numerical Analysis Proceedings*, Dundee, 1977.

12. Powell, M. J. D.: A Fast Algorithm for Non-linearly Constrained Optimisation Calculations. *Lecture Notes in Economics and Mathematics*, **630**, Springer-Verlag, Berlin, Germany, pp. 144–157, 1978.

13. Stoer, J.: Principles of Sequential Quadratic Programming Methods in Solving Non-Linear Problems. *Computational Mathematical Programming*, (K. Schittkowski, Ed.), NATO ASI Series, **15**, Springer-Verlag, Berlin, Germany, 1985.

14. Schittkowski, K.: NLPQLP: a FORTRAN Implementation of a Sequential Quadratic Programming Algorithm With Distributed and Non-Monotone Line Search – User's Guide, Version 3.1. Department of Computer Science, University of Bayreuth, Germany, Feb. 2010.

15. Anon.: Programming in VAX FORTRAN, Software Version: V4.0. *VAX/VMS Manual No. AA-D034D-TE*, Digital Equipment Corporation (DEC), Maynard, Mass., Sept. 1984.

16. Vaught, A.: G95 FORTRAN Compiler, Mesa, Ariz., Oct. 2006.

Appendix A

NLPQLP: A Fortran Implementation of a Sequential Quadratic Programming Algorithm with Distributed and Non-Monotone Line Search - User's Guide, Version 3.1

By Professor Klaus Schittkowski
Department of Computer Science
University of Bayreuth, Germany

Table of Contents

Abstract A-1

1 Introduction A-2

2 Sequential Quadratic Programming Methods A-7

Algorithm 2.1 ..... A-10

Algorithm 2.2 ..... A-11

3 Performance Evaluation A-14

3.1 The Test Environment A-14

3.2 Testing Distributed Function Calls A-17

3.3 Function Evaluations and Gradient Approximations by a Difference Formulae Under Random Noise A-18

3.4 Testing Scaled Restarts A-19

4 Program Documentation A-22

5 Examples A-29

6 Conclusions A-35

References A-36

NLPQLP: A Fortran Implementation of a Sequential Quadratic Programming Algorithm with Distributed and Non-Monotone Line Search

- User's Guide, Version 3.1 -

Address: Prof. K. Schittkowski
Department of Computer Science
University of Bayreuth
D - 95440 Bayreuth

Phone: (+49) 921 557750

E-mail: klaus.schittkowski@uni-bayreuth. de

Web: http://www.klaus-schittkowski.de

Date: May, 2010

Abstract

The Fortran subroutine NLPQLP solves smooth nonlinear programming problems by a sequential quadratic programming (SQP) algorithm. This version is specifically tuned to run under distributed systems controlled by an input parameter (l). In case of computational errors as for example caused by inaccurate function or gradient evaluations, a non-monotone line search is activated. Numerical results are included which show that in case of noisy function values, a significant improvement of the performance is achieved compared to the version with monotone line search. Further stabilization is obtained by performing internal restarts in case of errors when computing the search direction due to inaccurate derivatives. The new version of NLPQLP successfully solves more than 90% of our 306 test examples subject to a stopping tolerance of 10^{-7}, although at most two digits in function values are correct in the worst case and although numerical differentiation leads to additional truncation errors. In addition, automated initial and periodic scaling with restarts is implemented. The usage of the code is documented and illustrated by an example.

Keywords: SQP, sequential quadratic programming, nonlinear programming, non-monotone line search, numerical algorithm, distributed computing, Fortran code.

1 Introduction

We consider the general optimization problem to minimize an objective function f under nonlinear equality and inequality constraints,

$$\begin{aligned} & \min f(x) \\ x \in I\!R^n : \; & g_j(x) = 0, \qquad j = 1, \bullet\bullet\bullet, m_e \\ & g_j(x) \geq 0, \qquad j = m_e + 1, \bullet\bullet\bullet, m \\ & x_l \leq x \leq x_u \end{aligned} \tag{1}$$

where x is an *n-dimensional* parameter vector. It is assumed that all problem functions $f(x)$ and $g_j(x)$, $j = 1, \bullet\bullet\bullet, m$ are continuously differentiable on the whole $I\!R^n$.

Sequential quadratic programming (SQP) is the standard general purpose method to solve smooth nonlinear optimization problems, at least under the following assumptions:

- The problem is not too large.
- Functions and gradients can be evaluated with sufficiently high precision.
- The problem is smooth and well scaled.

The original code NLPQL of Schittkowski [47] is a Fortran implementation of a sequential quadratic programming (SQP) algorithm. The numerical algorithm is based on extensive comparative numerical tests, see Schittkowski [40, 44, 42], Schittkowski et al. [56], Hock and Schittkowski [25], and on further theoretical investigations published in [41, 43, 45, 46]. The algorithm is extended to solve also nonlinear least squares problems efficiently, see [49] or [51], and to handle problems with very many constraints, see [54]. To conduct the numerical tests, a random test problem generator is developed for a major comparative study, see [40], Two collections with together 306 test problems are published in Hock and Schittkowski [25] and in Schittkowski [48]. Fortran source codes and a test frame can be downloaded from the home page of the author,

http://www.klaus-schittkowski.de

Many of them became part of the Cute test problem collection of Bongartz et al. [7]. About 80 test problems based on a Finite Element formulation are collected for a comparative evaluation in Schittkowski et al. [56]. A set of 1,300 least squares test problems solved by an extension of the code NLPQL to retain typical features of a Gauss-Newton algorithm, is described in [51].

Also these problems can be downloaded from the home page of the author together with an interactive software system called EASY-FIT, see [52].

Moreover, there exist hundreds of commercial and academic applications of NLPQL, for example

1. mechanical structural optimization, see Schittkowski, Zillober, Zotemantel [56] and Kneppe, Krammer, Winkler [28],
2. data fitting and optimal control of transdermal pharmaceutical systems, see Boderke, Schittkowski, Wolf [3] or Blatt, Schittkowski [6],
3. computation of optimal feed rates for tubular reactors, see Birk, Liepelt, Schittkowski, and Vogel [5],
4. food drying in a convection oven, see Frias, Oliveira, and Schittkowski [15],
5. optimal design of horn radiators for satellite communication, see Hartwanger, Schittkowski, and Wolf [23],
6. receptor-ligand binding studies, see Schittkowski [50],
7. optimal design of surface acoustic wave filters for signal processing, see Bfinner, Schittkowski, and van de Braak [8].

Previous and present versions of NLPQLP are part of commercial libraries, modeling systems, or optimization systems like

- IMSL Library (Visual Numerics Inc., Houston) for general nonlinear programming (Version 1.0, 1981),
- ANSYS/POPT (CAD-FEM, Grafing) for structural optimization,
- DesignXplorer (ANSYS, Canonsburg) for structural design optimization,
- STRUREL (RCP, Munich) for reliability analysis,
- TEMPO (OECD Reactor Project, Halden) for control of power plants,
- Microwave Office Suit (Applied Wave Research, El Segundo) for electronic design,
- MOOROPT (Marintek, Trondheim) for the design of mooring systems,
- iSIGHT (Enginious Software/Dassault) for multi-disciplinary CAE,
- POINTER (Synaps, Atlanta) for design automation,
- EXCITE (AVL, Graz) for non-linear dynamics of power units,
- ModeFRONTIER (ESTECO, Trieste) for integrated multi-objective and multidisciplinary design optimization,

- TOMLAB/MathLab (Tomlab Optimization, Vasteras, Sweden) for general nonlinear programming, least squares optimization, data fitting in dynamical systems,
- EASY-FIT (Schittkowski, Bayreuth) for data fitting in dynamical systems,
- OptiSLang (DYNARDO, Weimar), for structural design optimization,
- AMESim (IMAGINE, Roanne), for multidisciplinary system design,
- LMS OPTIMUS (NOESIS, Leuven, Belgium) for multi-disciplinary CAE,
- RADIOSS/M-OPT (MECALOG/Altair, Antony, France) for multi-disciplinary CAE,
- CHEMASIM (BASF, Ludwigshafen) for the design of chemical reactors.

Customers include, among many others, AMD, Astrium, BASF, Bayer, Bell Labs, BMW, Chevron Research, DLR, Dow Chemical, DuPont, EADS, EMCOSS, ENSIGC, EPCOS, ESA-ESOC, Eurocopter, Fantoft Prosess, General Electric, Hoechst, Hidro-electrica Espanola, IABG, IBM, Institute for Energy Technology Halden, KFZ Karlsruhe, Kongsberg Maritime, Lockheed Martin, Loral Space Systems, Markov Processes, Marintek, MTU, NASA Langley, NASA Ames, Nevesbu, National Airspace Laboratory, Norsk Hydro Research, Norwegian Computing Center, Norwegian Defense Agency, OECDHalden, Philips, Polysar, ProSim, Rolls-Royce, Shell, Siemens, Sintef, Solar Turbines, Statoil, TNO, Transpower, USAF Research Lab, Wright R & D Center and in addition dozens of academic research institutions all over the world.

The general availability of parallel computers and in particular of distributed computing in networks motivates a careful redesign of the original implementation NLPQL to allow simultaneous function evaluations. The resulting extensions are implemented and the code is called NLPQLP. An additional input parameter l is introduced for the number of parallel machines, that is the number of function calls to be executed simultaneously. In case of $l-1$, NLPQLP is more or less identical to NLPQL besides of additional changes of the code. Otherwise, the line search procedure is modified to allow parallel function calls, which can also be applied for approximating gradients by difference formulae. The mathematical background is outlined, in particular the modification of the line search algorithm to retain convergence under parallel systems. It must be emphasized that dis tributed computation of function values is only simulated throughout the paper. It is up to the user to adopt the code to a particular parallel environment.

However, SQP methods are quite sensitive subject to round-off or any other errors in function and especially gradient values. If objective or constraint functions cannot be computed within machine accuracy or if the accuracy by which gradients are approximated is above the termination tolerance, the code could break down typically with the error message IFAIL = 4. In this situation, the line search cannot be terminated within a given number of iterations and the algorithm is stopped.

