SIAM J. OPTIM.
Vol. 20, No. 3, pp. 1655–1684
© 2010 Society for Industrial and Applied Mathematics

THE LIFTED NEWTON METHOD AND ITS APPLICATION IN OPTIMIZATION*

JAN ALBERSMEYER† AND MORITZ DIEHL‡

Abstract. We present a new "lifting" approach for the solution of nonlinear optimization problems (NLPs) that have objective and constraint functions with intermediate variables. Introducing these as additional degrees of freedom into the original problem, combined with adding suitable new constraints to ensure equivalence of the problems, we propose to solve this augmented system instead of the original system by a Newton-type method. This often offers advantages in terms of convergence rates and region of attraction. The main contribution of this article is an efficient algorithmic trick to generate the quantities needed for a Newton-type method on the augmented ("lifted") system with (a) almost no additional computational cost per iteration compared to a nonlifted Newton method, and (b) with negligible programming burden. We derive lifted schemes for Newton's method, as well as for constrained Gauss–Newton and adjoint based sequential quadratic programming (SQP) methods, and show equivalence of the new efficiently lifted approaches with "full-space" lifted Newton iterations. We establish conditions on the intermediate functions that imply faster local quadratic convergence for lifted versus nonlifted Newton iterations, a phenomenon often observed in practice but not yet explained theoretically. Finally, we compare numerically the behavior of the lifted approach with the nonlifted approach on several test problems, including a large scale example with 27 million intermediate variables. The algorithms and examples are available as open-source code in the C++ package LiftOpt.

Key words. nonlinear optimization, constrained optimization, shooting methods, Newton-type methods, SQP methods

AMS subject classifications. 90C55, 90C90, 90C06, 90C30, 65K05

DOI. 10.1137/080724885

1. Introduction.

It has occasionally been observed, particularly in the context of the solution of boundary value problems by shooting methods, that transferring a nonlinear root finding problem into a higher-dimensional space might offer advantages in terms of convergence rates and region of attraction [1, 11, 15]. This classical "multiple shooting" method works by introducing intermediate variables as additional degrees of freedom and corresponding constraints to ensure equivalence to the original problem. It then solves this augmented equivalent system, which we might call the "lifted" system, instead of the original system by a Newton-type method. At first sight, the increased size of the lifted system seems to render each Newton-type

*Received by the editors May 21, 2008; accepted for publication (in revised form) October 29, 2009; published electronically January 22, 2010. This research was supported by the DFG via IGK 710 (Complex processes: Modeling, Simulation and Optimization) and by the Research Council KUL (Center of Excellence on Optimization in Engineering (OPTEC) EF/05/006, GOA AMBioRICS, IOF-SCORES4CHEM, and Ph.D./postdoc/fellow grants), the Flemish Government via FWO (Ph.D./postdoc grants, projects G.0452.04, G.0499.04, G.0211.05, G.0226.06, G.0321.06, G.0302.07, G.0320.08, G.0558.08, research communities ICCoS, ANMMM, MLDM), and via IWT (Ph.D. grants, McKnow-E, Eureka-Flite), Helmholtz vICeRP, the EU via ERNSI, as well as the Belgian Federal Science Policy Office: IUAP P6/04 (DYSCO, Dynamical systems, control and optimization, 2007-2011).

http://www.siam.org/journals/siopt/20-3/72488.html

†Simulation and Optimization Team, Interdisciplinary Center for Scientific Computing (IWR), Ruprecht-Karls University Heidelberg, Im Neuenheimer Feld 368, D-69120 Heidelberg, Germany (jan.albersmeyer@iwr.uni-heidelberg.de).

‡Electrical Engineering Department (ESAT) and Optimization in Engineering Center (OPTEC), K.U. Leuven, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium (moritz.diehl@esat.kuleuven.be).

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

iteration more expensive. This can be overcome, however, by structure-exploiting linear algebra in each Newton-type step (see [14, 10, 13]). Besides the classical domain of multiple shooting, parameter estimation, and optimal control in ODEs and DAEs, where the natural choice of intermediate values are the system states at different timepoints, there exist other problem classes that can benefit from "lifting." A direct transfer of the idea can be made to the case of discrete time models or optimization in the context of kinematic chains arising in robotics. However, "multiple shooting" and related "lifted" approaches are often not used due to the increased programming burden. Usually it would be necessary to split up the original algorithm according to the choice of intermediate values and the structure of the problem into a sequence of subfunctions. From these one has to compute and assemble the quantities and derivatives of the augmented system. Afterward the augmented problem has to be "condensed" again to obtain small reduced subproblems needed for the step computation in efficient algorithms. All of these steps are technically nontrivial to implement.

In this paper, we propose algorithms for the solution of nonlinear optimization problems that solve the augmented system by a structure-exploiting Newton-type method yet do not require any additional user knowledge about the structure of the problem functions or the meaning of the intermediate variables. We show that the cost of each iteration of these "lifted" methods is nearly identical to the cost of one iteration for the solution of the original problems. Furthermore, we make a first step toward proving the superior local convergence speed of lifted Newton methods.

The paper is organized as follows: In section 2, we explain the idea of lifting in the example of Newton's method for a root finding problem and derive a "lifted" Newton algorithm. In section 3, we discuss the application of the lifting approach to optimization, deriving a lifted Gauss–Newton method and a lifted SQP method for equality and inequality constrained optimization problems that is based on adjoint gradient computations. Equivalence of the last method with the iterations obtained by a full-space SQP method is proven. In section 4 we discuss under which circumstances "lifted" approaches converge faster than nonlifted ones and give a proof in a simplified setting. In section 5 we present an open-source implementation of the derived lifted algorithms in the package LiftOpt and demonstrate the performance on several numerical examples, including a wave equation parameter estimation problem with 27 million intermediate variables. We conclude the paper in section 6.

2. The lifted Newton method. We first consider the problem of solving a nonlinear system of equations, represented by

$$F(u) = 0, \tag{2.1}$$

where the evaluation of the function $F \in \mathcal{C}^1(\mathbb{R}^{n_u}, \mathbb{R}^{n_u})$ is given in the form of a possibly complex algorithm with several intermediate variables. Denoting these intermediate variables by $x_i \in \mathbb{R}^{n_i}$ for $i = 1, 2, \ldots, m$ and disregarding further internal structure, we summarize the algorithm in the generic form

$$x_i := f_i(u, x_1, x_2, \ldots, x_{i-1}) \quad \text{for} \quad i = 1, 2, \ldots, m, \tag{2.2}$$

where the final output $F(u)$ is given by

$$f_F(u, x_1, x_2, \ldots, x_m). \tag{2.3}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

It is straightforward to see that the original system (2.1) is equivalent to the "lifted" system of nonlinear equations

$$G(u,x)=0, \tag{2.4}$$

with $n_x=\sum_{i=1}^m n_i$, $x=(x_1,\dots,x_m)$, and where $G\in\mathcal{C}^1(\mathbb{R}^{n_u}\times\mathbb{R}^{n_x},\mathbb{R}^{n_u}\times\mathbb{R}^{n_x})$ is given by

$$G(u,x)=\begin{pmatrix} f_1(u) & - & x_1\\ f_2(u,x_1) & - & x_2\\ \vdots & & \\ f_m(u,x_1,\dots,x_{m-1}) & - & x_m\\ f_F(u,x_1,\dots,x_m) & & \end{pmatrix}. \tag{2.5}$$

To solve the augmented system (2.4), the lifted Newton method iterates, starting at a guess (x^0,u^0),

$$\begin{pmatrix} x^{k+1}\\ u^{k+1}\end{pmatrix}=\begin{pmatrix} x^{k}\\ u^{k}\end{pmatrix}+\begin{pmatrix} \Delta x^{k}\\ \Delta u^{k}\end{pmatrix} \tag{2.6a}$$

with

$$\begin{pmatrix} \Delta x^{k}\\ \Delta u^{k}\end{pmatrix}=-\left[\frac{\partial G}{\partial(u,x)}(x^k,u^k)\right]^{-1}G(x^k,u^k). \tag{2.6b}$$

To initialize the value x^0, we can simply call the algorithm defining the original function $F(u^0)$ and add a few lines to output all intermediate variables; i.e., we call Algorithm 1. Please note that we are also free to choose the intermediate values otherwise and that it is often advantageous to do so.

Algorithm 1: Function with output of intermediate variables.

Input : $u\in\mathbb{R}^{n_u}$
Output: $x_1\in\mathbb{R}^{n_1},\dots,x_m\in\mathbb{R}^{n_m},F\in\mathbb{R}^{n_u}$
begin
  for $i=1,2,\dots,m$ **do**
    $x_i=f_i(u,x_1,x_2,\dots,x_{i-1})$;
  end for
  $F:=f_F(u,x_1,x_2,\dots,x_m)$;
end

By modifying the user given Algorithm 1 slightly, we can easily define the residual function $G(u,y)$ as follows by Algorithm 2.

Algorithm 2: Residual function $G(u,y)$.

Input : $u,y_1,\dots,y_m$
Output: $G_1,\dots,G_m,F$
begin
  for $i=1,2,\dots,m$ **do**
    $x_i=f_i(u,x_1,x_2,\dots,x_{i-1})$;
    $G_i=x_i-y_i$;
    $x_i=y_i$;
  end for
  $F=f_F(u,x_1,x_2,\dots,x_m)$;
end

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Thus, it is easy to transform a given user function into a function that outputs the residuals. Note that it is not necessary that all x_i are distinct variables with separately allocated memory within the program code. The only code modification is to add after each computation of an intermediate variable two lines (or even only one line calling a more convenient function defined in Algorithm 7) that store the residual and set the variable to the given input value y_i.

For notational convenience, we define

$$H(u,x) := \begin{pmatrix} f_1(u) \\ f_2(u,x_1) \\ \vdots \\ f_m(u,x_1,\ldots,x_{m-1}) \end{pmatrix}, \tag{2.7}$$

such that

$$G(u,x) = \begin{pmatrix} H(u,x) - x \\ f_F(u,x) \end{pmatrix}. \tag{2.8}$$

It is straightforward to see that Algorithm 1 delivers, for given u, the unique solution $\bar{x}$ of $H(u,\bar{x}) - \bar{x} = 0$.

To perform a Newton method on the augmented system, we have to calculate the increments in (2.6b). Dropping the index k for notational convenience, we have to solve at every iteration the linear system

$$H(u,x) - x + \left(\frac{\partial H}{\partial x}(u,x) - \mathbb{I}_{n_x}\right)\Delta x + \frac{\partial H}{\partial u}(u,x)\Delta u = 0, \tag{2.9}$$

$$f_F(u,x) + \frac{\partial f_F}{\partial x}(u,x)\Delta x + \frac{\partial f_F}{\partial u}(u,x)\Delta u = 0, \tag{2.10}$$

where $\mathbb{I}_{n_x}$ represents the identity operator in $\mathbb{R}^{n_x}$. Due to the fact that the square matrix $\left(\frac{\partial H}{\partial x}(u,x) - \mathbb{I}_{n_x}\right)$ is lower triangular with nonzero diagonal, and thus invertible, (2.9) is equivalent to

$$\begin{aligned} \Delta x &= \underbrace{-\left(\frac{\partial H}{\partial x}(u,x) - \mathbb{I}_{n_x}\right)^{-1}(H(u,x) - x)}_{=:a} \\ &\quad \underbrace{-\left(\frac{\partial H}{\partial x}(u,x) - \mathbb{I}_{n_x}\right)^{-1}\frac{\partial H}{\partial u}(u,x)}_{=:A}\Delta u \\ &= a + A\Delta u. \end{aligned} \tag{2.11}$$

Based on this, we can "condense" the second equation to

$$\begin{array}{rcccccc} 0 & = & f_F(u,x) + \frac{\partial f_F}{\partial x}(u,x)a & + & \left(\frac{\partial f_F}{\partial u}(u,x) + \frac{\partial f_F}{\partial x}(u,x)A\right) & \Delta u \\ & =: & b & + & B & \Delta u. \end{array} \tag{2.12}$$

If the "reduced quantities" a, A, b, and B were known, we could easily compute the step by

$$\Delta u = -B^{-1}\, b, \tag{2.13}$$

$$\Delta x = a + A\, \Delta u. \tag{2.14}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

2.1. An algorithmic trick for efficient derivative computation. In the following we will present an algorithmic trick to compute the vectors a, b and the matrices A, B efficiently, without the need to form or invert the derivatives of H explicitly. This trick is a generalization of "Schlöder's trick" [14], which is well known in the context of multiple shooting for parameter estimation and was extended to optimal control by Schäfer [13]. In all these existing approaches, however, the algorithms are especially tailored to specific sparsity structures. On the other hand, the new generalized trick does not require any structure or user input, apart from a minimal number of extra lines of code into the function to be "lifted," as illustrated in Algorithm 6.