All new versions since 2.0 makes use of non-monotone fine search in the error situation described above. The idea is to replace the reference value of the fine search termination check, $\psi_{r_k}(x_k, v_k)$, by

$$\max\left\{ \psi_{r_j}(x_j, v_j) \ : \ j = k-p, \bullet\bullet\bullet, k \right\}$$

where $\psi_r(x, v)$ is a merit function and p a given parameter. The general idea is not new and for example described in Dai [11], where a general convergence proof for the unconstrained case is presented. The general idea goes back to Grippo, Lampariello, and Lucidi [18], and was extended to constrained optimization and trust region methods in a series of subsequent papers, see Bonnans et al. [4], Deng et al. [13], Grippo et al. [19, 20], Ke and Han [26], Ke et al. [27], Lucidi et al. [30], Panier and Tits [34], Raydan [39], and Toint [59, 60]. However, there is a basic difference in the methodology: Our goal is to allow monotone line searches as long as they terminate successfully, and to apply a non-monotone one only in a special error situation.

Despite of strong analytical results, SQP methods do not always terminate successfully. Besides of the difficulties leading to the usage of non-monotone line search, it might happen that the search direction as computed from a certain quadratic programming sub-problem, is not a downhill direction of the merit function needed to perform a fine search. Possible reasons are again severe errors in function and especially gradient evaluations, or a violated regularity condition concerning linear independency of gradients of active constraints (LICQ). In the latter case, the optimization problem is not modelled in a suitable way to solve it directly by an SQP method. Our new version performs an automated restart as soon as a corresponding error message appears. The BFGS quasi-Newton matrix is reset to a multiple of the identity matrix and the matrix update procedure starts from there.

Scaling is an extremely important issue and an efficient procedure is difficult to derive in the general case without knowing too much about the numerical structure of the optimisation problem. If requested by the user, the first BFGS update is started from a multiple of the identity matrix, which takes into account information from the solution of the initial quadratic programming sub-problem. This restart can be repeated periodically with successively adapted scaling parameters.

In Section 2 we outline the general mathematical structure of an SQP algorithm, the non-monotone line search, and the modifications to run the code under distributed systems. Section 3 contains some numerical results obtained for a set of 306 standard test problems of the collections published in Hock and Schittkowski [25] and in Schittkowski [48]. They show the sensitivity of the new version with respect to the number of parallel machines and the influence of gradient approximations under uncertainty. Moreover, we test the non-monotone line search versus the monotone one, and generate noisy test problems by adding random errors to function values and by inaccurate gradient approximations. This situation appears frequently in practical environments, where complex simulation codes prevent accurate responses and where gradients can only be computed by a difference formula. The usage of the Fortran subroutine is documented in Section 4 and Section 5 contains an illustrative examples.

2 Sequential Quadratic Programming Methods

Sequential quadratic programming or SQP methods belong to the most powerful nonlinear programming algorithms we know today for solving differentiable nonlinear programming problems of the form (1). The theoretical background is described e.g. in Stoer [58] in form of a review, or in Spellucci [57] in form of an extensive text book. From the more practical point of view, SQP methods are also introduced in the books of Papalambros, Wilde [35] and Edgar, Himmelblau [14]. Their excellent numerical performance is tested and compared with other methods in Schittkowski [40], and since many years they belong to the most frequently used algorithms to solve practical optimization problems.

To facilitate the notation of this section, we assume that upper and lower bounds $x_u$and xi are not handled separately, i.e., we consider the somewhat simpler formulation

$$\begin{aligned} & \min f(x) \\ x \in I\!R^n : \quad & g_j(x) = 0, \qquad j = 1, \bullet\bullet\bullet, m_e \\ & g_j(x) \geq 0, \qquad j = m_e + 1, \bullet\bullet\bullet, m \end{aligned} \tag{2}$$

It is assumed that all problem functions $f(x)$ and $g_j(x)$, $j = 1, \bullet\bullet\bullet, m$ are continuously differentiable on $I\!R^n$.

The basic idea is to formulate and solve a quadratic programming sub-problem in each iteration which is obtained by linearising the constraints and approximating the Lagrangian function

$$L(x,u) := f(x) - \sum_{j=1}^{m} u_j g_j(x) \tag{3}$$

quadratically, where $x \in I\!R^n$ is the primal variable and $u = (u_1, \bullet\bullet\bullet, u_m)^T \in I\!R^m$ the multiplier vector.

To formulate the quadratic programming sub-problem, we proceed from given iterates $x \in I\!R^n$ an approximation of the solution, $v \in I\!R^m$ an approximation of the multipliers, and $B_k \in I\!R^{n\,x\,n}$, an approximation of the Hessian of the Lagrangian function. Then one has to solve the quadratic programming problem

$$\begin{aligned} & \min \tfrac{1}{2} d^T B_k d \; + \; \nabla f(x_k)^T d \\ d \in I\!R^n \; : \; & \nabla g_j \left(x_k\right)^T d \; + \; g_j \left(x_k\right) = 0 \, , \qquad j \; = 1, \bullet \bullet \bullet , \; m_e \\ & \nabla g_j \left(x_k\right)^T d \; + \; g_j \left(x_k\right) \geq 0 \, , \qquad j \; = m_e + 1, \bullet \bullet \bullet , \; m \end{aligned} \tag{4}$$

Let d_k be the optimal solution and u_k the corresponding multiplier of this sub-problem. A new iterate is obtained by

$$\begin{pmatrix} x_{k+1} \\ v_{k+1} \end{pmatrix} \; : \; = \begin{pmatrix} x_k \\ v_k \end{pmatrix} \; + \; \alpha_k \begin{pmatrix} d_k \\ u_k - v_k \end{pmatrix} \tag{5}$$

where $\alpha_k \in \left(0, 1\right]$ is a suitable step length parameter.

Although we are able to guarantee that the matrix B_k is positive definite, it is possible that (4) is not solvable due to inconsistent constraints. One possible remedy is to introduce an additional variable $\delta \in I\!R$ leading to a modified quadratic programming problem, see Schittkowski [47] for details.

The steplength parameter α_k is required in (5) to enforce global convergence of the SQP method, i.e., the approximation of a point satisfying the necessary Karush-Kuhn-Tucker optimality conditions when starting from arbitrary initial values, typically a user-provided $x_0 \in I\!R^n$ and $v_0 \in 0, \; B_0 \in I$. α_k should satisfy at least a sufficient decrease condition of a merit function $\phi_r(\alpha)$ given by

$$\phi_r(\alpha) \; : \; = \psi_r \left[\begin{pmatrix} x \\ v \end{pmatrix} \; + \; \alpha \begin{pmatrix} d \\ u - v \end{pmatrix} \right] \tag{6}$$

with a suitable penalty function by $\psi_r \left(x, u\right)$. Implemented is the augmented Lagrangian function

$$\psi_r \left(x, u\right) \; : = \; f(x) \; - \; \sum_{j \; \in \; J} \left[v_j g_j \left(x\right) - \tfrac{1}{2} r_j g_j \left(x\right)^2 \right] \; - \; \tfrac{1}{2} \sum_{j \; \in \; K} \frac{v_j^2}{r_j} \tag{7}$$

with $\qquad J \; : = \left\{ 1, \bullet \bullet \bullet , \; m_e \right\} \cup \; \left\{ j \; : \; m_e < \; j \; \leq \; m, \; g_j \left(x\right) \leq \; v_j / \, r_j \right\}$

and $K := \{1, \bullet\bullet\bullet, m\} \setminus J,$ cf. Schittkowski [45].

The objective function is *penalized* as soon as an iterate leaves the feasible domain. The corresponding penalty parameters r_j, $j = 1, \bullet\bullet\bullet, m$ that control the degree of constraint violation, must carefully be chosen to guarantee a descent direction of the merit function, see Schittkowski [45] or Wolfe [61] in a more general setting, i.e., to get

$$\phi'_{r_k}(0) = \nabla \psi_{r_k}(x_k, v_k)^T \begin{pmatrix} d_k \\ u_k - v_k \end{pmatrix} < 0 \qquad (8)$$

Finally one has to approximate the Hessian matrix of the Lagrangian function in a suitable way. To avoid calculation of second derivatives and to obtain a final super linear convergence rate, the standard approach is to update B_k, by the BFGS quasi-Newton formula, cf. Powell [37] or Stoer [58].

The implementation of a line search algorithm is a critical issue when implementing a nonlinear programming algorithm, and has significant effect on the overall efficiency of the resulting code. On the one hand we need a line search to stabilize the algorithm, on the other hand it is not desirable to waste too many function calls. Moreover, the behaviour of the merit function becomes irregular in case of constrained optimization because of very steep slopes at the border caused by large penalty terms. Even the implementation is more complex than shown above, if linear constraints and bounds of the variables are to be satisfied during the line search.

Usually, the step-length parameter α_k is chosen to satisfy the Armijo [1] condition

$$\phi_r(\sigma\beta^i) \leq \phi_r(0) + \sigma\beta^i \mu \phi'_{r_k}(0) \qquad (9)$$

see for example Ortega and Rheinboldt [33], The constants are from the ranges $0 < \mu < 0.5$, $0 < \beta < 1$, and $0 < \sigma \leq 1$. We start with $i = 0$ and increase i until (9) is satisfied for the first time, say at i_k. Then the desired steplength is $\alpha_k = \sigma\beta^{i_k}$.

Fortunately, SQP methods are quite robust and accept the steplength one in the neighbourhood of a solution. Typically the test parameter μ for the Armijo-type sufficient descent property (9) is

very small. Nevertheless the choice of the reduction parameter β must be adopted to the actual slope of the merit function. If β is too small, the line search terminates very fast, but on the other hand the resulting stepsizes are usually too small leading to a higher number of outer iterations. On the other hand, a larger value close to one requires too many function calls during the fine search. Thus, we need some kind of compromise, which is obtained by first applying a polynomial interpolation, typically a quadratic one, and use (9) only as a stopping criterion. Since $\phi_r(0)$, $\phi_r'(0)$, and $\phi_r(\alpha_i)$ are given, α_i the actual iterate of the line search procedure, we easily get the minimiser of the quadratic interpolation. We accept then the maximum of this value and the Armijo parameter as a new iterate, as shown by the subsequent code fragment implemented in NLPQLP.