To derive the trick, we introduce a function $Z(u, d)$ as follows: For given vectors u and d, the unique solution z of the system $H(u, z) - z - d = 0$, which we will denote in the following by $Z(u, d)$, can be computed easily by Algorithm 3. The algorithm simultaneously computes the value $f_F(u, Z(u, d))$.

Algorithm 3: Modified function $Z(u, d)$.

```
Input  : u, d_1, ..., d_m
Output: z_1, ..., z_m, F
begin
    for i = 1, 2, ..., m do
        x_i = f_i(u, x_1, x_2, ..., x_{i-1});
        z_i = x_i - d_i;
        x_i = z_i;
    end for
    F = f_F(u, x_1, x_2, ..., x_m);
end
```

The derivatives of the function $Z(u, d)$ with respect to u and d help us in computing the vector a and the matrix A as well as b and B. To see this, we first observe that

$$Z(u, H(u, x) - x) = x.$$

Thus, by setting $d = H(u, x) - x$, we can call Algorithm 3 to obtain $x = Z(u, d)$ and $f_F(u, x)$.

On the other hand, from the defining equation of $Z(u, d)$, namely,

$$H(u, Z(u, d)) - Z(u, d) - d = 0,$$

we obtain the following two equations by total differentiation with respect to u and d:

$$\frac{\partial H}{\partial u}(u, x) + \frac{\partial H}{\partial x}(u, x)\frac{\partial Z}{\partial u}(u, d) - \frac{\partial Z}{\partial u}(u, d) = 0$$

and

$$\frac{\partial H}{\partial x}(u, x)\frac{\partial Z}{\partial d}(u, d) - \frac{\partial Z}{\partial d}(u, d) - \mathbb{I}_{n_x} = 0.$$

Note that we have assumed $d = H(u, x) - x$ so that $Z(u, d) = x$. The two relations above are equivalent to

$$\frac{\partial Z}{\partial u}(u, d) = -\left(\frac{\partial H}{\partial x}(u, x) - \mathbb{I}_{n_x}\right)^{-1} \frac{\partial H}{\partial u}(u, x)$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

and

$$\frac{\partial Z}{\partial d}(u,d) = \left(\frac{\partial H}{\partial x}(u,x) - \mathbb{I}_{n_x}\right)^{-1}.$$

Therefore, the derivatives a and A can efficiently be computed as directional derivatives of Z,

$$a = -\frac{\partial Z}{\partial d}(u,d)\ d \quad \text{and} \quad A = \frac{\partial Z}{\partial u}(u,d),$$

by differentiation of Algorithm 3. As a by-product, the vector b and the matrix B are obtained as the derivatives of the last output F of Algorithm 3.

Summarizing, we propose Algorithm 4 to perform the computations within the lifted Newton method.

2.2. Practical implementation. The described Algorithms 1–3 used in the lifted Newton method can be obtained with minimal modification of the original function evaluation by adding calls to a "node" function. For this, we assume that the original function evaluation is given in the abstract form of Algorithm 5, where the $w_i, i = 1, \ldots, m$, now denote the intermediate values that should be used for lifting. It is then sufficient, as illustrated in Algorithm 6, to add after each computation of an intermediate value w_i a call to the node function defined in Algorithm 7. This modified function then evaluates the different Algorithms 1–3, depending on the value of the global flag f_{mode}, which has to be set appropriately by the calling function. The global variables x, z, and d serve as input/output values, depending on the chosen algorithm.

3. Application to optimization. The idea of lifting can also be used in the context of optimization. Let us see how we can use the new trick to efficiently generate the quantities needed for "lifted" Gauss–Newton and SQP methods. The new methods shall need to solve only subproblems in the original degrees of freedom to determine a step, while the iterations will be operating in the whole variable space. We also need to show that the iterations made by the proposed lifting approach and by full-space methods are identical.

3.1. A lifted Gauss–Newton method. Using the idea of lifting from above, we can develop a lifted Gauss–Newton method to solve least-squares–type problems with (in)equality constraints. We consider the following type of optimization problem:

$$\min_u \ \frac{1}{2}\|F_1(u)\|_2^2 \tag{3.1a}$$

$$\text{s.t.}$$

$$F_2(u) \left\{\begin{matrix}=\\ \geq\end{matrix}\right\} 0, \tag{3.1b}$$

where $F_1 : \mathbb{R}^{n_u} \to \mathbb{R}^{n_{\text{res}}}$ is a vector valued function, and the function $F_2 : \mathbb{R}^{n_u} \to \mathbb{R}^{n_c}$ represents nonlinear equality and inequality constraints. The Gauss–Newton approach then uses the Jacobian $J_1(u) = \frac{\partial F_1}{\partial u}(u)$ to compute an approximation $J_1(u)^T J_1(u)$ of the Hessian of the Lagrangian of the system. The increments Δu_k are in the constrained Gauss–Newton method computed via the solution of the subproblem

$$\min_{\Delta u} \ \frac{1}{2}\|F_1(u_k) + J_1(u_k)\Delta u\|_2^2 \tag{3.2a}$$

$$\text{s.t.}$$

$$F_2(u_k) + J_2(u_k)\Delta u \left\{\begin{matrix}=\\ \geq\end{matrix}\right\} 0. \tag{3.2b}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Algorithm 4: The lifted Newton method.

Input : u^0, TOL, $o_{\text{manNodeInit}}$, x^0_{user}
Output: u^*, x^*, d^*, F^*
begin
  Set $k = 0$;
  if $o_{\text{manNodeInit}} ==$ false **then**
    `// Initialize node values` x^0 `by function evaluation`
    Set $d^0 = 0$;
    Call Algorithm 3 with inputs u^0, d^0 and set $x^0 = Z(u^0, d^0)$;
    $F^0 = f_F(u^0, Z(u^0, d^0))$;
  else
    `// Initialize node values` x^0 `manually`
    Set $x^0 = x^0_{\text{user}}$;
    Call Algorithm 2 with inputs u^0, x^0 to obtain d^0, F^0;
  end
  while $\|F^k\| + \|d^k\| \geq$ TOL **do**
    Differentiate Algorithm 3 directionally at (u^k, d^k) to obtain
    $a^k = -\frac{\partial Z}{\partial d}(u^k, d^k)d$;
    $A^k = \frac{\partial Z}{\partial u}(u^k, d^k)$;
    $b^k = F^k - \frac{\mathrm{d}f_F(u,Z(u,d))}{\mathrm{d}d}d$;
    $B^k = \frac{\mathrm{d}f_F(u,Z(u,d))}{\mathrm{d}u}$;
    Solve the condensed Newton system to obtain
    $\Delta u^k = -(B^k)^{-1}b^k$;
    Perform the Newton step
    $x^{k+1} = x^k + a^k + A^k \Delta u^k$;
    $u^{k+1} = u^k + \Delta u^k$;
    Call Algorithm 2 with inputs (u^{k+1}, x^{k+1}) to obtain
    $d^{k+1} = H(u^{k+1}, x^{k+1}) - x^{k+1}$;
    $F^{k+1} = f_F(u^{k+1}, x^{k+1})$;
    Set $k = k + 1$;
  end
  Set
  $u^* = u^k$;
  $x^* = x^k$;
  $d^* = d^k$;
  $F^* = F^k$;
end

The Gauss–Newton approach can also be lifted, which offers impressive advantages in convergence speed, as has been demonstrated by Bock [2], Schlöder [14], and Kallrath, Bock, and Schlöder [9]. The augmented problem after the introduction of intermediate values $x_i, i = 1, \ldots, m$, reads then in the notation from section 2 as

$$\min_{u,x} \frac{1}{2}\|f_{F_1}(u, x)\|_2^2 \tag{3.3a}$$
s.t.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Algorithm 5: Original function evaluation.

Input : $u \in \mathbb{R}^{n_u}$
Output: $F \in \mathbb{R}^{n_u}$
begin
  for $i = 1, 2, \ldots, m$ **do**
    $w_i = f_i(u, w_1, w_2, \ldots, w_{i-1})$;
  end for
  $F := f_F(u, w_1, w_2, \ldots, w_m)$;
end

Algorithm 6: Function evaluation modified for use in lifted algorithms.

Input : $u \in \mathbb{R}^{n_u}$
Output: $F \in \mathbb{R}^{n_u}$
begin
  for $i = 1, 2, \ldots, m$ **do**
    $w_i = f_i(u, w_1, w_2, \ldots, w_{i-1})$;
    Call node(w_i);
  end for
  $F := f_F(u, w_1, w_2, \ldots, w_m)$;
end

$$f_{F_2}(u, x) \left\{\begin{matrix}=\\ \geq\end{matrix}\right\} 0, \tag{3.3b}$$

$$H(u, x) - x = 0. \tag{3.3c}$$

This results in the following augmented quadratic program (QP) for the step determination:

$$\min_{\Delta u, \Delta x} \quad \frac{1}{2} \left\| f_{F_1}(u_k, x_k) + \frac{\partial f_{F_1}}{\partial (u, x)}(u_k, x_k) \begin{pmatrix} \Delta u \\ \Delta x \end{pmatrix} \right\|_2^2 \tag{3.4a}$$

s.t.

$$f_{F_2}(u_k) + \frac{\partial f_{F_2}}{\partial (u, x)}(u_k, x_k) \begin{pmatrix} \Delta u \\ \Delta x \end{pmatrix} \left\{\begin{matrix}=\\ \geq\end{matrix}\right\} 0, \tag{3.4b}$$

$$H(u_k, x_k) - x_k + \frac{\partial H}{\partial (u, x)}(u_k, x_k) \begin{pmatrix} \Delta u \\ \Delta x \end{pmatrix} - \Delta x = 0. \tag{3.4c}$$

While at first sight this transformation seems to be disadvantageous due to the increased size of the QP and the need to compute the derivatives of the functions also with respect to the intermediate values x, the problem can be set up and solved at roughly the cost of one Gauss–Newton iteration of the original problem, which is formulated only with variables u as degrees of freedom. This was discovered by Schlöder [14] in the context of multiple shooting for parameter estimation and extended to direct multiple shooting for optimal control by Schäfer [13].

In order to compute the iterates efficiently, we propose here to simply lift the evaluation of $F(u) := (F_1(u)^T, F_2(u)^T)^T$ and use Algorithm 4 to directly compute the condensed quantities a, A, $b = (b_1^T, b_2^T)^T$, and $B = (B_1^T, B_2^T)^T$ needed for the

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Algorithm 7: Node (w).

```
Global variables: x_1, ..., x_m, z_1, ..., z_m, d_1, ..., d_m, i, f_mode
Input / Output : w
begin
    switch f_mode do
        case f_mode == "original"
        end
        case f_mode == "init"
            // (see Algorithm 1);
            x_i := w;
            i := i + 1;
        end
        case f_mode == "residual"
            // (see Algorithm 2);
            d_i := w - x_i;
            w := x_i;
            i := i + 1;
        end
        case f_mode == "reduced"
            // (see Algorithm 3);
            z_i := w - d_i;
            w := z_i;
            i := i + 1;
        end
    end
end
```

condensed QP and the following step expansion. The condensed QP is of the form

$$\min_{\Delta u} \ \frac{1}{2}\|b_1 + B_1 \Delta u\|_2^2 \tag{3.5a}$$

s.t.

$$b_2 + B_2 \Delta u \left\{ \begin{matrix} = \\ \geq \end{matrix} \right\} 0. \tag{3.5b}$$

It is then solved using the same least-squares QP solver as in the nonlifted Gauss–Newton method to generate a solution Δu_k. This is then expanded using the relation $\Delta x_k = a + A\Delta u_k$. This procedure obviously delivers the same result as solving (3.3). Numerical results for our new way to implement this well-known approach are given in section 5.3.

3.2. Nonlinear optimization via the lifted Newton method. We can also derive a partially reduced SQP method for nonlinear optimization that is based on the lifting idea. By "partially reduced" we understand in this context that the constraints resulting from the introduction of intermediate variables and the intermediate variables themselves are eliminated from the subproblems, while the constraints of the nonlifted nonlinear problem remain. In the following we first show equivalence of the lifted SQP iterations with a classical full-space SQP approach in the simpler unconstrained case, and afterward we treat the general constrained case.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

3.2.1. Full-space exact-Hessian SQP iterations. Let us consider the solution of an unconstrained optimization problem. Assume that we want to minimize a function $\varphi(u), \varphi : \mathbb{R}^{n_u} \to \mathbb{R}$, and we have lifted the evaluation of φ by introducing additional variables $w_i, i = 1, \ldots, m$. This results in the augmented optimization problem (with the notation from above)

$$\min_{u,w} \quad f_\varphi(u, w_1, w_2, \ldots, w_m) \tag{3.6a}$$

s.t.

$$g(u,w) = \begin{pmatrix} f_1(u) & - & w_1 \\ f_2(u, w_1) & - & w_2 \\ \vdots & & \\ f_m(u, w_1, \ldots, w_{m-1}) & - & w_m \end{pmatrix} = 0, \tag{3.6b}$$

with the corresponding KKT system for the first order necessary optimality conditions

$$\nabla_u \mathcal{L}(u, w, \lambda) = \nabla_u f_\varphi(u, w) + \nabla_u g(u, w)\lambda = 0, \tag{3.7a}$$

$$\nabla_w \mathcal{L}(u, w, \lambda) = \nabla_w f_\varphi(u, w) + \nabla_w g(u, w)\lambda = 0, \tag{3.7b}$$

$$\nabla_\lambda \mathcal{L}(u, w, \lambda) = g(u, w) = 0, \tag{3.7c}$$

using the notation $\nabla_u F(u) = \frac{\partial F}{\partial u}(u)^T$. The variables λ are the Lagrange multipliers for the equality constraints concerning the intermediate values w and $\mathcal{L}(u, w)$ the Lagrange function of the augmented optimization system. The standard full-space exact-Hessian approach then employs a standard Newton method to solve this root finding problem, iterating in the full variable space of (u, w, λ).

3.2.2. How to compute the full-space iterations efficiently. To use the lifting approach efficiently we start by evaluating the gradient $\nabla\varphi$ of the original objective using the principles of the adjoint mode of automatic differentiation; cf. Griewank [7]. This leads to the following evaluation sequence of the function $F : u \mapsto \bar{u} \equiv \nabla_u \varphi$ with intermediate values w and $\bar{w}$:

$$w_1 = f_1(u), \tag{3.8a}$$

$$w_2 = f_2(u, w_1) \tag{3.8b}$$

$$\vdots$$

$$w_m = f_m(u, w_1, \ldots, w_n - 1), \tag{3.8c}$$

$$y \equiv f_\varphi(u, w_1, \ldots, w_m), \tag{3.8d}$$

$$\bar{w}_m = \nabla_{w_m} f_\varphi, \tag{3.8e}$$

$$\bar{w}_{m-1} = \nabla_{w_{m-1}} f_\varphi + \nabla_{w_{m-1}} f_m \bar{w}_m \tag{3.8f}$$

$$\vdots$$

$$\bar{w}_1 = \nabla_{w_1} f_\varphi + \sum_{i=2}^{m} \nabla_{w_1} f_i \bar{w}_i, \tag{3.8g}$$

$$\bar{u} = \nabla_u f_\varphi + \sum_{i=1}^{m} \nabla_u f_i \bar{w}_i. \tag{3.8h}$$

We now lift all intermediate variables $x = (w_1, \ldots, w_m, \bar{w}_m, \ldots, \bar{w}_1)$ in the gradient evaluation procedure $F(u)$, i.e., we interpret (3.8a)–(3.8g) to be $H(u, x) - x = 0$, as

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

before, and (3.8h) as $\bar{u} = f_F(u, x)$. Doing this we can show that the lifted Newton iterations toward the solution of

$$f_F(u, x) = 0, \tag{3.9a}$$

$$H(u, x) - x = 0 \tag{3.9b}$$

and the iterations of a full-space exact-Hessian SQP method to solve system (3.7) in variables u, w, and λ are identical. To see this first observe that (3.8a)–(3.8c) is equivalent to (3.7c) and (3.8e)–(3.8g) to (3.7b) if we set $\lambda \equiv \bar{w}$ and that (3.8h) with $\bar{u} = 0$ is equivalent to (3.7a).

THEOREM 1. *The lifted Newton iterations in variables (u, x) applied to the lifted equivalent of the function $F(u) := \nabla_u \varphi(u)$ are identical to the exact-Hessian full-space SQP iterates in variables (u, w, λ).*

3.2.3. A lifted SQP method. Let us now consider the nonlinear constrained optimization problem

$$\begin{aligned} &\min_u && \varphi(u) \\ &\text{s.t.} \\ &&& h(u) \left\{ \begin{matrix} = \\ \geq \end{matrix} \right\} 0, \end{aligned} \tag{3.10a, 3.10b}$$

where $\varphi : \mathbb{R}^{n_u} \to \mathbb{R}$ is a nonlinear cost function and $h : \mathbb{R}^{n_u} \to \mathbb{R}^{n_c}$ represents again nonlinear equality and inequality constraints. The corresponding Lagrange function is then $\mathcal{L}(u, \mu) = \varphi(u) + \mu^T h(u)$. In the standard exact-Hessian SQP approach one solves this problem iteratively, starting at a point (u_0, μ_0). The increments are computed by solving the quadratic problem

$$\begin{aligned} &\min_{\Delta u} && \frac{1}{2} \Delta u^T M_k \Delta u + \nabla_u \varphi(u_k)^T \Delta u \\ &\text{s.t.} \\ &&& h(u_k) + \nabla_u h(u_k)^T \Delta u \left\{ \begin{matrix} = \\ \geq \end{matrix} \right\} 0, \end{aligned} \tag{3.11a, 3.11b}$$

where $M_k = \nabla^2_{uu} \mathcal{L}(u_k, \mu_k)$. The iteration uses the QP solution Δu_k as a step in the primal variables, and the corresponding QP multipliers as new multiplier guess μ_{k+1}. If we now introduce intermediate variables $w_i, i = 1, \ldots, m$, we obtain the augmented optimization problem of the form

$$\begin{aligned} &\min_{u,w} && f_\varphi(u, w) \\ &\text{s.t.} \\ &&& f_h(u, w) \left\{ \begin{matrix} = \\ \geq \end{matrix} \right\} 0, \\ &&& g(u, w) \;\; = \;\; 0, \end{aligned} \tag{3.12a, 3.12b, 3.12c}$$

with the Lagrangian $\mathcal{L}(u, w, \lambda, \mu) = f_\varphi(u, w) + \lambda^T g(u, w) + \mu^T f_h(u, w)$. The full-space QP subproblem in case of an exact-Hessian SQP method then reads

$$\begin{aligned} &\min_{\Delta u, \Delta w} && \frac{1}{2} \begin{pmatrix} \Delta u \\ \Delta w \end{pmatrix}^T \nabla^2 \mathcal{L}(\cdot) \begin{pmatrix} \Delta u \\ \Delta w \end{pmatrix} + \nabla f_\varphi(u_k, w_k)^T \begin{pmatrix} \Delta u \\ \Delta w \end{pmatrix} \\ &\text{s.t.} \end{aligned} \tag{3.13a}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

$$f_h(u_k, w_k) + \nabla f_h(u_k, w_k)^T \begin{pmatrix} \Delta u \\ \Delta w \end{pmatrix} \left\{ \begin{matrix} = \\ \geq \end{matrix} \right\} 0, \tag{3.13b}$$

$$g(u_k, w_k) + \nabla g(u_k, w_k)^T \begin{pmatrix} \Delta u \\ \Delta w \end{pmatrix} = 0. \tag{3.13c}$$

Again we can use the lifting algorithm to compute the iterates more efficiently than by solving the full-space QP. By a straightforward application of the lifting idea to the combined function evaluation

$$F(u, \mu) := \nabla \mathcal{L}(u, \mu) \equiv \begin{pmatrix} \nabla_u \varphi(u) + \nabla_u h(u) \mu \\ h(u) \end{pmatrix}$$

we obtain the required condensed quantities b_1, b_2, B_1, and B_2 for the condensed QP

$$\min_{\Delta u} \frac{1}{2} \Delta u^T B_1 \Delta u + b_1^T \Delta u \tag{3.14a}$$

s.t.

$$b_2 + B_2 \Delta u \left\{ \begin{matrix} = \\ \geq \end{matrix} \right\} 0, \tag{3.14b}$$

which can afterward be expanded using the multiplier of the QP subproblem solution to a step $\Delta\mu$ and using a and A finally to steps Δw and $\Delta\lambda$. Note that it is sufficient in this case to build the condensed quantities only for the degrees of freedom u to obtain valid b_1, b_2 and B_1, B_2 for the condensed QP. After the QP is solved and $\Delta\mu$ is computed one has to compute one additional directional derivative to expand the step to Δw and $\Delta\lambda$. Again, equivalence of iterations generated by the lifted algorithm and the full-space SQP iterations can be shown, provided that the lifting is done as described in the unconstrained case. The lengthy derivation of this fact and the details of the step expansion procedure for the constrained case are described in the appendix.

4. Local convergence analysis of lifted Newton methods. While the main contribution of this paper lies in the derivation of easy-to-implement algorithms for lifted Newton methods that each have the same computational complexity per iteration as the corresponding nonlifted methods, the idea of "lifting" in itself is an old idea. However, its convergence properties are not well understood. For the solution of boundary value problems with underlying nonlinear ODE models, the multiple shooting method (which can be regarded a lifted algorithm) is long since known to outperform the single shooting method [11]. Three reasons are often cited for the superiority of the "lifted" compared to the nonlifted Newton approaches:

- more freedom for initialization,
- better conditioned and block-sparse linear systems, and
- faster local convergence.

While the first two reasons are well understood, no detailed local convergence analysis exists so far that explains this superior local convergence of "lifted" Newton methods. In order to make a first step in approaching this question, we regard a model root finding problem $F(u) = 0$, where we have a chain of nonlinear functions that each depend only on the output of the immediate predecessor function, and that all have the same input and output dimensions:

$$x_1 = f_1(u),\ x_2 = f_2(x_1), \ldots, x_m = f_m(x_{m-1}),\ \text{and}\ f_F(x_m) \equiv f_{m+1}(x_m).$$

In order to simplify the following discussion further, we will now restrict ourselves to the simplest case, where u and all other variables are scalar. We regard the local

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

convergence rate in the neighborhood of the solution. At this solution (u^*, x^*), all Jacobians must be invertible. Therefore, by suitable affine variable transformations[1] for $x_0 := u$ and for $x_1, \dots, x_m$ we can assume both that the solution is zero for all variables and that all functions f_i are given by

$$f_i(x) = x + b_i(x)^2 + O(|x|^3). \tag{4.1}$$

4.1. Local convergence of the nonlifted Newton method. In the simplified setting outlined above, the nonlifted function $F(u)$ is given by

$$F(u) = f_{m+1}(f_m(\dots f_1(u)\dots)) = u + \left(\sum_{i=1}^{m+1} b_i\right) u^2 + O(|u|^3)$$

and its derivative is given by

$$F'(u) = 1 + 2\left(\sum_{i=1}^{m+1} b_i\right) u + O(|u|^2).$$

To regard the local convergence behavior near zero, we first note that for any Newton method it holds that

$$u^{[k+1]} = u^{[k]} - F'(u^{[k]})^{-1} F(u^{[k]}) = F'(u^{[k]})^{-1}\Big(F'(u^{[k]})u^{[k]} - F(u^{[k]})\Big),$$

and due to the fact that in our case

$$F'(u^{[k]})u^{[k]} = \left(u^{[k]} + 2\left(\sum_{i=1}^{m+1} b_i\right)(u^{[k]})^2 + O(|u^{[k]}|^3)\right),$$

this leads to the iteration formula

$$u^{[k+1]} = \left(\sum_{i=1}^{m+1} b_i\right)(u^{[k]})^2 + O(|u^{[k]}|^3).$$

Thus, the local contraction constant for quadratic convergence is given by $(\sum_{i=1}^{m+1} b_i)$.