Algorithm 2.1

Let β, μ *with* $0 < \beta < 1, \quad 0 < \mu < 0.5$ *be given*

Start: $\alpha_0 := 1$

For $i = 0, 1, 2, \bullet\bullet\bullet,$ *do*:

1) *If* $\phi_r(\alpha_i) < \phi_r(0) + \mu \alpha_i \phi_{r_k}'(0),$ *then stop.*

2) *Compute* $\alpha_i := \dfrac{0.5\, \alpha_i^2\ \phi_r'(0)}{\alpha_i \phi_{r_k}'(0) - \phi_r(\alpha_i) + \phi_r(0)}$

3) *Let* $\alpha_{i+1} := \max(\beta \alpha_i\ \overline{\alpha}_i).$

Corresponding convergence results are found in Schittkowski [45]. $\overline{\alpha}_i$ is the minimiser of the quadratic interpolation, and we use the Armijo descent property for checking termination. Step 3 is required to avoid irregular values, since the minimiser of the quadratic interpolation could be outside of the feasible domain $(0, 1]$. The search algorithm is implemented in NLPQLP together with additional safeguards, for example to prevent violation of bounds. Algorithm 4.1 assumes that $\phi_r(1)$ is known before calling the procedure, i.e., that the corresponding function values are given. We have to stop the algorithm, if sufficient descent is not observed after a certain number of iterations, say 10. If the tested stepsize falls below machine precision or the accuracy by which model function values are computed, the merit function cannot decrease further.

To outline the new approach, let us assume that functions can be computed simultaneously on l different machines. Then l test values $\alpha_i = \beta^{i-1}$ with $\beta = \varepsilon^{1/(l-1)}$ are selected, $i = 1, \bullet\bullet\bullet, l$. where ε is a guess for the machine precision. Next we require l parallel function calls to get the corresponding model function values. The first α_i satisfying a sufficient descent property (9), say for $i = i_k$ is accepted as the new steplength to set the subsequent iterate by $\alpha_k := \alpha_{i_k}$. One has to be sure that existing convergence results of the SQP algorithm are not violated.

The proposed parallel line search will work efficiently, if the number of parallel machines l is sufficiently large, and works as follows, where we omit the iteration index k.

Algorithm 2.2

Let β, μ *with* $0 < \beta < 1, \quad 0 < \mu < 0.5$ *be given.*

Start: *For* $\alpha_i = \beta^{i-1}$ *compute* $\phi_r(\alpha_i)$ *for* $i = 0, 1, 2, \bullet\bullet\bullet, l-1$

For $i = 0, 1, 2, \bullet\bullet\bullet$ *do*:

If $\phi_r(\alpha_i) < \phi_r(0) + \mu\alpha_i\phi_r'(0)$, *then stop.*

To precalculate l candidates in parallel at log-distributed points between a small tolerance $\alpha = \tau$ and $\alpha = 1$, $0 < \tau << 1$, we propose $\beta = \tau^{1/(l-1)}$.

The paradigm of parallelism is SPMD, i.e., Single Program Multiple Data. In a typical situation we suppose that there is a complex application code providing simulation data, for example by an expensive Finite Element calculation in mechanical structural optimisation. It is supposed that various instances of the simulation code providing function values, are executable on a series of different machines, so-called slaves, controlled by a master program that executes NLPQLP. By a message passing system, for example PVM, see Geist et al. [16], only very few data need to be transferred from the master to the slaves. Typically only a set of design parameters of length n must to be passed. On return, the master accepts new model responses for objective function and constraints, at most $m+1$ double precision numbers. All massive numerical calculations and model data, for example the stiffness matrix of a Finite Element model in a mechanical engineering application, remain on the slave processors of the distributed system.

In both situations, i.e., the serial or parallel version, it is still possible that Algorithm 2.1 or Algorithm 2.2 breaks down because to too many iterations. In this case, we proceed from a descent direction of the merit function, but $\phi_r'(0)$ is extremely small. To avoid interruption of the whole iteration process, the idea is to repeat the line search with another stopping criterion. Instead of testing (9), we accept a stepsize α_k as soon as the inequality

$$\phi_{r_k}(\alpha_k) \leq \max_{k-p(k)\leq j\leq k}\left[\phi_{r_j}(0) + \alpha_{i_k}\mu\phi'_{r_k}(0)\right] \qquad (10)$$

is satisfied, where $p(k)$ is a predetermined parameter with $p(k) = \min[k, p]$, p a given tolerance. Thus, we allow an increase of the reference value $\phi_{r_{j_k}}(0)$ in a certain error situation, i.e., an increase of the merit function value. To implement the non-monotone line search, we need a queue consisting of merit function values at previous iterates. In case of $k = 0$, the reference value is adapted by a factor greater than 1, i.e., $\phi_{r_{j_k}}(0)$ is replaced by $t\phi_{r_{j_k}}(0)$, $t > 1$.

The basic idea to store reference function values and to replace the sufficient descent property by a sufficient 'ascent' property in max-form, is for example described in Dai [11], where a general convergence proof for the unconstrained case is presented. The general idea goes back to Grippo, Lampariello, and Lucidi [18], and was extended to constrained optimization and trust region methods in a series of subsequent papers, see Bonnans et al. [4], Deng et al. [13], Grippo et al. [19, 20], Ke and Han [26], Ke et al. [27], Lucidi et al. [30], Panier and Tits [34], Raydan [39], and Toint [59, 60]. However, there is a difference in the methodology: Our goal is to allow monotone line searches as long as they terminate successfully, and to apply a non-monotone one only in an error situation.

The final step of an SQP method consists of updating the quasi-Nerwton Matrix B_k, e.g., by the BFGS formula

$$B_k := B_k + \frac{q_k q_k^T}{p_k^T q_k} - \frac{B_k p_k p_k^T B_k}{p_k^T B_k p_k} , \qquad (11)$$

where $q_k := \nabla_x L(x_{k+1}, u_k) - \nabla_x L(x_k, u_k)$ and $p_k := x_{k+1} - x_k$. Special safeguards guarantee that $p_k^T q_k > 0$ and that thus all matrices B_k r remain positive definite provided that B_0 is positive definite. A possible scaling factor and restart procedure is to replace an actual B_k by $\gamma_k I$ before performing the update (11), where $\gamma_k = \frac{p_k^T q_k}{p_k^T p_k}$ and where I denotes the identity matrix, see for example Liu and Nocedal [29]. Scaled restarts are recommended if, e.g., the convergence turns out to become extremely slow.

3 Performance Evaluation

3.1 The Test Environment

Our numerical tests use the 306 academic and real-life test problems published in Hock and Schittkowski [25] and in Schittkowski [48]. Part of them are also available in the Cute library, see Bongartz et. al [7], and their usage is described in Schittkowski [55].

Since analytical derivatives are not available for all problems, we approximate them numerically. The test examples are provided with exact solutions, either known from analytical precalculations *by hand* or from the best numerical data found so far.

First we need a criterion to decide whether the result of a test run is considered as a successful return or not. Let $\varepsilon > 0$ be a tolerance for defining the relative accuracy, x_k the final iterate of a test run, and x^* the supposed exact solution known from the test problem collection. Then we call the output a successful return, if the relative error in the objective function is less than e and if the maximum constraint violation is less than ε^2 i.e., if

$$f(x_k) - f(x^*) < \varepsilon |f(x^*)|, \quad if\ f(x^*) \neq 0$$

or

$$f(x_k) < \varepsilon, \quad if\ f(x^*) = 0$$

and

$$r(x_k) = \left\| g(x_k)^- \right\|_\infty < \varepsilon^2 ,$$

where $\| \bullet \bullet \bullet \|_\infty$ denotes the maximum norm and $g(x_k)^- = \min[0, g(x_k)]$, $j > m_e$ and $g(x_k)^- = g(x_k)$ otherwise.

We take into account that a code returns a solution with a better function value than the known one, subject to the error tolerance of the allowed constraint violation. However, there is still the possibility that an algorithm terminates at a local solution different from the known one. Thus, we call a test run a successful one, if in addition to the above decision the internal termination conditions are satisfied subject to a reasonably small tolerance (IFAIL = 0), and if

$$f(x_k) - f(x^*) \geq \varepsilon \left| f(x^*) \right|, \quad if\ f(x^*) \neq 0$$

or

$$f(x_k) \geq \varepsilon, \quad if\ f(x^*) = 0$$

and

$$r(x_k) < \varepsilon^2 ,$$

For our numerical tests, we use $\varepsilon = 0.01$ to determine a successful return, i.e., we require a final accuracy of one percent. Note that in all cases, NLPQLP is called with a termination tolerance of 10^{-7}.

If gradients are not available in analytical form, they must be approximated in a suitable way. The three most popular difference formulae are the following ones:

1. **Forward differences:**

$$\frac{\partial}{\partial x_i} f(x) \approx \frac{1}{\eta_i} \left[f(x + \eta_i e_i) - f(x) \right] \tag{12}$$

2. **Two-sided differences:**

$$\frac{\partial}{\partial x_i} f(x) \approx \frac{1}{2\eta_i} \left[f(x + \eta_i e_i) - f(x - \eta_i e_i) \right] \tag{13}$$

3. **Forth-order formula:**

$$\frac{\partial}{\partial x_i} f(x) \approx \frac{1}{4!\,\eta_i} \left[2f(x - 2\eta_i e_i) - 16f(x - \eta_i e_i) + 16f(x + \eta_i e_i) - 2f(x + 2\eta_i e_i) \right] \tag{14}$$

Here $\eta_i = \eta \max\left(10^{-5}, |x_i|\right)$ and e_i is the $i-th$ unit vector, $i = 1, \bullet\bullet\bullet, n$. The tolerance η depends on the difference formula and is set to $\eta = \eta_m^{1/2}$ for forward differences, $\eta = \eta_m^{1/3}$ for two-sided differences, and $\eta = \left(\eta_m / 72\right)^{1/4}$ for fourth-order formulae. η_m is a guess for the accuracy by which function values are computed, i.e., either machine accuracy in case of

analytical formulae or an estimate of the noise level in function computations. In a similar way, derivatives of constraints are computed.