4.2. Local convergence of the lifted Newton method. In the simplified setting outlined above, the lifted function $G(u, x)$ is given by

$$G(u,x) = \begin{pmatrix} u + b_1 u^2 & - & x_1 \\ x_1 + b_2 x_1^2 & - & x_2 \\ \vdots & & \\ x_{m-1} + b_m x_{m-1}^2 & - & x_m \\ x_m + b_{m+1} x_m^2 & & \end{pmatrix} + O\left(\left\| \begin{pmatrix} u \\ x \end{pmatrix} \right\|^3\right), \tag{4.2}$$

[1] The affine transformations to new variables and functions are

$$x_i^{\text{new}} = (x_i - x_i^*)/a_i, \qquad a_0 := 1, \quad a_i := \prod_{j=1}^{i} f_j'(x_{j-1}^*), 1 \le i \le m+1, \quad \text{and}$$

$$f_i^{\text{new}}(x_{i-1}^{\text{new}}) = \frac{1}{a_i}\left(f_i(x_{i-1}) - x_i^*\right), \quad b_i := \frac{f_i''(x_{i-1}^*)}{2f_i'(x_{i-1}^*)} \prod_{j=1}^{i-1} f_j'(x_{j-1}^*).$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

and its derivative $\frac{\partial G(u,x)}{\partial (u,x)}$ is given by

$$\begin{pmatrix} 1+2b_1u & -1 & & & \\ & 1+2b_2x_1 & -1 & & \\ & & \ddots & \ddots & \\ & & 1+2b_mx_{m-1} & -1 & \\ & & & 1+2b_{m+1}x_m & \end{pmatrix} + O\left(\left\|\begin{pmatrix} u \\ x \end{pmatrix}\right\|^2\right).$$

From this particular form, it follows first that

$$\frac{\partial G(u,x)}{\partial (u,x)} \cdot \begin{pmatrix} u \\ x \end{pmatrix} = \begin{pmatrix} u + 2b_1u^2 & - & x_1 \\ x_1 + 2b_2x_1^2 & - & x_2 \\ \vdots & & \\ x_{m_1} + 2b_m x_{m_1}^2 & - & x_m \\ x_m + 2b_{m+1}x_m^2 & & \end{pmatrix} + O\left(\left\|\begin{pmatrix} u \\ x \end{pmatrix}\right\|^3\right)$$

and second that

$$\frac{\partial G(u,x)}{\partial (u,x)}^{-1} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ & 1 & 1 & \dots & 1 \\ & & \ddots & \ddots & \vdots \\ & & & 1 & 1 \\ & & & & 1 \end{pmatrix} + O\left(\left\|\begin{pmatrix} u \\ x \end{pmatrix}\right\|\right).$$

Using again the formula for the Newton iteration

$$\begin{pmatrix} u \\ x \end{pmatrix}^{[k+1]} = \frac{\partial G(u^{[k]}, x^{[k]})}{\partial (u,x)}^{-1} \left(\frac{\partial G(u^{[k]}, x^{[k]})}{\partial (u,x)} \cdot \begin{pmatrix} u \\ x \end{pmatrix}^{[k]} - G(u^{[k]}, x^{[k]}) \right),$$

we obtain the iteration

$$\begin{pmatrix} u \\ x_1 \\ \vdots \\ x_{m-1} \\ x_m \end{pmatrix}^{[k+1]} = \begin{pmatrix} b_1 \left(u^{[k]}\right)^2 + \sum_{i=2}^{m+1} b_i \left(x_{i-1}^{[k]}\right)^2 \\ \sum_{i=2}^{m+1} b_i \left(x_{i-1}^{[k]}\right)^2 \\ \vdots \\ b_m \left(x_{m-1}^{[k]}\right)^2 + b_{m+1} \left(x_m^{[k]}\right)^2 \\ b_{m+1} \left(x_m^{[k]}\right)^2 \end{pmatrix} + O\left(\left\|\begin{pmatrix} u \\ x \end{pmatrix}^{[k]}\right\|^3\right).$$

It can be seen that, neglecting third order terms, the last component, x_m, converges independently from all others, with quadratic contraction constant b_{m+1}. All other components x_i converge based on their own quadratic contraction constant, b_{i+1}, and those of the higher indexed components, and the same holds for u. Thus, x_m is leading the convergence, with x_{m-1} as follower, etc., until u.

4.3. Comparison of lifted and nonlifted Newton methods. We have to compare the nonlifted quadratic convergence constant

$$\left(\sum_{i=1}^{m+1} b_i\right)$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

with the interdependent chain of quadratically converging sequences x_i in the lifted case, each with its dominant quadratic convergence constant b_{i+1}. Let us assume we start the nonlifted variant close to the solution with $u^{[0]} = \epsilon$, and the lifted variant with the corresponding values resulting from a forward function evaluation, which in our special setting turn out to be $x_i^{[0]} = \epsilon + O(|\epsilon|^2)$. As expected, the first step in u is identical in both methods and results in

$$u^{[1]} = \left(\sum_{i=1}^{m+1} b_i \right) \epsilon^2 + O(|\epsilon|^3).$$

However, in the lifted variant, the values x_i have been contracted according to their own contraction constants, to values

$$x_j^{[1]} = \left(\sum_{i=j+1}^{m+1} b_i \right) \epsilon^2 + O(|\epsilon|^3).$$

In the second iteration, the differing values for x will already lead to different iterates $u^{[2]}$. Which of the two methods converges faster depends on the signs of the b_i.

Same direction of curvature. If all b_i have the same sign, i.e., all subfunctions f_i are curved in the same direction, then the contraction constants for all x_i are better than the nonlifted variant, with the last component x_m converging the fastest. The improved convergence speed of x spills over to the convergence of u and therefore makes the lifted Newton method converge faster.

To see the effect in an example, let us regard the simplest setting with only one intermediate function evaluation, i.e., $m = 1$, with constants $b_1 = b_2 = 1$. After four iterations, the value of u in the nonlifted variant is $u^{[4]} = 2^{15}\, \epsilon^{16}$, while in the lifted variant it is $u^{[4]} = 677\, \epsilon^{16}$, which is more than two decimal digits more accurate.

Opposite directions of curvature. In the other extreme, let us regard a setting where all b_i add to zero but are each independently different from zero. Note that this can occur only if the subfunctions f_i have different directions of curvature. In this case, the nonlifted variant converges even faster than quadratically, while the lifted variant has the usual quadratic rate.

To see this in an even more extreme example, regard the simple chain of two functions $f_1(u) = \frac{1}{2}(1+u)^2 - \frac{1}{2}$ and $f_2(x) = \sqrt{1+2x} - 1$. These functions satisfy our assumptions with $b_1 = 1$ and $b_2 = -1$. Moreover, they are constructed such that $F(u) = u$. As F is a linear function, the nonlifted Newton method converges in the first iteration, $u^{[1]} = 0$, while the lifted Newton method performs the same favorable first step in u, but as x is not yet converged, it will continue iterating and changing u until both variables have been converged to sufficient accuracy.

Practical advice. In a practical application, even if we would have a chain of subsequent functions each depending only on the output of its predecessor, we do not know which local curvature constants the typically multi-input-multi-output functions f_i would have relative to each other, after the affine variable transformation based on linearization at the solution, to make them comparable. However, we might make an educated guess in the following case that occurs, e.g., in the simulation of continuous time dynamic systems: if we have repeated calls of the same function, i.e., $f_{i+1} = f_i$, and the variables x_{i+1} and x_i differ only slightly, then we can expect lifting to have a favorable effect on the required number of Newton iterations, even if both methods are initialized identically. A second case where a lifted approach surely is beneficial is the

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

case where the freedom for initializing the x_i based on extra knowledge can be used, e.g., when state measurement data are present in parameter estimation problems [2]. On the other hand, lifting should not be applied to simple linear subfunctions f_i, i.e., scalar multiplications and additions/subtractions, as no accelerated convergence can be gained, but memory requirement and operation counts are increased. In all other cases, we do not dare make predictions, but suggest experimenting with the lifted and nonlifted Newton methods in specific application examples. It is the aim of the present paper to propose an algorithmic trick and an open-source software package that makes the switching between the two methods as simple as possible. In the past, the implementation of structure-exploiting lifted algorithms was often a tedious task, deterring many users who might have benefited from the lifted approach.

The insight gained in this section can be expressed by the following theorem that characterizes the local convergence speed of lifted and nonlifted Newton methods.

THEOREM 2 (local convergence speed of lifted and nonlifted Newton methods). *Let $f_i : \mathbb{R} \to \mathbb{R}$, $1 \le i \le m+1$, be a chain of twice continuously differentiable scalar functions, such that $F(u) = f_{m+1}(f_m(\dots(f_1(u))\dots))$, $x_1 = f_1(u)$, and $x_i = f_i(x_{i-1})$, $2 \le i \le m+1$. Assume that the solution of the problem $F(u) = 0$ is given by u^* and that in the solution the Jacobians of $f_i, 1 \le i \le m+1$, are invertible. Define*

$$a_i = \prod_{j=1}^{i} f_j'(x_{j-1}^*), \qquad 1 \le i \le m+1, \tag{4.3a}$$

$$b_1 = \frac{f_1''(u^*)}{2f_1'(u^*)}, \tag{4.3b}$$

$$b_i = \frac{f_i''(x_{i-1}^*)}{2f_i'(x_{i-1}^*)} f_{i-1}'(x_{i-2}^*)\dots f_1'(u^*), \qquad 2 \le i \le m+1, \tag{4.3c}$$

with $x_i^* = f_i(f_{i-1}(\dots(f_1(u^*))\dots))$. Then the local convergence speed of a nonlifted Newton method for the solution of $F(u) = 0$ is given by

$$|u^{[k+1]} - u^*| \le \left(\sum_{i=1}^{m+1} b_i\right) |u^{[k]} - u^*|^2 + O(|u^{[k]} - u^*|^3),$$

and the local convergence of the lifted Newton iterates is componentwise staggered, following the estimation

$$\left|\frac{x_i^{[k+1]} - x^*}{a_i}\right| \le \sum_{j=i+1}^{m+1} b_j \left|\frac{x_{j-1}^{[k]} - x^*}{a_j}\right|^2 + O\left(\left\|\begin{pmatrix} u^{[k]} - u^* \\ x^{[k]} - x^* \end{pmatrix}\right\|^3\right), \quad 1 \le i \le m,$$

$$|u^{[k+1]} - u^*| \le b_1 |u^{[k]} - u^*|^2 + \sum_{j=i+1}^{m+1} b_j \left|\frac{x_{j-1}^{[k]} - x^*}{a_j}\right|^2 + O\left(\left\|\begin{pmatrix} u^{[k]} - u^* \\ x^{[k]} - x^* \end{pmatrix}\right\|^3\right).$$

4.3.1. Cost comparison. To compare the overall numerical effort for the solution of the example we analyze the cost of a single iteration in the lifted, nonlifted, and also full-space approach in more detail. We estimate computational effort in terms of floating point operations (flops). We follow the usual convention that an addition, subtraction, and multiplication, as well as a combined multiply-add cost 1 flop, while one division takes 4 flops. The cost in flops for the evaluation of the subfunctions f_i are denoted with c_{f_i}, where we set for notational convenience $f_{m+1} := f_F$. The cost

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

of computing their directional derivative is c_{df_i}. We refer to the cost of solving an n-dimensional linear equation system as $c_{LS}(n)$, which can in general be estimated with $c_{LS}(n) = \frac{2n^3+3n^2-5n}{6} + 2n(n+1)$ flops. For simplicity we assume one-dimensional node values and omit the cost of memory access. We split the cost for one iteration into the cost for (cf. Algorithm 4)

- evaluation of the residuals and the value of F,
- computation of the quantities of the Newton system,
- the solution of the Newton system to compute the step, and
- application of the step, which in the lifted case includes the cost of the expansion of the step in the controls u to the step in the nodes.