The Fortran implementation of the SQP method introduced in the previous section, is called NLPQLP. The code represents the most recent version of NLPQL which is frequently used in academic and commercial institutions. NLPQLP is prepared to run also under distributed systems, but behaves in exactly the same way as the serial version, if the number of simulated processors is set to one. Functions and gradients must be provided by reverse communication and the quadratic programming sub-problems are solve by the primal-dual method of Goldfarb and Idnani [17] based on numerically stable orthogonal decompositions. NLPQLP is executed with termination accuracy ACC = 10^{-7} as mentioned already above, and a maximum number of iterations MAXIT = 500.

In the subsequent tables, we use the notation

- n_{succ} - number of successful test runs (according to above definition)
- n_{func} - average number of function evaluations
- n_{grad} - average number of gradient evaluations or iterations, respectively
- $f(x)$ - final objective function value
- $r(x)$ - final constraint violation
- i_{fail} - failure code

To get n_{func} or n_{grad}, we count each evaluation of a whole set of function or gradient values, respectively, for a given iterate x_k also in the case of several simulated processors, $l > 0$. However, additional function evaluations needed for gradient approximations, are not counted for n_{func}. Their average number is n_{func} for forward differences, $2 \times n_{func}$ for two-sided differences, and $4 \times n_{func}$ for fourth-order formulae. One gradient computation corresponds to one iteration of the SQP method.

The Fortran codes are compiled by the Intel Visual Fortran Compiler, Version 10.1, EM64T, under Windows XP64 and Dual Core AMD Opteron Processor 265, 1.81 GHz, with 8 GB of RAM.

3.2 Testing Distributed Function Calls

First we investigate the question, how parallel line searches influence the overall performance. Table 1 shows the number of successful test runs and the average number of iterations or gradient evaluations, n_{it} for an increasing number of simulated parallel calls of model functions denoted by l. The forward difference formula (12) is used for gradient approximations and non-monotone line search is applied with a queue size of $p = 30$. Calculation time is about one second for solving all 306 test problems without random perturbations.

$l = 1$ corresponds to the sequential case, when Algorithm 2.1 is applied to the line search consisting of a quadratic interpolation combined with an Armijo-type bisection strategy and a non-monotone stopping criterion.

l	n_{succ}	n_{grad}	l	n_{succ}	n_{grad}
1	306	23	8	299	39
3	224	177	9	301	32
4	242	170	10	302	31
5	268	114	15	303	23
6	289	75	20	303	22
7	297	45	50	303	22

Table 1: Performance Results for Parallel Line Search

In all other cases, $l > 1$ simultaneous function evaluations are made according to Algorithm 2.2. To get a reliable and robust line search, one should use at least seven parallel processors. No significant improvements are observed, if we evaluate more than ten functions in parallel.

The most promising possibility to exploit a parallel system architecture occurs, when gradients cannot be calculated analytically, but have to be approximated numerically, for example by forward differences, two-sided differences, or even higher order methods. Then we need at least n additional function calls, where n is the number of optimisation variables, or a suitable multiple of n.

3.3 Function Evaluations and Gradient Approximations by a Difference Formulae Under Random Noise

For our numerical tests, we apply the forth-order difference formula (14). To test the stability of the formula and the underlying SQP code, we add randomly generated noise to each function value. Non-monotone line search is applied with a queue length of p = 30 in error situations, and the serial line search calculation by Algorithm 2.1 is used. Moreover, the BFGS quasi-Newton updates are restarted with ρI if a descent direction cannot be computed.

To compare the different stabilisation approaches, we apply three different scenarios:

Table 2 - monotone line search, no restarts

Table 3 - non-monotone line search, no restarts

Table 4 - non-monotone line search and restarts

ε_{err}	n_{succ}	n_{func}	n_{grad}
0	304	35	22
10^{-12}	303	36	22
10^{-10}	297	40	23
10^{-8}	293	43	23
10^{-6}	280	57	24
10^{-4}	243	74	26
10^{-2}	133	133	32

Table 2: Test Results for Monotone Line Search without Restarts

ε_{err}	n_{succ}	n_{func}	n_{grad}
0	306	38	22
10^{-12}	303	37	23
10^{-10}	300	43	25
10^{-8}	300	53	24
10^{-6}	295	71	25
10^{-4}	268	116	30
10^{-2}	294	185	33

Table 3: Test Results for Non-Monotone Line Search without Restarts

ε_{err}	n_{succ}	n_{func}	n_{grad}
0	306	38	22
10^{-12}	305	39	23
10^{-10}	300	47	24
10^{-8}	303	83	27
10^{-6}	302	105	30
10^{-4}	297	318	43
10^{-2}	279	647	64

Table 4: Test Results for Non-Monotone Line Search and Restarts

The corresponding results are evaluated for increasing random perturbations (ε_{err}). More precisely, if ρ denotes a uniformly distributed random number between 0 and 1, we replace $f(x_k)$ by $f(x_k)\left[1 + \varepsilon_{err}(2\rho - 1)\right]$ at each iterate x_k. In the same way, restriction functions are perturbed. The tolerance for approximating gradients, η_m is set to the machine accuracy in case of $\varepsilon_{err} = 0$, and to the random noise level otherwise.

The numerical results are surprising and depend heavily on the new non-monotone line search strategy and the additional stabilisation procedures. We are able to solve about 90% of the test examples in case of extremely noisy function values with at most one correct digit in partial derivative values. However, the stabilization process is costly. The more test problems are successfully solved, the more iterations, especially function evaluations, are needed.

3.4 Testing Scaled Restarts

In some situations, the convergence of an SQP method becomes quite slow for many reasons, e.g., badly scaled variables or functions, inaccurate derivatives, or inaccurate solutions of the quadratic program (4). In these situations, errors in the search direction or the partial derivatives influence the update procedure (11) and the quasi-Newton matrices B_k are getting more and more inaccurate.

A frequently proposed remedy is to restart the update algorithm by replacing the actual matrix B_k by the initial matrix B_0 or any similar one, if more information is available. One possibility is to multiply a special scaling factor with the identity matrix, i.e., to let $B_k := \gamma_k I$ for selected

iterates k, where $\gamma_k := \dfrac{p_k^T q_k}{p_k^T p_k}$ and where I denotes the identity matrix, see for example Liu and Nocedal [29]. $q_k := \nabla_x L\left(x_{k+1}, u_k\right) - \nabla_x L\left(x_k, u_k\right)$ and $p_k := x_{k+1} - x_k$.

Scaled restarts are recommended if convergence turns out to become extremely slow. To illustrate the situation, we consider a few test runs where the examples are generated by discretising a two-dimensional elliptic partial differential equation, see Maurer and Mittelmann [31, 32]. The original formulation is that of an optimal control problem where the state and control variables are both discretised.

From a total of 13 original test cases, we select five problems that could not be solved by NLPQLP as efficiently as expected with standard solution tolerances. Depending on the grid size in our case 20 in each direction, we get problems with $n = 722$ or $n = 798$ variables, respectively, and $m_e = 361$ or $m_e = 437$ nonlinear equality constraints. They are obtained by applying the five-star formula to approximate second partial derivatives.

Tables 5 to 8 contain numerical results first for standard tolerances and MODE = 0, where ACC is set to 10^{-7} in all cases. For the results of the remaining tables, We used MODE = 2, MODE = 7, and MODE = 20. MODE = 2 means that the scaled restart is applied at the very first step. Note that also for all other test cases with MODE > 0, the initial BFGS matrix is $B_0 = \gamma_0 I$.

problem	n	m_e	n_{func}	n_{grad}	$f(x)$	$r(x)$	i_{fail}
EX 1	722	361	64	64	0.45903100E-1	0.33E-10	0
EX 2	722	361	109	109	0.40390974E-1	0.22E-8	0
EX 3	722	361	88	88	0.11009561E+0	0.49E-9	0
EX 4	798	437	113	113	0.75833416E-1	0.12E-9	0
EX 5	798	437	200	200	0.51376012E-1	0.60E-5	1

Table 5: Test Results for Scaled Restarts: MODE = 0

problem	$n\,m_e$	n_{func}	n_{grad}	$f(x)$	$r(x)$	i_{fail}	
EX 1	722	361	64	64	0.45903100E-1	0.64E-11	0
EX 2	722	361	108	108	0.40390974E-1	0.21E-8	0
EX 3	722	361	75	75	0.11009568E+0	0.46E-9	0
EX 4	798	437	108	108	0.75833417E-1	0.63E-9	0
EX 5	798	437	200	200	0.51369466E-1	0.14E-6	1

Table 6: Test Results for Scaled Restarts: MODE = 2

problem	$n\,m_e$	n_{func}	n_{grad}	$f(x)$	$r(x)$	i_{fail}	
EX 1	722	361	20	20	0.45903160E-1	0.19E-7	0
EX 2	722	361	21	21	0.40390974E-1	0.79E-7	0
EX 3	722	361	41	41	0.11009561E+0	0.23E-9	0
EX 4	798	437	58	58	0.75833423E-1	0.44E-8	0
EX 5	798	437	112	112	0.51365403E-1	0.15E-12	0

Table 7: Test Results for Scaled Restarts: MODE = 7

problem	$n\,m_e$	n_{func}	n_{grad}	$f(x)$	$r(x)$	i_{fail}	
EX 1	722	361	50	50	0.45903100E-1	0.68E-8	0
EX 2	722	361	33	33	0.40390974E-1	0.22E-8	0
EX 3	722	361	36	36	0.11009561E+0	0.23E-7	0
EX 4	798	437	61	61	0.75833414E-1	0.17E-8	0
EX 5	798	437	75	75	0.51365398E-1	0.14E-7	0

Table 8: Test Results for Scaled Restarts: MODE = 20

The error codes are the same as produced by NLPQLP through the parameter IFAIL, i.e., IFAIL = 0 for successful termination and IFAIL = 1 for reaching the upper limit of 200 iterations.

We observe a significant speedup for scaled restarts every seven iterations. Initial scaling and more then 7 restarts do not yield any better results.