Figure 4.1 shows a comparison of these costs for the three approaches, as well as the resulting overall effort. Note that the cost estimation in the full-space approach is based on the assumption that no further internal structure of the problem is known and exploited other than the decomposition of $F(u)$ into the sequence of mappings f_i. Further exploitation of the internal structure might lead to the higher efficiency of a full-space approach, but already this relative simple exploitation of structure will lead to significantly higher implementation effort compared to the nonlifted and lifted approaches. Additionally, to be efficient, it has to be adjusted manually for each new problem (and also each new decomposition of F).

	Nonlifted	Lifted
Residual evaluation	$\sum_{i=1}^{m+1} c_{f_i}$	$\sum_{i=1}^{m+1} c_{f_i} + m$
Newton system	$n_u \sum_{i=1}^{m+1} c_{df_i}$	$(n_u+1)(\sum_{i=1}^{m+1} c_{df_i} + 2m) + n_u$
Step computation	$c_{LS}(n_u)$	$c_{LS}(n_u)$
Step application	n_u	$n_u + m + n_u m$

	Full-space
Residual evaluation	$\sum_{i=1}^{m+1} c_{f_i} + m$
Newton system	$\sum_{i=1}^{m+1} (n_u + i - 1) c_{df_i}$
Step computation	$c_{LS}(n_u + m)$
Step application	$n_u + m$

	Overall effort per iteration
Nonlifted	$c_{LS}(n_u) + \sum_{i=1}^{m+1} [c_{f_i} + n_u c_{df_i}] + n_u$
Lifted	$c_{LS}(n_u) + \sum_{i=1}^{m+1} [c_{f_i} + (n_u+1) c_{df_i}] + 3n_u m + 3m + 2n_u$
Full-space	$c_{LS}(n_u + m) + \sum_{i=1}^{m+1} [c_{f_i} + (n_u + i - 1) c_{df_i}] + 2m + n_u$

FIG. 4.1. *Cost analysis (in flops) for one iteration of a nonlifted, lifted, and full-space Newton method. Described are the costs for the different phases of an iteration (top, middle) and the overall cost for one iteration (bottom).* n_u *is the number of controls,* m *is the number of nodes, and* c_{f_i}, c_{df_i} *describe the cost of an evaluation, respectively, a directional derivative of the subfunction* f_i. *The quantity* $c_{LS}(n)$ *stands for the effort to solve a linear equation system with* n *unknowns.*

5. Numerical tests at tutorial and at large scale examples. We illustrate the numerical behavior of the lifted Newton methods in the examples of three tutorial problems and two larger parameter estimation problems, the second with 27 million internal variables. In each case we state the problem itself and how it is lifted and compare the performance of the lifted approach and the corresponding nonlifted version of the algorithm.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

5.1. The LiftOpt software package. All computations are performed on a Linux machine with a 3.0 GHz Pentium D CPU and 3 GB RAM using LiftOpt, a C++ software package which is available on the webpage http://www.liftopt.org including the examples presented in this paper. In the Gauss–Newton and SQP algorithms within LiftOpt, the QP solver qpOASES [5] is used to solve the QP subproblems. LiftOpt implements the three Newton-based optimization algorithms presented in this paper. The derivatives needed are computed efficiently by automatic differentiation using the C++ package ADOL-C [8] which has been coupled to LiftOpt. For the linear algebra operations the package ATLAS [16] is used. Also the reader's own optimization algorithms can be adapted to make use of the lifting idea. The authors are looking forward to hearing about interesting new applications.

5.2. A Newton example. In this first example we consider the root finding problem

$$F(u) := u^{16} - 2 = 0.$$

We introduce $m = 4$ intermediate values $x_1, \dots, x_4$ and lift the evaluation of F in the following way:

$$\begin{aligned} x_1 &:= u^2, & x_2 &:= x_1^2, \\ x_3 &:= x_2^2, & x_4 &:= x_3^2, \\ F &:= x_4 - 2. \end{aligned}$$

To solve the problem, we employ Algorithm 4 as well as a standard full-step Newton iteration applied to the nonlifted problem. The termination criterion is based on the Euclidean norm of the function value plus, in the lifted case, the Euclidean norm of the actual node residual. We require this sum to be smaller than 10^{-6}.

For the initial value $u_0 = 0.8$ we obtain convergence after 7 iterations in the lifted case and 27 iterations in the nonlifted. The progress of the iterates toward the solution during the iterations is depicted in Figure 5.1. The first iteration is identical, as is always the case if the node values x in the lifted algorithm are initialized by a function evaluation (cf. Algorithm 1). In subsequent iterations we observe that the lifted version benefits from the additional degrees of freedom which result in much faster progress toward the solution $u^* \approx 1.044$.

5.2.1. Convergence analysis for the example $F(u) := u^{16} - 2$. We analyze now the local convergence of the test example. F was decomposed into $f_i(x) = x^2$, $1 \le i \le 4$, and $f_5(x) = x - 2$, with the solution $(u^*, x_1^*, x_2^*, x_3^*, x_4^*) = (\sqrt[16]{2}, \sqrt[8]{2}, \sqrt[4]{2}, \sqrt{2}, 2)$. We then compute using Theorem 2

$$\begin{aligned} b_1 &= \frac{1}{2\sqrt[16]{2}} \approx 0.478801, \\ b_2 &= \frac{1}{2\sqrt[8]{2}} 2\sqrt[16]{2} = \frac{1}{\sqrt[16]{2}} \approx 0.957603, \\ b_3 &= \frac{1}{2\sqrt[4]{2}} 4\sqrt[16]{2}\sqrt[8]{2} = \frac{2}{\sqrt[16]{2}} \approx 1.91521, \\ b_4 &= \frac{1}{2\sqrt{2}} 8\sqrt[16]{2}\sqrt[8]{2}\sqrt[4]{2} = \frac{4}{\sqrt[16]{2}} \approx 3.83041, \\ b_5 &= 0. \end{aligned}$$

This leads to a nonlifted local quadratic contraction constant of $b \approx 7.18202$.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

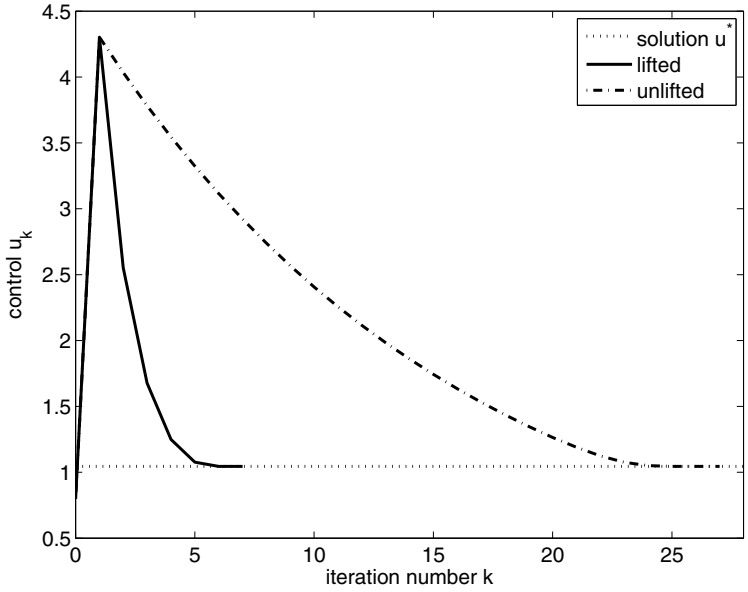


FIG. 5.1. *Iterates u_k of lifted and nonlifted approaches for the solution of $F(u) := u^{16} - 2 = 0$ and solution value $u^* \approx 1.044$. After an identical first iteration, which is due to the automatic initialization of the intermediate values using a function evaluation, the lifted method makes much faster progress toward the solution.*

Figure 5.2 shows the error of the iterates and the convergence rates for each iteration during solution with the nonlifted and the lifted Newton methods. The convergence rates are determined numerically for each component using the formula

$$\beta_y^{[k]} \approx \frac{\|y^{[k]} - y^*\|}{\|y^{[k+1]} - y^*\|^2}.$$

We observe, besides the faster convergence of the lifted iterations already described in section 5.2, that, as we approach the solution, the predicted convergence rates are reached in both cases. Additionally, it can be seen, as predicted by Theorem 2, that in the lifted case the components converge in a staggered way, starting with x_4, which converges due to linearity of f_5 in one step, followed by $x_3, x_2, \ldots$.

5.3. A Gauss–Newton toy example. For this example we consider the tutorial optimal control problem for a one-dimensional dynamical system from [4]:

$$\min_{x(\cdot),u(\cdot)} \int_0^3 |x(t)|^2 + |u(t)|^2 dt \tag{5.1a}$$

s.t.

$$\dot{x}(t) = x(t)\,(x(t)+1) + u(t), \tag{5.1b}$$

$$x(0) = x_0, \tag{5.1c}$$

$$x(3) = 0, \tag{5.1d}$$

$$|x(t)| \le 1, \tag{5.1e}$$

$$|u(t)| \le 1. \tag{5.1f}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

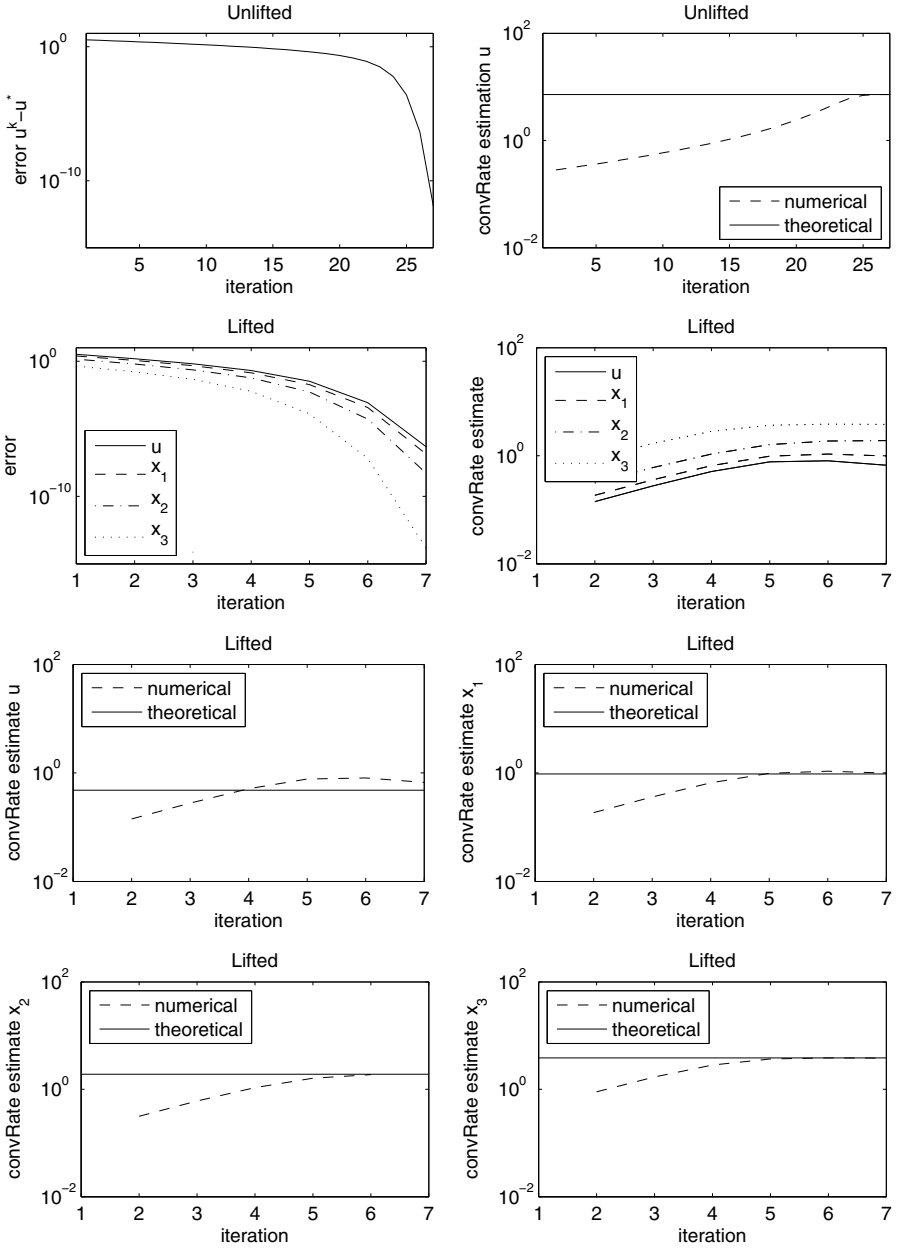


FIG. 5.2. *Shown is a comparison of the behavior of the iterates of the nonlifted and lifted Newton method on the root finding example* $f(u) = u^{16} - 2 = 0$ *with start value* $u_0 = 0.8$ *and tolerance* $\mathrm{tol} = 10^{-6}$. *Depicted is the Euclidean norm of the errors of the iterates (upper left pictures) and the local convergence estimates (upper right pictures), where in the lifted case the quantities are depicted componentwise with exception of* x_4, *as it converges after the first iteration due to the linearity of* f_5. *Additionally, the lower* 4 *pictures show for the lifted method for each component (except* x_4*) the convergence rate estimate as well as its theoretical prediction. We observe that in both the nonlifted and the lifted cases the convergence rates finally reach the predicted values. Furthermore, the lifted case shows the staggered convergence of the components that is predicted by Theorem* 2.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

The objective is to minimize the absolute value of the state over the whole time horizon while penalizing the control. Note that the system cannot be controlled and "blows up" if the state goes beyond $x_b \approx 0.619$.

To solve this infinite-dimensional problem, we divide, as is common in shooting methods, the time horizon into a grid with 30 equal subintervals of length 0.1 and discretize the controls to be piecewise constant on each of these intervals. The integral objective is approximated as a sum, where the function evaluations are made at the grid points. The state constraints are enforced also only at the grid points (this form of constraint discretization is a common approach, though more elaborated strategies exist to deal with semi-infinite constraints in a shooting context [12]). To solve the system dynamics we use Euler's method with timesteps equal to the subintervals. Doing this, we obtain a finite-dimensional NLP, which we solve using the proposed lifted Gauss–Newton algorithm as well as with the nonlifted version. The lifting is done by introducing the 30 state values at the grid points as intermediate values.

In this example, we employ two different initialization strategies when using the lifted approach. To begin with, we apply the lifted Gauss–Newton approach and use a function evaluation to initialize the intermediate values. Additionally, we use the fact that we want to minimize the absolute values of the states and set all intermediate values to zero. The convergence criterion is based on the sum of the Euclidean norm of the step in the controls, the Euclidean norm of the constraint violations, and, in the lifted case, the Euclidean norm of the residual vector. The tolerance was chosen to be 10^{-6}.

In Figure 5.3 we show a comparison of the results for different initial values x_0 of the dynamical system. The controls are initialized to zero on all subintervals in every case, which means that with growing initial state the problems become more difficult to solve. We observe that although the lifted approach performs in most cases already slightly better than the nonlifted one, we can still improve the performance considerably by using a priori information in node initialization, an advantage well known from the context of direct multiple shooting [2]. Furthermore, we observe that a reasonable initialization of the nodes makes the optimization more robust against bad initial guesses of the controls. This allows a quick solution, even when in the nonlifted or automatically initialized lifted algorithms the initial guess for the controls would lead to a blow-up of the system.

5.4. An SQP example. To test the proposed SQP algorithm, we consider again the optimal control problem (5.1). We use the same set-up as in the Gauss–Newton case with small modifications. The gradient of the Lagrangian is evaluated using the adjoint mode of automatic differentiation. To lift the system we introduce along with the system states at the grid points the corresponding adjoint values, which leads to 60 intermediate values. Again we apply and compare the proposed exact-Hessian SQP algorithm in three variants: (i) the nonlifted version, (ii) the lifted version using automatic node initialization, and (iii) the lifted version using a zero initialization. The results are shown in Figure 5.4. First, we observe that compared to the Gauss–Newton methods the SQP versions lag slightly behind, especially when we start at some distance from the solution. In this case the Gauss–Newton approximation leads to faster convergence than the exact-Hessian with its bad initial multiplier guesses. As we use here an exact-Hessian and undamped method, in one case we cannot avoid the bad luck of running into an area where the Hessian is not positive definite, leading the QP solver to quit the iterations. When started closer to the solution by zero initialization, we see that the SQP method converges faster than Gauss–Newton due

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

x_0	# iterations unlifted	#iterations lifted (autom. init.)	#iterations lifted (zero init.)
0.02	5	5	4
0.03	6	5	4
0.04	6	6	4
0.05	7	6	4
0.06	8	7	5
0.07	9	7	5
0.08	10	8	5
0.09	13	10	5
0.10	17	13	5
0.20	err_{nan}	err_{nan}	6
0.30	err_{nan}	err_{nan}	7

FIG. 5.3. *Results of the Gauss–Newton approaches for the optimal control example described in section* 5.3. *Shown are the numbers of iterations needed until convergence for different initial states* x_0 *of the dynamical system.* err_{nan} *denotes that the run was not successful because the system "blew up" during integration at some iterate, such that the QP solver quits due to "nan"-values. Compared here are the nonlifted Gauss–Newton approach, the lifted Gauss–Newton approach with automatic initialization of the intermediate values by system integration, and the lifted Gauss–Newton approach when started with nodes initialized to zero.*

x_0	# iterations unlifted	#iterations lifted (autom. init.)	#iterations lifted (zero init.)
0.02	6	5	3
0.03	7	5	3
0.04	8	6	3
0.05	9	6	3
0.06	10	6	3
0.07	12	7	4
0.08	15	8	4
0.09	19	11	4
0.10	25	err_{pd}	4
0.20	err_{nan}	err_{nan}	4
0.30	err_{nan}	err_{nan}	4

FIG. 5.4. *Results of the SQP approaches for the optimal control example described in section* 5.4. *Shown are the numbers of iterations needed until convergence for tolerance* $\text{tol} = 10^{-6}$ *and different initial states* x_0 *of the dynamical system.* err_{nan} *denotes that the run was not successful because the system "blew up" during integration at some iterate, such that the QP solver quits due to "nan"-values.* err_{pd} *denotes that the run was not successful because at some point the Hessian became indefinite, leading to an exit of the convex QP solver. Compared here are the nonlifted SQP approach, the lifted SQP approach with automatic initialization of the intermediate values by system integration, and the lifted SQP approach when started with nodes initialized to zero.*

to the better local convergence properties. Besides that, comparing the lifted and nonlifted versions of the SQP, we again see a better performance of the lifted version, and again the zero initialization leads to much faster convergence and to a more robust behavior.

5.5. Parameter estimation example 1. In this example we want to estimate the parameters $c = (c_0, c_1, c_2)$ in the following model for a dynamical system on the

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

domain $\Omega = (0,1) \times (0,1)$:

$$\begin{aligned}
\partial_t u(t,x_1,x_2) - d(x_1,x_2)\Delta u(t,x_1,x_2) &= 0, \qquad (x_1,x_2) \in \Omega, t \in [0,1],\\
d(x_1,x_2) &= c_0 + c_1 x_1 + c_2 x_2,\\
u(t,x_1,x_2) &= 100, \qquad (x_1,x_2) \in \{0\} \times [0,1], t \in [0,1],\\
u(t,x_1,x_2) &= 0, \qquad (x_1,x_2) \in \partial\Omega \setminus (\{0\} \times [0,1]), t \in [0,1].
\end{aligned}$$

To simulate the system, we discretize it using the method of lines with finite differences and gridsize $h = 0.1$, resulting in a coarse space grid of size 11×11 . The timestepping is done, as this is only an educational example, using an explicit Euler method with equidistant timesteps. The step length was chosen as $h = 0.005$; i.e., the number of timesteps is $n = 200$. We assume that at every 10 timesteps measurements over the entire domain are made. The least-squares objective function is to minimize the quadratic deviation of the system states from the measured data in Euclidean norm, summed up over all 20 measurements. This means we have a residual vector $F_1(c)$ of dimension $9 \cdot 9 \cdot 20 = 1620$ (the first part of the "user function"). The objective is $\|F_1(c)\|_2^2$, and as constraints we impose that neither the parameter nor the states should become negative, $F_2(c) \geq 0$ (the second part of the "user function"). The convergence criterion is based on the sum of the Euclidean norm of the step in the controls, the Euclidean norm of the constraint violation, and, in the lifted case, the Euclidean norm of the residual vector. The tolerance was chosen to be 10^{-6}. For the numerical test we generate measurement data by simulating the system for $c_0 = 0.01, c_1 = 0.08, c_2 = 0.16$ and perturb with Gaussian noise with mean value 0 and standard deviation 0.1. We apply the lifted and nonlifted Gauss–Newton approaches to solve the problem. The lifting is done by introducing all 1620 system states at the measurement times as node values. In addition to automatic initialization by system simulation, we test the lifted algorithm using the measurement data as initialization of the node values. The results for different initial guesses of the parameters are displayed in Figure 5.5. If we start close to the true parameter, the lifting does not improve the performance much, as convergence is reached quite quickly anyway in this case. On the other hand, if we start with worse initial guesses for the parameter, the lifted approach leads to a significantly faster convergence. When additionally initialized with the measurement data, the gain becomes quite impressive. This behavior can be seen in Figure 5.6, which illustrates the progress in the objective function value for the run to the initial guess $c_0 = 0.5, c_1 = 0.5, c_2 = 0$.

Note that for this problem class, where the variables correspond directly to a discretization of a continuous function, also hierarchical full-space approaches have been shown to be very promising; see, e.g., [6] and references therein. They take the special discretization nature into account by working on different levels, corresponding to coarser, respectively, finer discretizations of the problem and might compete with the lifted Newton method. Note, however, that they require major manipulations of the "user function," while using LiftOpt requires only insertion of the function "node" after definition of any internal variable that shall be lifted.

5.6. Parameter estimation example 2 (27 million internal variables). As a second parameter estimation example we choose a shallow water equation model describing wave propagation in a basin that we found on the Internet [3] in search of some truly large scale "user function" for use within LiftOpt. The model is described

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

c_0	c_1	c_2	# iterations unlifted	#iterations lifted (autom. init.)	#iterations lifted (meas init.)
0.0	0.0	0.0	10	9	6
0.1	0.1	0.1	5	5	4
0.2	0.2	0.2	6	6	5
0.25	0.25	0.25	42	29	8
0.5	0.5	0.5	244	114	77
0.5	0.5	0.0	159	78	27
0.0	0.5	0.5	101	53	22
0.6	0.0	0.0	71	37	10
0.7	0.0	0.0	139	56	12

FIG. 5.5. *Results of the parameter estimation example described in section* 5.5. *Shown are the numbers of iterations needed until convergence for tolerance* $\text{tol} = 10^{-6}$ *and different sets of initial parameter guesses. Compared here are the nonlifted Gauss–Newton approach, the lifted Gauss–Newton approach with automatic initialization of the intermediate values by system simulation, and the lifted approach when initializing the node values using the measurement data. The "true" parameter values are* $c_0 = 0.01, c_1 = 0.08, c_2 = 0.16$.