4 Program Documentation

NLPQLP is implemented in form of a Fortran subroutine. The quadratic programming problem is solved by the code QL, an implementation of the primal-dual method of Goldfarb and Idnani [17] going back to Powell [38], see also Schittkowski [53] for more details about implementation and usage. Model functions and gradients must be provided by reverse communication. The user has to evaluate function and gradient values in the same program that executes NLPQLP, according to the following rules:

1. Choose starting values for the variables to be optimised, and store them in the first column of an array called X.
2. Compute objective and all constraint function values, store them in XF(1) and the first column of G, respectively.
3. Compute gradients of objective function and all constraints, and store them in DF and DG, respectively. The J-th row of DG contains the gradient of the J-th constraint, $J = 1, \bullet \bullet \bullet, M$.
4. Set IFAIL = 0 and execute NLPQLP.
5. If NLPQLP returns with IFAIL = -1, compute objective and constraint function values for all variables found in the first L columns of X, store them in F (first L positions) and G (first L columns), and call NLPQLP again.
6. If NLPQLP terminates with IFAIL = -2, compute gradient values with respect to the variables stored in the first column of X, and store them in DF and DG. Only derivatives for active constraints, ACT(J) = .TRUE., need to be computed. Then call NLPQLP again.
7. If NLPQLP terminates with IFAIL = 0, the internal optimality criteria are satisfied. In the case of IFAIL > 0, an error has occurred.

If analytical derivatives are not available, simultaneous function calls can be used for gradient approximations, for example by forward differences $2N > L$ two-sided differences $4N > L \geq 2N$, or even higher order formulae $L \geq 4N$.

Usage:

```
      CALL NLPQLP(    L,      M,     ME,   MMAX,      N,
     /                NMAX,   MNN2,  X,    F,         G,
     /                DF,     DG,    U,    XL,        XU,
     /                C,      D,     ACC,  ACCQP,     STPMIN,
     /                MAXFUN, MAXIT, MAXNM, RHOB,     IPRINT,
     /                MODE,   IOUT,  IFAIL, WA,        LWA,
     /                KWA,    LKWA,  ACT,   LACT,      LQL,
     /                QPSLVE,                          )
```

Definition of the parameters:

L :	Number of parallel systems, i.e., function calls during line search at predetermined iterates.
M :	Total number of constraints.
ME :	Number of equality constraints.
MMAX :	Row dimension of array DG containing Jacobian of constraints. MMAX must be at least one and greater or equal to M.
N :	Number of optimisation variables.
NMAX :	Row dimension of C. NMAX must be at least two and greater than N.
MNN2 :	Must be equal to M + N +N +2 when calling NLPQLP.
X(NMAX, L):	Initially, the first column of X has to contain starting values for the optimal solution. On return, X is replaced by the current iterate. In the driving program the row dimension of X has to be equal to NMAX. X is used internally to store L different arguments for which function values should be computed simultaneously.
F(L):	On return, F(1) contains the final objective function value. F is used also to store L different objective function values to be computed from L sets of arguments stored in X.
G(MMAX, L):	On return, the first column of G contains the constraint function values at the final iterate X. In the driving program, the row dimension of G has to be equal to MMAX. G is used internally to store L different sets of constraint function values to be computed from L sets of argumentsstored in X.
DF(NMAX):	DF contains the current gradient of the objective function. In case of numerical differentiation and a distributed system (L > 1) , it is recommended to apply parallel evaluations of F to compute DF.

U(MNN2): U contains the multipliers with respect to the actual iterate stored in the first column of X. The first M locations contain the multipliers of the M nonlinear constraints, the subsequent N locations the multipliers of the lower bounds, and the final N locations the multipliers of the upper bounds. At an optimal solution, all multipliers with respect to inequality constraints should be nonnegative

XL (N), XU (N): On input, the one-dimensional arrays XL and XU must contain the lower and upper bounds of the variables, respectively.

C(NMAX, NMAX): On return, C contains the last computed approximation of the Hessian matrix of the Lagrangian function. C is stored in form of an Cholesky decomposition, is LQL is set to false, see below. In this case, C contains the lower triangular factor of an LDL factorization of the final quasi-Newton matrix (without diagonal elements, which are always one). In the driving program, the row dimension of C has to be equal to NMAX.

D (NMAX): The elements of the diagonal matrix of the LDL decomposition of the quasi-Newton matrix are stored in the one-dimensional array D, if LQL is false.

ACC : The user has to specify the desired final accuracy (e.g. 1.0D-7). The termination accuracy should not be much smaller than the accuracy by which gradients are computed.

ACCQP : The tolerance is needed for the QP solver to perform several tests, for example whether optimality conditions are satisfied or whether a number is considered as zero or not. If ACCQP is less or equal to zero, then the machine precision is computed by NLPQLP and subsequently multiplied by 1.0D+4.

STPMIN : Minimum steplength in case of L > 1. Recommended is any value in the order of the accuracy by which functions are computed. The value is needed to compute a steplength reduction factor by $\text{STPMIN} ** \left(1/\left(\text{L}-1\right)\right)$.

If STPMIN $\leq$ 0, then STPMIN = ACC is used.

MAXFUN : The integer variable defines an upper bound for the number of function calls during the line search (e.g., 20). MAXFUN is only needed in case of L = 1, and must not be greater than 50.

MAXIT : Maximum number of outer iterations, where one itera\tion corresponds to one formulation and solution of the quadratic programming subproblem, or, alternatively, one evaluation of gradients (e.g., 100).

MAXNM : Stack size for storing merit function values at previous iterations for non-monotone line search (e.g., 10). If MAXNM = 0, a monotone line search is performed. MAXNM should not be greater than 50.

RHOB : Parameter for performing a restart in case of IFAIL = 2 by setting the BFGS-update matrix to RHOB * I, where I denotes the identity matrix. The number of restarts is bounded by MAXFUN. A value greater than one is recommended. (e.g., 100).

IPRINT : Specification of the desired output level.

0 - No output of the program.

1 - Only final convergence analysis.

2 - One line of intermediate results for each iteration.

3 - More detailed information for each iteration.

4 - More line search data is displayed.

Note that constraint and multiplier values are not displayed for N, M > 1000.

MODE : The parameter specifies the desired version of NLPQLP.

0 - Normal execution (reverse communication!).

1 - Initial guess for multipliers in U and Hessian of the Lagrangian function in C and D provided. In case of LQL = .TRUE., D is ignored. Otherwise, the lower part of C has to contain the lower triangular factor of an LDL decomposition and D the diagonal part.

2 - Initial scaling (Oren-Luenberger) after first step, BFGS updates started from multiple of identity matrix.

3 - Scaled restart, if scaling parameter is less than square root of ACC.

>3 - Initial and repeated scaling every MODE steps, reset of BFGS matrix to multiple of identity matrix.

IOUT : Integer indicating the desired output unit number, i.e., all write statements start with 'WRITE(IOUT, ...)'.

IFAIL : The parameter shows the reason for terminating a solution process. Initially, IFAIL must be set to zero. On return, IFAIL could contain the following values:

-2 - Compute new gradient values.

-1 - Compute new function values.

0 - Optimality conditions satisfied.

1 - Stop after MAXIT iterations

2 - Uphill search direction.

3 - Underflow when computing new BFGS-update matrix.

4 - Line search exceeded MAXFUN iterations.

5 - Length of a working array too short.

6 - False dimensions, if M > MMAX, N ≥ NMAX, or MNN2 ≠ M + N + N + 2.

7 - Search direction close to zero at infeasible iterate.

8 - Starting point violates lower or upper bound.

9 - Wrong input parameter, e.g., MODE, IPRINT, IOUT.

10 - Inconsistency in QP, division by zero.

>100 - Error message of QP solver.

WA(LWA): WA is a double precision working array of length LWA. On return, the first N positions contain the best feasible iterate obtained, WA(N+1) the corresponding objective function value, and the subsequent M positions the constraint values. If no intermediate feasible solution exists, WA(N+1) contains a large value, e.g., 1.0D+72.

LWA : Length of WA, has to be at least at least 23*N+4*M+3*MMAX+150.

NOTE: The standard QP-solver coming together with NLPQLP (QL). needs additional memory for 3*NMAX*NMAX/2+10*NMAX+MMAX+N+1 double precision numbers.

KWA (LKWA): KWA is an integer working array of length LKWA. On return, the first 5 positions contain the following information.

KWA(1) - Number of function evaluations.

KWA(2) - Number of gradient evaluations.

KWA(3) - Iteration count.

KWA(4) - Number of QP's solved.

KWA(5) - Flag for better feasible, but non-stationary iterate (=1) or not (=0), see below.

LKWA : Length of KWA, has to be at least at least 20.

NOTE: The standard QP-solver coming together with NLPQLP (QL) needs additional memory for N double precision numbers.

ACT(LACT K): The logical array indicates constraints, which NLPQLP considers to be active at the last computed iterate, i.e., G (J, 1) is active, if and only if ACT (J) is true for $J = 1, \bullet\bullet\bullet, M$.

LACT : Length of ACT, has to be at least 2*M + 10.

LQL : If LQL is set to true in the calling program, the quadratic programming problem is solved proceeding from a full positive definite quasi-Newton matrix. Otherwise, a Cholesky decomposition (LDL) is performed and updated internally, so that matrix C always consists of the lower triangular factor and D of the diagonal

QPSLVE : External subroutine to solve the quadratic programming subproblem. The calling sequence is

```
CALL QPSLVE(  M,    ME,   MMAX,    N,  NMAX,
/           MNN,    C     D,       A,    B,
/           XL,     XU,   X,       U,    EPS,
/           MODE,   IOUT, IFAIL,   IPRINT, WAR,
/           LWAR,   IWAR, LIWAR            )
```

For more details about the choice and dimensions of arguments, see [53].

Some of the termination reasons depend on the accuracy used for approximating gradients. If we assume that all functions and gradients are computed within machine precision and that the implementation is correct, there remain only the following possibilities that could cause an error message:

1. The termination parameter ACC is too small, so that the numerical algorithm plays around with round-off errors without being able to improve the solution. Especially the Hessian approximation of the Lagrangian function becomes unstable in this case. A straightforward remedy is to restart the optimization cycle again with a larger stopping tolerance.

2. The constraints are contradicting, i.e., the set of feasible solutions is empty. There is no way to find out, whether nonlinear and nonconvex constraints are feasible or not. Thus, the nonlinear programming algorithms will proceed until running in any of the mentioned error situations. In this case, the correctness of the model must be carefully checked.