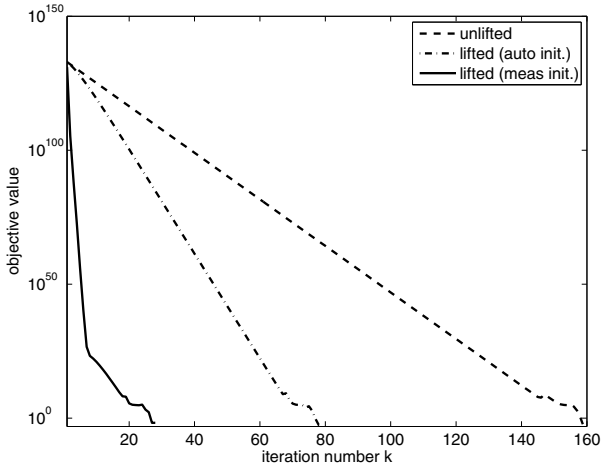


FIG. 5.6. *The behavior of the objective function value for a run of the parameter estimation example described in section* 5.5. *The run is started with the initial parameter guesses* $c_0 = 0.5, c_1 = 0.5, c_2 = 0$. *The "true" parameter values are* $c_0 = 0.01, c_1 = 0.08, c_2 = 0.16$. *Compared here are the nonlifted Gauss–Newton approach, the lifted Gauss–Newton approach with automatic initialization of the intermediate values by system simulation, and the lifted approach when initializing the node values using the measurement data. We observe that the first iterations of the nonlifted and the automatically initialized lifted versions are again identical, as expected. Afterward the lifted version converges considerably faster but is easily outperformed by the lifted version where the measurement data were used for initialization.*

by a system of hyperbolic PDEs, and the equations are

$$\begin{aligned}
\partial_t u(t,x,y) &= -g\partial_x h(t,x,y) - bu(t,x,y),\\
\partial_t v(t,x,y) &= -g\partial_y h(t,x,y) - bv(t,x,y),\\
\partial_t h(t,x,y) &= -H\left[\partial_x u(t,x,y) + \partial_y v(t,x,y)\right],
\end{aligned}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

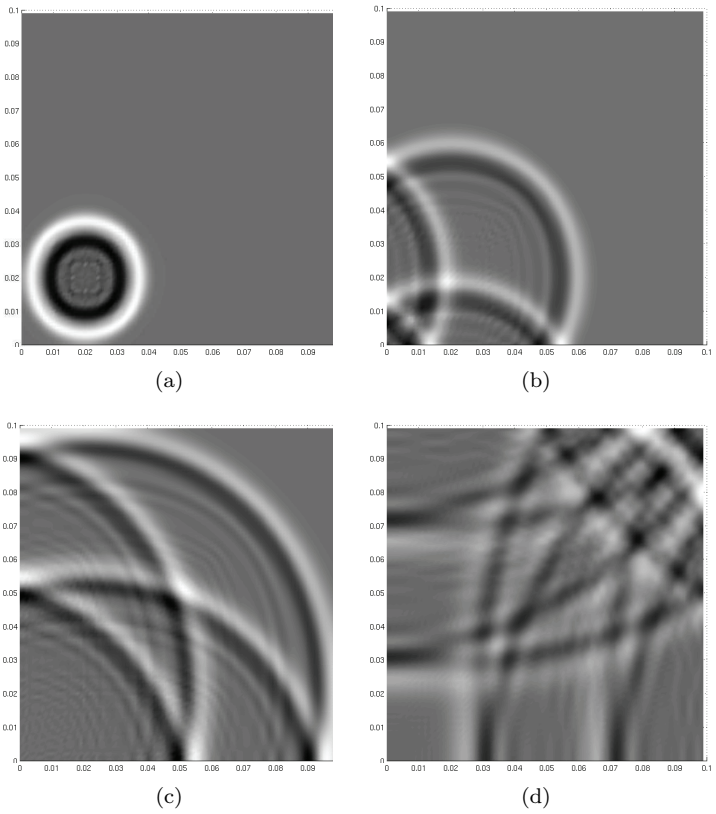


FIG. 5.7. *Numerical solution for component h of the shallow water equation model described in section* 5.6 *for the true values $b = 2$ and $H = 0.01$, depicted at timepoints* (a) $t = 0.1$, (b) $t = 0.25$, (c) $t = 0.5$, *and* (d) $t = 1.0$.

where u, v are the horizontal and vertical water velocities, h is the deviation of the water surface from the mean water height H, $g \approx 9.81$ is the gravitational constant, and b is the viscous drag. As "true" values for b and H we use $b = 2$ and $H = 0.01$. For the numerical test we assume a quadratic basin corresponding to $\Omega = (0, 0.2) \times (0, 0.2)$ and consider the time horizon $t \in [0, 1]$. Furthermore, we assume that the basin is bounded by walls that reflect the incoming waves. We use an equidistant discretization in space of 30-by-30 grid points, finite differences in space, and for timestepping we use the stepsize $dt = 10^{-4}$ with an explicit Euler scheme, resulting in $3{\cdot}30{\cdot}30{\cdot}10^4 = 27{\cdot}10^6$ internal variables. In our scenario we start with a plain surface and add at start time a splash of height 0.01 and radius 0.03. The numerical solution is depicted for component h in Figure 5.7. During system simulation we take measurements only of component h every 100th timestep.

The least-squares objective function we use in the parameter estimation to determine b and H is the quadratic deviation of h from the measured data in Euclidean norm, summed up over all 90000 measurements. As constraints we impose that neither b nor H should become negative. The convergence criterion is based on the sum of the Euclidean norm of the step in the controls, the Euclidean norm of the constraint violations, and, in the lifted case, the Euclidean norm of the residual vector. The tolerance was chosen to be 10^{-6}. We apply the lifted and nonlifted Gauss–Newton

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

approaches to solve the problem. The lifting is done by introducing the values of h at the measurement times as nodes, leading to overall 90000 nodes. We again test in the lifted case using automatic node initialization as well as using the measurement data for node initialization. The results for different initial guesses of b and H are displayed in Figure 5.8. The average time needed for one iteration in the unlifted case is 8.86s, while one lifted iteration takes on average 11.81s. Note that this difference in the effort for one lifted versus one unlifted iteration will usually be smaller for problems with a larger number of degrees of freedom.

Similar to the previous example, lifting does not improve the performance much if we start close to the true parameter values. On the other hand, if we start further away from the solution, the lifted approach again leads to a significantly faster convergence, especially for perturbations in the parameter H. When the lifted approach is initialized with the measurement data, the performance is even better, although the gain is not as impressive as in the previous example. This might be due to the fact that only a part of the system state, i.e., h, is measured here.

An important remark is that the problem under consideration could in principle also be formulated in AMPL and solved by a full-space algorithm such as that implemented in IPOPT, but this would result in a sparse NLP with 27 million variables.

b	H	# iterations unlifted	#iterations lifted (autom. init.)	#iterations lifted (meas init.)
0.5	0.01	5	5	4
5	0.01	6	5	4
15	0.01	17	7	6
30	0.01	27	7	6
2	0.005	31	9	5
2	0.02	38	12	5
2	0.1	44	13	8
0.2	0.001	33	12	7
1	0.005	47	10	5
4	0.02	56	10	5
1	0.02	44	9	6
20	0.001	24	10	6

FIG. 5.8. *Results of the parameter estimation example for the shallow water equation model described in section* 5.6. *Shown are the numbers of iterations needed until convergence for tolerance* $\text{tol} = 10^{-6}$ *and different sets of initial parameter guesses. Compared here are the nonlifted Gauss–Newton approach, the lifted Gauss–Newton approach with automatic initialization of the intermediate values by system simulation, and the lifted approach when using the measurement data for node initialization. The "true" parameter values are* $b = 2$ *and* $H = 0.01$. *The average time needed for one unlifted iteration is* $8.86s$ *versus* $11.81s$ *for one lifted iteration.*

6. Conclusion. In this article we have developed, starting with Newton's method and inspired by the multiple shooting method, a more general "lifting" approach for Newton-type methods. We have demonstrated how the needed quantities for lifted algorithms can be obtained at practically no additional cost per iteration and with very few modifications to the original function code. For the use in optimization, a lifted Gauss–Newton and an adjoint-based lifted SQP method were proposed, and the equivalence of the latter with full-space exact-Hessian SQP iterations was shown. We have given a theorem that underpins the observed superior local convergence speed

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

of lifted Newton methods by a theoretical analysis in a simplified setting. Finally, the potential of the proposed lifting algorithms was illustrated by numerical tests for several examples, one with 27 million intermediate variables, 90000 of which were lifted. The lifted Newton method for optimization and all examples of this paper are available as open-source C++ code in the package LiftOpt on http://www.liftopt.org.

Appendix. Equivalence of lifted SQP and full-space iterations in the constrained case.

We consider here the equality constrained nonlinear optimization problem

$$\min_u \varphi(u) \tag{A.1a}$$

$$\text{s.t.}$$

$$h(u) = 0, \tag{A.1b}$$

with $\varphi \in \mathcal{C}^2(\mathbb{R}^{n_u}, \mathbb{R})$, $h \in \mathcal{C}^2(\mathbb{R}^{n_u}, \mathbb{R}^{n_{eq}})$ to show the equivalence and to explain the computation of the reduced quantities. The results follow directly for the case of inequality constraints from the equivalence of Newton's method for the KKT-conditions and exact-Hessian SQP.

Augmented problem. Introducing additionally intermediate values $w_i, 1 \le i \le n_w$, occurring during evaluation of f and h as variables, we obtain the equivalent augmented full-space problem

$$\min_{u,w} f_\varphi(u, w) \tag{A.2a}$$

$$\text{s.t.}$$

$$g(u, w) = 0, \tag{A.2b}$$

$$f_h(u, w) = 0, \tag{A.2c}$$

with $f_\varphi \in \mathcal{C}^2(\mathbb{R}^{n_u} \times \mathbb{R}^{n_w}, \mathbb{R})$, $g \in \mathcal{C}^2(\mathbb{R}^{n_u} \times \mathbb{R}^{n_w}, \mathbb{R}^{n_w})$, $f_h \in \mathcal{C}^2(\mathbb{R}^{n_u} \times \mathbb{R}^{n_w}, \mathbb{R}^{n_{eq}})$.

KKT-conditions for the full-space problem. We define the Lagrangian of the augmented problem

$$\mathcal{L}(u, w, \lambda, \mu) := f_\varphi(u, w) + \lambda^T g(u, w) + \mu^T f_h(u, w).$$

The necessary first order optimality conditions are then

$$\nabla\mathcal{L}(u^*, w^*, \lambda^*, \mu^*) := \begin{pmatrix} \nabla_u f_\varphi(v^*) + \nabla_u g(v^*)\lambda^* + \nabla_u f_h(v^*)\mu^* \\ \nabla_w f_\varphi(v^*) + \nabla_w g(v^*)\lambda^* + \nabla_w f_h(v^*)\mu^* \\ g(v^*) \\ f_h(v^*) \end{pmatrix} \stackrel{!}{=} 0. \tag{A.3}$$

Here and in the following $\nabla_y \psi \equiv \frac{\partial \psi}{\partial y}^T$ denotes the transpose of the Jacobian of a function ψ with respect to y , $\nabla^2_{yz}\psi \equiv \frac{\partial^2 \psi}{\partial y \partial z}$ denotes the (here mixed) Hessian, and $v = (u, w, \lambda, \mu)$. For improved readability we sometimes omit the arguments of the functions if no ambiguity is caused.