3. Constraints are feasible, but active constraints are degenerate, e.g., redundant. One should know that SQP algorithms assume the satisfaction of the so-called linear independency constraint qualification, i.e., that gradients of active constraints are linearly independent at each iterate and in a neighborhood of an optimal solution. In this situation, it is recommended to check the formulation of the model constraints.

However, some of the error situations also occur if, because of wrong or non-accurate gradients, the quadratic programming subproblem does not yield a descent direction for the underlying merit function. In this case, one should try to improve the accuracy of function evaluations, scale the model functions in a proper way, or start the algorithm from other initial values.

Since Version 2.1, NLPQLP returns the best iterate obtained. In case of successful termination (IFAIL = 0), this is always the last one. But it might be possible that in an exceptional situation, an intermediate iterate is feasible with a better objective function value than that one of the final iterate, but the KKT optimality conditions axe not satisfied. In this case, the better feasible solution is stored at the first n positions of the double precision working array and the corresponding objective function value at position $n+1$. Moreover, positions $n+2$ to $n+1+m$ contain the constraint values. Note that feasibility is tested by sum of constrained violations tested against ACC.

On successful return with IFAIL = 0, KWA(5) is set to zero. If, however, a better feasible objective function value has been found during the first five iterations, then KWA(5) is set to 1, the BFGS-update matrix C is set to ρI with $\rho < 1$, where I denotes the identity matrix. The corresponding formal argument of NLPQLP is called RHOB. Moreover, the multiplier approximation vector U is set to 0. Thus, an immediate restart under control of the user is possible with MODE = 1. Some information is printed on the standard IO channel in case of IPRINT > 0. For compatibility reasons with previous versions, RHOB replaces TOLNM and is set to zero for all input values less than one.

The QP solver is defined in form of an external subroutine to allow a replacement in case of exploiting special sparsity patterns. A typical example is the usage of NLPQLP for solving least squares problems, where artificially introduced equality constraints lead to a Jacobian which consist partially of the identity matrix, see Schittkowski [50, 51].

The internal scaling and restart option is borrowed from limited-memory quasi-Newton methods, see for example Liu and Nocedal [29]. If requested by the user, the quasi-Newton matrix is replaced by a scalar multiple of the identity matrix just before updating. Either an initial scaling or a reset of the whole matrix and computation of a new scaling parameter is performed depending on the input parameter MODE. A scaled restart is recommended, if, e.g., the convergence turns out to become extremely slow.

5 Examples

To give an example how to organize the code, we consider Rosenbrock's post office problem, i.e., test problem TP37 of Hock and Schittkowski [25].

$$x_1, x_2 \in IR: \quad \begin{cases} \min -x_1 x_2 x_3 \\ x_1 + 2x_2 + 2x_3 \geq 0 \\ 72 - x_1 - 2x_2 - 2x_3 \geq 0 \\ 0 \leq x_1 \leq 42 \\ 0 \leq x_2 \leq 42 \\ 0 \leq x_3 \leq 42 \end{cases} \tag{15}$$

NLPQLP comes with a couple of demo programs by which the following situations are to be illustrated:

File name	Comments
nlp_demoA.for	numerical differentiation and distributed function calls
nlp_demoB.for	numerical differentiation
nlp_demoC.for	numerical derivatives
nlp_demoD.for	warm and cold restarts
nlp_demoE.for	simultaneous function and gradient evaluation
nlp_demoF.for	active set strategy
nlp_demoG.for	active set strategy

A Fortran source code for a typical situation is listed below. Gradients are approximated by forward differences. The function block inserted in the main program can be replaced by a subroutine call. Also the gradient evaluation is easily exchanged by an analytical one or higher order derivatives

```
      IMPLICIT          NONE
      INTEGER           NMAX, MMAX, LMAX, MNN2X, LWA, LKWA, LACTIV
      PARAMETER (       NMAX = 4,
     /                  MMAX = 2,
     /                  LMAX = 10,
     /                  MNN2X = MMAX + NMAX + NMAX + 2,
     /                  LWA = 1.5*NMAX*NMAX + 33*NMAX + 9*MMAX + 200,
     /                  LKWA = NMAX + 10,
     /                  LACTIV = 2+MMAX + 10)
      INTEGER           KWA(LKWA), N, ME, M, L, MNN2, MAXIT, MAXFUN,
     /                  IPRINT, MAXNM, IOUT, MODE, IFAIL, I, J, K, NFUNC
      DOUBLE PRECISION  X(NMAX,LMAX), F(LMAX), G(MMAX,LMAX), DF(NMAX),
     /                  DG(MMAX,NMAX), U(MNN2X), XL(NMAX), XU(NMAX),
     /                  C(NMAX,NMAX), D(NMAX), WA(LWA), ACC, ACCQP,
```

```
      /                 STPMIN, EPS, EPSREL, FBCK, GBCK(MMAX), XBCK,
      /                 RHOB
       LOGICAL          ACTIVE(LACTIV), LQL
       EXTERNAL         QL
C
C  Set some constants and initial values
C
       IOUT    = 6
       ACC     = 1.0D-8
       ACCQP   = 1.0D-12
       STPMIN  = 1.0D-10
       EPS     = 1.0D-7
       MAXIT   = 100
       MAXFUN  = 10
       MAXNM   = 10
       RHOB    = O.ODO
       LQL     = .TRUE.
       IPRINT  = 2
       N       = 3
       L       = N
       M       = 2
       ME      = 0
       MNN2    = M + N + N + 2
       MODE    = 0
       IFAIL   = 0
       NFUNC   = 0
       DO I=1,N
          DO K=1,L
             K(I,K) = 10.0D0
          ENDDO
             XL(I) = 0.0D0
             XU(I) = 42.0D0
       ENDDO
      1CONTINUE
C=====================================================================
C  This is the main block to compute all function value
C  simultaneously, assuming that there are L nodes.
C  The block is executed either for computing a steplength
C  or for approximating gradients by forward differences.
C
       DO K=1,L
          F(K)   = -X(1,K)*X(2,K)*X(3,K)
          G(1,K) = X(1,K) + 2.0D0*X(2,K) + 2.0D0*X(3,K)
          G(2,K) = 72.0D0 - X(1,K) - 2.0D0*X(2,K) - 2.0D0*X(3,K)
       ENDDO
C
C=====================================================================
       NFUNC = NFUNC + 1
       IF (IFAIL.EQ.-1) GOTO 4
       IF (NFUNC.GT.l) GOTO 3
      2CONTINUE
       FBCK = F(l)
       DO J=1,M
          GBCK(J) = G(J,1)
       ENDDO
       XBCK = X(l,l)
       DO I=1,N
          EPSREL = EPS*DMAX1(1.0D0,DABS(X(I,1)))
          DO K=2,L
             X(I,K) = X(I,1)
          ENDDO
          X(I,I) = X(I,1) + EPSREL
       ENDDO
```

```
        GOTO 1
       3 CONTINUE

        X(l,l) = XBCK
        DO 1=1,N
           EPSREL = EPS*DMAX1(1.ODO,DABS(X(I,1)))
           DF(I) = (F(I) - FBCK)/EPSREL
           DO J=1,M
              DG(J,I) = (G(J,I) - GBCK(J))/EPSREL
           ENDDO
        ENDDO
        F(l) = FBCK
        DO J=1,M
           G(J,1) = GBCK(J)
        ENDDO
C
       4 CONTINUE
        CALL NLPQLP (      L,        M,      ME,     MMAX,        N,
       /                 NMAX,     MNN2,       X,        F,        G,
       /                   DF,       DG,       U,       XL,       XU,
       /                    C,        D,     ACC,    ACCQP,   STPMIN,
       /               MAXFUN,    MAXIT,   MAXNM,     RHOB,   IPRINT,
       /                 MODE,     IOUT,   IFAIL,       WA,      LWA,
       /                  KWA,     LKWA,  ACTIVE,   LACTIV,      LQL,
       /                   QL)
        IF (IFAIL.EQ.-1) GOTO 1
        IF (IFAIL.EQ.-2) GOTO 2
C
        WRITE(IOUT,1000) NFUNC
   1000 FORMAT(' *** Number of function calls: ',13)
C
        STOP
        END
```

The following output should appear on screen:

```
-----------------------------------------------------------------
START OF THE SWQUENTIAL QUADRATIC PROGRAMMING ALGORITHM
-----------------------------------------------------------------


Parameters:
   N       =        3
   M       =        2
   ME      =        0
   MODE    =        0
   ACC     =   0.1000D-07
   ACCQP   =   0.1000D-11
   STPMIN  =   0.1000D-09
   MAXFUN  =        3
   MAXNM   =       10
   MAXIT   =      100
   IPRINT  =        2

Output in the following order:
   IT    - iteration number
   F     - objectivefunction value
   SCV   - sum of constraint violations
   NA    - number ofactive constraints
   I     - number of line search iterations
   ALPHA - steplength parameter
   DELTA - additional variable to prevent inconsistency
   KKT   - Karush-Kuhn-Tucker optimality criterion
```

```
  IT         F           SCV     NA  I    ALPHA     DELTA      KKT
 ------------------------------------------------------------------------------
   1 -0.10000000D+04  0.00D+00    2  0  0.00D+00  0.00D+00  0.44D+04
   2 -0.23625000D+04  0.64D-07    1  1  0.10D+01  0.00D+00  0.11D+04
   3 -0.32507304D+04  0.11D-13    1  1  0.10D+01  0.00D+00  0.69D+03
   4 -0.33041403D+04  0.11D-13    1  1  0.10D+01  0.00D+00  0.36D+03
   5 -0.34527380D+04  0.11D-13    1  1  0.10D+01  0.00D+00  0.58D+01
   6 -0.34559625D+04  0.00D+00    1  1  0.10D+01  0.00D+00  0.10D+00
   7 -0.34559625D+04  0.00D+00    1  2  0.10D-04  0.00D+00  0.23D+00
   8 -0.34559625D+04  0.00D+00    1  2  0.10D-04  0.00D+00  0.76D-01
   9 -0.34560000D+04  0.17D-10    1  1  0.10D+01  0.00D+00  0.24D-04
  10 -0.34560000D+04  0.48D-12    1  1  0.10D+01  0.00D+00  0.20D-07
  11 -0.34560000D+04  0.11D-13    1  1  0.10D+01  0.00D+00  0.25D-11

--- Final Convergence Analysis at Last Iterate ---

      Best result at iteration:         ITER =       11
      Objective function value:         F(X) =  -0.34560000D+04
      Solution values:                  X    =
         0.24000000D+02  0.12000000D+02    0.12000000D+02
      Distances from lower bounds:      X-XL =
         0.24000000D+02  0.12000000D+02    0.12000000D+02
      Distances from upper bounds:      XU-X =
         0.18000000D+02  0.30000000D+02    0.30000000D+02
      Multipliers for lower bounds:     U    =
         0.00000000D+00  0.00000000D+00    0.00000000D+00
      Multipliers for upper bounds:     U    =
         0.00000000D+00  0.00000000D+00    0.00000000D+00
      Constraint values:                G(X) =
         0.72000000D+02 -0.10658141D-13
      Multipliers for constraints:      U    =
         0.00000000D+00  0.14400000D+03
      Number of function calls:         NFUNC =      11
      Number of gradient calls:         NFUNC =      11
      Number of calls of QP solver:     NFUNC =      11

   *** Number of function calls:  22
```