Full-space iterations. We use Newton's method to solve the system for the KKT-conditions, which is then an exact-Hessian SQP method. It iterates

$$\begin{pmatrix} \nabla^2_{uu}\mathcal{L} & \nabla^2_{uw}\mathcal{L} & \nabla_u g & \nabla_u f_h \\ \nabla^2_{wu}\mathcal{L} & \nabla^2_{ww}\mathcal{L} & \nabla_w g & \nabla_w f_h \\ \nabla_u g^T & \nabla_w g^T & 0 & 0 \\ \nabla_u {f_h}^T & \nabla_w {f_h}^T & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \Delta u \\ \Delta w \\ \Delta\lambda \\ \Delta\mu \end{pmatrix} = - \begin{pmatrix} \nabla_u\mathcal{L} \\ \nabla_w\mathcal{L} \\ g \\ f_h \end{pmatrix}. \tag{A.4}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

From the assumed structure of g,

$$g(u,w) = \begin{pmatrix} f_1(u) & -w_1 \\ f_2(u,w_1) & -w_2 \\ \vdots & \vdots \\ f_m(u,w_1,\dots,w_{m-1}) & -w_m \end{pmatrix}, \tag{A.5}$$

we have

$$\nabla_w g^T(u,w) = \begin{pmatrix} -\mathbb{I}_{n_{w_1}} & 0 & \dots & 0 \\ \frac{\partial f_2}{\partial w_1} & -\mathbb{I}_{n_{w_2}} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ \frac{\partial f_m}{\partial w_1} & \dots & \frac{\partial f_m}{\partial w_{m-1}} & -\mathbb{I}_{n_{w_m}} \end{pmatrix}, \tag{A.6}$$

and thus $\nabla_w g$ and $\nabla_w g^T$ are invertible. Therefore, we can solve the third equation of (A.4) for Δw and the second equation for $\Delta\lambda$ to obtain

$$\Delta w = a_x + A_x^u \Delta u \qquad \text{with } a_x := -\nabla_w g^{-T} g,\ A_x^u := -\nabla_w g^{-T} \nabla_u g^T \tag{A.7}$$

and

$$\Delta\lambda = a_\lambda + A_\lambda^u \Delta u + A_\lambda^\mu \Delta\mu \tag{A.8}$$

with

$$a_\lambda := -\nabla_w g^{-1} \left(\nabla_w \mathcal{L} + \nabla^2{}_{ww} \mathcal{L}(-\nabla_x g^{-T} g) \right), \tag{A.8a}$$

$$A_\lambda^u := -\nabla_w g^{-1} \left(\nabla^2{}_{ww} \mathcal{L}(-\nabla_w g^{-T} \nabla_u g^T) + \nabla^2{}_{wu} \mathcal{L} \right), \tag{A.8b}$$

$$A_\lambda^\mu := -\nabla_w g^{-1} \nabla_w f_h. \tag{A.8c}$$

Now we can condense the problem to

$$\begin{aligned} &\begin{pmatrix} \nabla^2{}_{uu}\mathcal{L} + \nabla^2{}_{uw}\mathcal{L} A_x^u + \nabla_u g A_\lambda^u & \nabla_u f_h + \nabla_u g A_\lambda^\mu \\ \nabla_u {f_h}^T + \nabla_w {f_h}^T A_x^u & 0 \end{pmatrix} \cdot \begin{pmatrix} \Delta u \\ \Delta\mu \end{pmatrix} \\ &\qquad = -\begin{pmatrix} \nabla_u \mathcal{L} + \nabla^2{}_{uw}\mathcal{L}\, a_x + \nabla_u g\, a_\lambda \\ f_h + \nabla_w {f_h}^T a_x \end{pmatrix}, \end{aligned} \tag{A.9}$$

solve for $\Delta u, \Delta\mu$, and afterward expand to $\Delta w, \Delta\lambda$ using (A.7) and (A.8) to obtain the complete full-space SQP step. Note that this system is symmetric of the form

$$\begin{pmatrix} B_1 & B_2^T \\ B_2 & 0 \end{pmatrix} \cdot \begin{pmatrix} \Delta u \\ \Delta\mu \end{pmatrix} = -\begin{pmatrix} \nabla_u \mathcal{L} + \nabla^2{}_{uw}\mathcal{L}\, a_x + \nabla_u g\, a_\lambda \\ f_h + \nabla_w {f_h}^T a_x \end{pmatrix}, \tag{A.10}$$

with symmetric

$$\begin{aligned} B_1 = \nabla^2{}_{uu}\mathcal{L} &- \nabla^2{}_{uw}\mathcal{L} \nabla_w g^{-T} \nabla_u g^T - \nabla_u g \nabla_w g^{-1} \nabla^2{}_{wu}\mathcal{L} \\ &+ \nabla_u g \nabla_w g^{-1} \nabla^2{}_{ww}\mathcal{L} \nabla_w g^{-T} \nabla_u g^T \end{aligned} \tag{A.11}$$

and

$$B_2 = \nabla_u {f_h}^T - \nabla_w {f_h}^T \nabla_w g^{-T} \nabla_u g^T. \tag{A.12}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Lifted Newton formulation. We assume that the combined evaluation of

$$\begin{pmatrix} \nabla_u \mathcal{L}^{\text{orig}}. \\ h \end{pmatrix} = \begin{pmatrix} f_u + \nabla_u h^T \mu \\ h \end{pmatrix} \tag{A.13}$$

is lifted in such a way that the computation of $\mathcal{L}$ and h use common intermediate values w which are introduced as nodes. Furthermore, the evaluation of $\nabla_u \mathcal{L}$ is done in the way of the adjoint mode of automatic differentiation, and the corresponding adjoint values $\bar{w}$ are also introduced as nodes.

The nodes $(w, \bar{w})$ are then defined by the equations

$$G(u, w, \bar{w}, \mu) = \begin{pmatrix} G_1(u, w) \\ \hline G_2(u, w, \bar{w}, \mu) \end{pmatrix} = \begin{pmatrix} g(u, w) \\ \hline G_2(u, w, \bar{w}, \mu) \end{pmatrix} \tag{A.14}$$

(A.15)

$$= \left(\begin{array}{cc} f_1(u) & -w_1 \\ f_2(u, w_1) & -w_2 \\ \vdots & \\ f_m(u, w_1, \ldots, w_{m-1}) & -w_m \\ \hline \frac{\partial f_\varphi(u, w_1, \ldots, w_m)}{\partial w_m} + \frac{\partial f_h(u, w_1, \ldots, w_m)}{\partial w_m}^T \mu & -\bar{w}_m \\ \vdots & \\ \frac{\partial f_\varphi(u, w_1, \ldots, w_m)}{\partial w_2} + \frac{\partial f_h(u, w_1, \ldots, w_m)}{\partial w_2}^T \mu + \sum_{i=3}^m \frac{\partial f_i(u, w_1, \ldots, w_{i-1})}{\partial w_2}^T \bar{w}_i & -\bar{w}_2 \\ \frac{\partial f_\varphi(u, w_1, \ldots, w_m)}{\partial w_1} + \frac{\partial f_h(u, w_1, \ldots, w_m)}{\partial w_1}^T \mu + \sum_{i=2}^m \frac{\partial f_i(u, w_1, \ldots, w_{i-1})}{\partial w_1}^T \bar{w}_i & -\bar{w}_1 \end{array} \right) = 0,$$

and for the gradient $\nabla_u \mathcal{L}$ of the Lagrangian we have

(A.16)

$$\nabla_u \mathcal{L}(u, w, \bar{w}, \mu) = \left(\frac{\partial f_\varphi(u, w_1, \ldots, w_m)}{\partial u} + \frac{\partial f_h(u, w_1, \ldots, w_m)}{\partial u}^T \mu + \sum_{i=1}^m \frac{\partial f_i(u, w_1, \ldots, w_{i-1})}{\partial u}^T \bar{w}_i \right).$$

Following the lifting approach, we then apply Newton to solve the root finding problem

$$\begin{pmatrix} \nabla_u \mathcal{L}(u, w, \bar{w}, \mu) \\ f_h(u, w) \\ g(u, w) \\ G_2(u, w, \bar{w}, \mu) \end{pmatrix} = 0. \tag{A.17}$$

We compare now this system with system (A.4) from the full-space approach. We observe that, if identifying $\bar{w}_i$ with λ_i and exchanging the second and fourth equations in system (A.16), the systems are identical. As a result, the Newton method we apply to these systems in both cases will lead to the same iterations, which shows the equivalence of the lifted SQP approach and full-space exact-Hessian iterations. Note that by construction the lifted approach leads to the same condensed system as the full-space approach.

Furthermore, we observe that, for the computation of B_1 and B_2 of the symmetric system (A.10), by using the modified function of the lifting approach only directional derivatives in u are needed. After the computation of Δu and $\Delta \mu$ from the condensed system, we first can expand the step to Δw via (A.7), as a_x and A_x^u have been computed together with B_1, B_2. Afterward we need one additional directional derivative of the modified function to expand the step to $\Delta \lambda$ via (A.8).

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Acknowledgment. Both authors thank the anonymous reviewers for their insightful comments that helped to improve the paper, in particular by requiring the theoretical analysis presented in section 4.

REFERENCES

[1] H. G. Bock, *Numerical solution of nonlinear multipoint boundary value problems with applications to optimal control*, Z. Angew. Math. Mech., 58 (1978), pp. 407–409.

[2] H. G. Bock, *Randwertproblemmethoden zur Parameteridentifizierung in Systemen nichtlinearer Differentialgleichungen*, Bonner Mathematische Schriften 183, Universität Bonn, Bonn, 1987.

[3] D. Copsey, *Wave model and simulation code of D. Copsey*, available online at http://www.dcopsey.i12.com/.

[4] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber, *Fast direct multiple shooting algorithms for optimal robot control*, in Fast Motions in Biomechanics and Robotics, Lecture Notes in Control and Inform. Sci. 340, Springer-Verlag, New York, 2006, pp. 65–93.

[5] H. J. Ferreau, H. G. Bock, and M. Diehl, *An online active set strategy to overcome the limitations of explicit MPC*, Internat. J. Robust Nonlinear Control, 18 (2008), pp. 816–830.

[6] S. Gratton, A. Sartenaer, and P. L. Toint, *Recursive trust-region methods for multiscale nonlinear optimization*, SIAM J. Optim., 19 (2008), pp. 414–444.

[7] A. Griewank, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, Frontiers Appl. Math. 19, SIAM, Philadelphia, 2000.

[8] A. Griewank, D. Juedes, and J. Utke, *Algorithm 755: ADOL-C: A package for the automatic differentiation of algorithms written in C/C++*, ACM Trans. Math. Software, 22 (1996), pp. 131–167.

[9] J. Kallrath, H. G. Bock, and J. P. Schlöder, *Least squares parameter estimation in chaotic differential equations*, Celestial Mech. Dynam. Astronom., 56 (1993), pp. 353–371.

[10] D. Leineweber, *Efficient reduced SQP methods for the optimization of chemical processes described by large sparse DAE models*, Fortschritt-Berichte VDI Reihe 3 613, Verfahrenstechnik, VDI Verlag, Düsseldorf, 1999.

[11] M. Osborne, *On shooting methods for boundary value problems*, J. Math. Anal. App., 27 (1969), pp. 417–433.

[12] A. Potschka, H. G. Bock, and J. P. Schlöder, *A minima tracking variant of semi-infinite programming for the treatment of path constraints within direct solution of optimal control problems*, Optim. Methods Softw., 24 (2009), pp. 237–252.

[13] A. Schäfer, *Efficient Reduced Newton-Type Methods for Solution of Large-Scale Structured Optimization Problems with Application to Biological and Chemical Processes*, Ph.D. thesis, Universität Heidelberg, Heidelberg, Germany, 2005.

[14] J. P. Schlöder, *Numerische Methoden zur Behandlung hochdimensionaler Aufgaben der Parameteridentifizierung*, Bonner Mathematische Schriften 187, Universität Bonn, Bonn, 1988.

[15] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, Springer-Verlag, New York, 1992.

[16] R. C. Whaley, A. Petitet, and J. Dongarra, *Automated empirical optimization of software and the ATLAS project*, Parallel Comput., 27 (2001), pp. 3–35.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.