In case of $L = 1$ and analytical derivative computations, the corresponding serial implementation of the main program is as follows:

```
      IMPLICIT          NONE
      INTEGER           NMAX, MMAX, MNN2X, LWA, LKWA, LACTIV
      PARAMETER (       NMAX = 4,
     /                  MMAX = 2,
     /                  MNN2X = MMAX + NMAX + NMAX + 2,
     /                  LWA = 1.5*NMAX*NMAX + 33*NMAX + 9*MMAX + 200,
     /                  LKWA = NMAX + 10,
     /                  LACTIV = 2+MMAX + 10)
      INTEGER           KWA(LKWA), N, ME, M, L, MNN2, MAXIT, MAXFUN,
     /                  IPRINT, MAXNM, IOUT, MODE, IFAIL, I, J, NFUNC
      DOUBLE PRECISION  X(NMAX), F, G(MMAX), DF(NMAX), DG(MMAX,NMAX),
     /                  U(MNN2X), XL(NMAX), XU(NMAX), C(NMAX,NMAX),
     /                  D(NMAX), WA(LWA), ACC, ACCQP, STPMIN, RHOB
      LOGICAL           ACTIVE(LACTIV), LQL
      EXTERNAL          QL
C
C  Set some constants and initial values
C
      IOUT    = 6
      ACC     = 1.0D-10
      ACCQP   = 1.0D-12
      STPMIN  = 0.0
      EPS     = 1.0D-7
      MAXIT   = 100
      MAXFUN  = 10
```

```
        MAXNM    = 0
        RHOB     = O.ODO
        LQL      = .TRUE.
        IPRINT   = 2
        N        = 3
        M        = 2
        ME       = 0
        MNN2     = M + N + N + 2
        MODE     = 0
        IFAIL    = 0
        NFUNC    = 0
        DO I=1,N
           X(I)  = 10.0D0
           XL(I) = 0.0D0
           XU(I) = 42.0D0
        ENDDO
      1 CONTINUE
C=======================================================================
C  This block computes all function values.
C
        F    = -X(1)*X(2)*X(3)
        G(1) =  X(1) + 2.0D0*X(2) + 2.0D0*X(3)
        G(2) = 72.0D0 - X(1) - 2.0D0*X(2) - 2.0D0*X(3)
C
C=======================================================================
        NFUNC = NFUNC + 1
        IF (IFAIL.EQ.-1) GOTO 4
      2 CONTINUE
C=======================================================================
C  This block computes all derivative values.
C
        DF(1)   = -X(2)*X(3)
        DF(2)   = -X(1)*X(3)
        DF(3)   = -X(1)*X(2)
        DG(1,1) =  1.0D0
        DG(1,2) =  2.0D0
        DG(1,3) =  2.0D0
        DG(2,1) = -1.0D0
        DG(2,2) = -2.0D0
        DG(2,3) = -2.0D0
C
C=======================================================================
      4 CONTINUE
        CALL NLPQLP (      L,        M,      ME,     MMAX,        N,
       /                NMAX,     MNN2,       X,        F,        G,
       /                  DF,       DG,       U,       XL,       XU,
       /                   C,        D,     ACC,    ACCQP,   STPMIN,
       /              MAXFUN,    MAXIT,   MAXNM,     RHOB,   IPRINT,
       /                MODE,     IOUT,   IFAIL,       WA,      LWA,
       /                 KWA,     LKWA,  ACTIVE,   LACTIV,      LQL,
       /                  QL)
        IF (IFAIL.EQ.-1) GOTO 1
        IF (IFAIL.EQ.-2) GOTO 2
C
        WRITE(IOUT,1000) NFUNC
   1000 FORMAT(' *** Number of function calls: ',13)
C
        STOP
        END
```

NLPQLP displays the following output:

```
 ---------------------------------------------------------------
 START OF THE SWQUENTIAL QUADRATIC PROGRAMMING ALGORITHM
 ---------------------------------------------------------------


 Parameters:
    N       =        3
    M       =        2
    ME      =        0
    MODE    =        0
    ACC     =   0.1000D-09
    ACCQP   =   0.1000D-11
    STPMIN  =   0.1000D-09
    MAXFUN  =       10
    MAXNM   =        0
    MAXIT   =      100
    IPRINT  =        2

 Output in the following order:
    IT    - iteration number
    F     - objectivefunction value
    SCV   - sum of constraint violations
    NA    - number ofactive constraints
    I     - number of line search iterations
    ALPHA - steplength parameter
    DELTA - additional variable to prevent inconsistency
    KKT   - Karush-Kuhn-Tucker optimality criterion

  IT         F           SCV     NA  I    ALPHA     DELTA      KKT
 ------------------------------------------------------------------------
    1 -0.10000000D+04  0.00D+00    2  0  0.00D+00  0.00D+00  0.44D+04
    2 -0.23625000D+04  0.00D+00    1  1  0.10D+01  0.00D+00  0.11D+04
    3 -0.32507304D+04  0.36D-14    1  1  0.10D+01  0.00D+00  0.69D+03
    4 -0.33041403D+04  0.00D+00    1  1  0.10D+01  0.00D+00  0.36D+03
    5 -0.34527380D+04  0.00D+00    1  1  0.10D+01  0.00D+00  0.58D+01
    6 -0.34559629D+04  0.00D+00    1  1  0.10D+01  0.00D+00  0.76D-01
    7 -0.34560000D+04  0.00D+00    1  1  0.10D+01  0.00D+00  0.25D-04
    8 -0.34560000D+04  0.36D-14    1  1  0.10D+01  0.00D+00  0.90D-10

--- Final Convergence Analysis at Last Iterate ---

 Objective function value:         F(X) =  -0.34560000D+04
 Solution values:                  X    =
    0.24000003D+02  0.11999999D+02    0.11999999D+02
 Distances from lower bounds:      X-XL =
    0.24000003D+02  0.11999999D+02    0.11999999D+02
 Distances from upper bounds:      XU-X =
    0.17999997D+02  0.30000001D+02    0.30000001D+02
 Multipliers for lower bounds:     U    =
    0.00000000D+00  0.00000000D+00    0.00000000D+00
 Multipliers for upper bounds:     U    =
    0.00000000D+00  0.00000000D+00    0.00000000D+00
 Constraint values:                G(X) =
    0.72000000D+02 -0.35527137D-14
 Multipliers for constraints:      U    =
    0.00000000D+00  0.14400000D+03
 Number of function calls:         NFUNC =       8
 Number of gradient calls:         NFUNC =       8
 Number of calls of QP solver:     NFUNC =       8

  *** Number of function calls:   8
```

6 Conclusions

We present a modification of an SQP algorithm designed for execution under a parallel computing environment (SPMD) and where a non-monotone line search is applied in error situations. Numerical results indicate stability and robustness for a set of 306 standard test problems. It is shown that not more than 7 parallel function evaluations per iterations are required for performing a sufficiently accurate line search. Significant performance improvement is achieved by the non-monotone line search especially in case of noisy function values and numerical differentiation, and by restarts in a severe error situation. With the new version of NLPQLP, we are able to solve about 90% of a standard set of 306 test examples subject to an termination accuracy of 10^{-7} in case of extremely noisy function values with relative accuracy of 1% and numerical differentiation. In the worst case, at most one digit of a partial derivative value is correct.

References

[1] Armijo L. (1966): Minimization of functions having Lipschitz continuous first partial derivatives, Pacific Journal of Mathematics, Vol. 16, 1-3.

[2] Barzilai J., Borwein J.M. (1988): Two-point stepsize gradient methods, IMA Journal of Numerical Analysis, Vol. 8, 141-148.

[3] Boderke P., Schittkowski K., Wolf M., Merkle H.P. (2000): Modeling of diffusion and concurrent metabolism in cutaneous tissue, Journal on Theoretical Biology, Vol. 204, No. 3, 393-407.

[4] Bonnans J.F., Panier E., Tits A., Zhou J.L. (1992): *Avoiding the Maratos effect by means of a nonmonotone line search, II: Inequality constrained problems – feasible iterates,* SIAM Journal on Numerical Analysis, Vol. 29, 1187-1202.

[5] Birk J., Liepelt M., Schittkowski K., Vogel F. (1999): *Computation of optimal feed rates and operation intervals for tubular reactors,* Journal of Process Control, Vol. 9, 325-336.

[6] Blatt M., Schittkowski K. (1998): *Optimal Control of One-Dimensional Partial Differential Equations Applied to Transdermal Diffusion of Substrates,* in: Optimization Techniques and Applications, L. Caccetta, K.L. Teo, P.F. Siew, Y.H. Leung, L.S. Jennings, V. Rehbock ed., School of Mathematics and Statistics, Curtin University of Technology, Perth, Australia, Vol. 1, 81-93.

[7] Bongartz I., Conn A.R., Gould N., Toint Ph. (1995): *CUTE: Constrained and unconstrained testing environment,* Transactions on Mathematical Software, Vol. 21, No. 1, 123-160.

[8] Bünner M.J., Schittkowski K., van de Braak G. (2004): *Optimal design of electronic components by mixed-integer nonlinear programming,* Optimization and Engineering, Vol. 5, 271-294.

[9] Dai Y.H., Liao L.Z. (1999): *R-Linear Convergence of the Barzilai and Borwein Gradient Method,* Research Reoport 99-039, Institute of Computational Mathematics and Scientific/Engineering Computing, Chinese Academy of Sciences.

[10] Dai Y.H. (2000): *A nonmonotone conjugate gradient algorithm for unconstrained optimization,* Research Report, Institute of Computational Mathematics and Scientific/Engineering Computing, Chinese Academy of Sciences.

[11] Dai Y.H. (2002): *On the nonmonotone line search*, Journal of Optimization Theory and Applications, Vol. 112, No. 2, 315-330.

[12] Dai Y.H., Schittkowski K. (2008): *A sequential quadratic programming algorithm with non-monotone line search,* Pacific Journal of Optimization, Vol. 4, 335-351.

[13] Deng N.Y., Xiao Y., Zhou F.J. (1993): *Nonmonotonic trust-region algorithm,* Journal of Optimization Theory and Applications, Vol. 26, 259-285.

[14] Edgar T.F., Himmelblau D.M. (1988): *Optimization of Chemical Processes,* Mc-Graw Hill.

[15] Frias J.M., Oliveira J.C, Schittkowski K. (2001): *Modelling of maltodextrin DE12 drying process in a convection oven,* Applied Mathematical Modelling, Vol. 24, 449-462.

[16] Geist A., Beguelin A., Dongarra J.J., Jiang W., Manchek R., Sunderam V. (1995): *PVM 3.0. A User's Guide and Tutorial for Networked Parallel Computing,* The MIT Press.

[17] Goldfarb D., Idnani A. (1983): *A numerically stable method for solving strictly convex quadratic programs,* Mathematical Programming, Vol. 27, 1-33.

[18] Grippo L., Lampariello F., Lucidi S. (1986): *A nonmonotone line search technique for Newtons's method,* SIAM Journal on Numerical Analysis, Vol. 23, 707-716.

[19] Grippo L., Lampariello F., Lucidi S. (1989): *A truncated Newton method with nonmonotone line search for unconstrained optimization,* Journal of Optimization Theory and Applications, Vol. 60, 401-419.

[20] Grippo L., Lampariello F., Lucidi S. (1991): *A class of nonmonotone stabilization methods in unconstrained optimization,* Numerische Mathematik, Vol. 59, 779-805.

[21] Han S.-P. (1976): *Superlinearly convergent variable metric algorithms for general nonlinear programming problems,* Mathematical Programming, Vol. 11, 263-282.

[22] Han S.-P. (1977): *A globally convergent method for nonlinear programming*, Journal of Optimization Theory and Applications, Vol. 22, 297 309.

[23] Hartwanger C., Schittkowski K., Wolf H. (2000): *Computer aided optimal design of horn radiators for satellite communication,* Engineering Optimization, Vol. 33, 221-244.

[24] Hock W., Schittkowski K. (1981): *Test Examples for Nonlinear Programming Codes,* Lecture Notes in Economics and Mathematical Systems, Vol. 187, Springer.

[25] Hock W., Schittkowski K. (1983): *A comparative performance evaluation of 21 nonlinear programming codes,* Computing, Vol. 30, 335-358.

[26] Ke X., Han J. (1995): *A nonmonotone trust region algorithm for equality constrained optimization,* Science in China, Vol. 38A, 683-695.

[27] Ke X., Liu G., Xu D. (1996): *A nonmonotone trust-region algorithm for unconstrained optimization,* Chinese Science Bulletin, Vol. 41, 197-201.

[28] Kneppe G., Krammer J., Winkler E. (1987): *Structural optimization of large scale problems using MBB-LAGRANGE,* Report MBB-S-PUB-305, Messerschmitt-Bölkow-Blohm, Munich.

[29] Liu D.C., Nocedal J. (1989): *On the limited memory BFGS method for large scale optimization,* Mathematical Programming, Vol. 45, 503-528.

[30] Lucidi S., Rochetich F, Roma M. (1998): *Curvilinear stabilization techniques for truncated Newton methods in large-scale unconstrained optimization,* SIAM Journal on Optimization, Vol. 8, 916-939.

[31] Maurer H., Mittelmann H.D. (2000): *Optimization techniques for solving elliptic control problems with control and state constraints: Part 1. Boundary control,* Computational Optimization and Applications, Vol. 16, 29-55.

[32] Maurer H., Mittelmann H.D. (2001): *Optimization techniques for solving elliptic control problems with control and state constraints. Part 2: Distributed control,* Computational Optimization and Applications, Vol. 18, 141-160.

[33] Ortega J.M., Rheinbold W.C. (1970): *Iterative Solution of Nonlinear Equations in Several Variables, Academic Press,* New York-San Francisco-London.

[34] Panier E., Tits A. (1991): *Avoiding the Maratos effect by means of a nonmonotone line search, I: General constrained problems,* SIAM Journal on Numerical Analysis, Vol. 28, 1183-1195.

[35] Papalambros P.Y., Wilde D.J. (1988): *Principles of Optimal Design,* Cambridge University Press.

[36] Powell M.J.D. (1978): *A fast algorithm for nonlinearly constraint optimization calculations*, in: Numerical Analysis, G.A. Watson ed., Lecture Notes in Mathematics, Vol. 630, Springer.

[37] Powell M.J.D. (1978): *The convergence of variable metric methods for nonlinearly constrained optimization calculations,* in: Nonlinear Programming 3, O.L. Mangasarian, R.R. Meyer, S.M. Robinson eds., Academic Press.

[38] Powell M.J.D. (1983): *On the quadratic programming algorithm of Goldfarb and Idnani.* Report DAMTP 1983/Na 19, University of Cambridge, Cambridge.

[39] Raydan M. (1997): *The Barzilai and Borwein gradient method for the large-scale unconstrained minimization problem,* SIAM Journal on Optimization, Vol. 7, 26-33.

[40] Schittkowski K. (1980): *Nonlinear Programming Codes,* Lecture Notes in Economics and Mathematical Systems, Vol. 183 Springer.

[41] Schittkowski K. (1981): *The nonlinear programming method of Wilson, Han and Powell. Part 1: Convergence analysis,* Numerische Mathematik, Vol. 38, 83-114.

[42] Schittkowski K. (1981): *The nonlinear programming method of Wilson, Han and Powell. Part 2: An efficient implementation with linear least squares subproblems,* Numerische Mathematik, Vol. 38, 115-127.

[43] Schittkowski K. (1982): *Nonlinear programming methods with linear least squares subproblems,* in: Evaluating Mathematical Programming Techniques, J.M. Mulvey ed., Lecture Notes in Economics and Mathematical Systems, Vol. 199, Springer.

[44] Schittkowski K. (1983): *Theory, implementation and test of a nonlinear programming algorithm,* in: Optimization Methods in Structural Design, H. Eschenauer, N. Olhoff eds., Wissenschaftsverlag.

[45] Schittkowski K. (1983): *On the convergence of a sequential quadratic programming method with an augmented Lagrangian search direction,* Mathematische Operationsforschung und Statistik, Series Optimization, Vol. 14, 197-216.

[46] Schittkowski K. (1985): *On the global convergence of nonlinear programming algorithms,* ASME Journal of Mechanics, Transmissions, and Automation in Design, Vol. 107, 454-458.

[47] Schittkowski K. (1985/86): NLPQL: *A Fortran subroutine solving constrained nonlinear programming problems,* Annals of Operations Research, Vol. 5, 485-500.

[48] Schittkowski K. (1987): *More Test Examples for Nonlinear Programming,* Lecture Notes in Economics and Mathematical Systems, Vol. 182, Springer.

[49] Schittkowski K. (1988): *Solving nonlinear least squares problems by a general purpose SQP-method,* in: Trends in Mathematical Optimization, K. H. Hoffmann, J.- B. Hiriart-Urruty, C. Lemarechal, J. Zowe eds., International Series of Numerical Mathematics, Vol. 84, Birkhäuser, 295-309.

[50] Schittkowski K. (1994): *Parameter estimation in systems of nonlinear equations,* Numerische Mathematik, Vol. 68, 129-142.

[51] Schittkowski K. (2002): *Numerical Data Fitting in Dynamical Systems,* Kluwer Academic Publishers, Dordrecht.

[52] Schittkowski K. (2002): *EASY-FIT: A software system for data fitting in dynamic systems,* Structural and Multidisciplinary Optimization, Vol. 23, No. 2, 153-169.

[53] Schittkowski K. (2003): QL: *A Fortran code for convex quadratic programming - user's guide,* Report, Department of Mathematics, University of Bayreuth, 2003.

[54] Schittkowski K. (2008): *An active set strategy for solving optimization problems with up to 200,000,000 nonlinear constraints,* Applied Numerical Mathematics, Vol. 59, 2999-3007.

[55] Schittkowski K. (2008): *An updated set of 306 test problems for nonlinear programming with validated optimal solutions - user's guide,* Report, Department of Computer Science, University of Bayreuth.

[56] Schittkowski K., Zillober C., Zotemantel R. (1994): *Numerical comparison of nonlinear programming algorithms for structural optimization,* Structural Optimization, Vol. 7, No. 1, 1-28.

[57] Spellucci P (1993): *Numerische Verfahren der nichtlinearen Optimierung,* Birkhäuser.

[58] Stoer J. (1985): *Foundations of recursive quadratic programming methods for solving nonlinear programs,* in: Computational Mathematical Programming, K. Schittkowski, ed., NATO ASI Series, Series F: Computer and Systems Sciences, Vol. 15, Springer.

[59] Toint P.L. (1996): *An assessment of nonmontone line search techniques for unconstrained optimization*, SIAM Journal on Scientific Computing, Vol. 17, 725-739.

[60] Toint P.L. (1997): *A nonmonotone trust-region algorithm for nonlinear optimization subject to convex constraints,* Mathematical Programming, Vol. 77, 69-94.

[61] Wolfe P. (1969*): Convergence conditions for ascent methods,* SIAM Review, Vol. 11, 226-235.

[62] Zhou J.L., Tits A. (1993): *Nonmonotone line search for minimax problems,* Journal of Optimization Theory and Applications, Vol. 76, 455-476.