An efficient trust region method for unconstrained discrete-time optimal control problems[1]

Thomas F. Coleman[2] and Aiping Liao[3]

July 8, 1993

Abstract

Discrete-time optimal control (DTOC) problems are large-scale optimization problems with a dynamic structure. In previous work this structure has been exploited to provide very fast and efficient local procedures. Two examples are the differential dynamic programming algorithm (DDP) and the stagewise Newton procedure – both require only $O(N)$ operations per iteration, where N is the number of timesteps. Both exhibit a quadratic convergence rate. However, most algorithms in this category do not have a satisfactory global convergence strategy. The most popular global strategy is shifting: this sometimes works poorly due to the lack of automatic adjustment to the shifting element.

In this paper we propose a method that incorporates the trust region idea with the local stagewise Newton's method. This method possesses advantages of both the trust region idea and the stagewise Newton's method, i.e., our proposed method has strong global and local convergence properties yet remains economical. Preliminary numerical results are presented to illustrate the behavior of the proposed algorithm. We also collect in the Appendix some DTOC problems that have appeared in the literature.

[1]Partially supported by the Cornell Theory Center, which receives major funding from the National Science Foundation and IBM Corporation, with additional support from the State of New York and its Corporate Research Institutes; and by NSF, AFOSR, and ONR through grant DMS-8920550.

[2]Computer Science Department, Cornell University, Ithaca, New York 14853.

[3]Advanced Computing Research Institute and Center for Applied Mathematics, Cornell University, Ithaca, New York 14853.

1 Introduction

Discrete-time optimal control (DTOC) problems arise in many practical applications including multi-reservoir control problems[14], the treatment of polluted groundwater [3], and inventory control [2]. In this paper we are concerned with the unconstrained discrete-time optimal control problem,

$$\begin{array}{ll} \min F := \sum_{i=1}^{N-1} L_i(y_i, x_i) + L_N(y_N) & \\ y_{i+1} = T_i(y_i, x_i), \quad i = 1, \ldots, N-1 & \qquad\qquad (\mathrm{P}) \\ y_1 = \bar{y}_1. & \end{array}$$

The vectors $y_i \in R^{n_y}, i = 1, \ldots, N$ are called state variables and the vectors $x_i \in R^{n_x}, i = 1, \ldots, N-1$ are called control variables, $\bar{y}_1$ is a constant vector;

$$F = F(y, x) : R^{n_y N \times n_x (N-1)} \to R$$

where y is a vector in $R^{n_y N}$ and x is a vector in $R^{n_x(N-1)}$ (we put them in the matrix form, i.e., $y = [y_1, \ldots, y_N]$ and $x = [x_1, \ldots, x_{N-1}]$). We assume that

$$T_i(y_i, x_i) : R^{n_y \times n_x} \to R, i = 1, \ldots, N-1$$

and $L_i,\ i = 1, \ldots, N$ are all twice continuously differentiable functions. Throughout this paper we denote $n = n_x(N-1)$, y_i the i-th state variable, x_i the i-th control variable, $y_{i,j}$ the j-th component of the i-th state variable, $x_{i,j}$ the j-th component of the i-th control variable and $T_{i,j}$ the j-th component of the i-th transition function.

Obviously, problem (P) is an optimization problem with a dynamic structure. Taking advantage of this structure, and using the principle of optimality, Mayne [10] and Jacobson and Mayne [6] propose a second order algorithm called the differential dynamic programming (DDP) algorithm for solving (P).

It has been proven, see Murray and Yakowitz [14] for example, that the DDP algorithm is locally quadratically convergent. However, we note that most DDP-like algorithms do not have a satisfactory global convergence strategy. For example, Liao and Shoemaker [7] propose an "adaptive shift" procedure which guarantees global convergence. This procedure is similar to the method proposed in Luenberger [9]: the smallest eigenvalue of each of the stagewise Hessian matrices $\{C_i\}$ needs to be calculated. If the matrices $\{C_i\}$ are large the adaptive shift procedure can be very expensive. Another disadvantage of this kind procedure is that the value of the shifting parameter is not easy to choose.

In the next section we propose a new method for problem (P): this method has strong global convergence properties, local quadratic convergence and a remarkably low computational cost per iteration. In section 3 numerical results are presented to illustrate the behavior of our algorithm.

2 Description of the algorithm

In this section we describe our algorithm which combines the stagewise Newton's method of Pantoja [16] with the trust region method of Nocedal and Yuan [15].

2.1 The stagewise Newton's method

Pantoja [16] proposes a modified DDP method for solving (P) which produces iterates identical to those that Newton's method would produce for problem $(\bar{P})$:

$$\begin{array}{ll} \min f(x) & \\ x \in R^n & \qquad (\bar{P}) \end{array}$$

which is obtained by eliminating the state variables in (P). Pantoja's algorithm is regarded as a stagewise Newton's method. Unlike the conventional Newton's method for $(\bar{P})$ which requires $\Omega(N^3)$ works per iteration[1] Pantoja's algorithm needs only $O(N)$ operations per iteration.

Pantoja [16] shows that the DTOC problem (P) can be transformed into the following equivalent problem:

$$\begin{array}{ll} \min F' := y'_{N,n_y+1} + L_N(y_N) & \\ y'_{i+1} = T'_i(y'_i, x_i), \quad i = 1, \ldots, N-1 & \qquad \text{(PP)} \\ y'_1 = \bar{y}'_1 & \end{array}$$

where

$$\begin{aligned} y'_i &= [y_i^T, y'_{i,n_y+1}]^T \in R^{n_y+1}, \ i = 1, \ldots, N, \\ T'_i(y'_i, x_i) &= [T_i(y_i, x_i)^T, y'_{i,n_y+1} + L_i(y_i, x_i)]^T, \ i = 1, \ldots, N-1 \end{aligned}$$

and $\bar{y}'_1 = [\bar{y}_1^T, 0]^T$.

In the following we denote $n_{y'} := n_y + 1$, $(f)_x$ the gradient of f with respect to x and $\frac{\partial f}{\partial x}$ the Jacobian of f with respect to x. Algorithm 1 is Pantoja's algorithm

[1]We denote by $p = \Omega(q)$ if there are $0 < c_1 < c_2$ such that $c_1 q < p < c_2 q$.

for calculating the Newton direction $d = -H^{-1}g$, where H is the Hessian of $f(x)$ (we assume that H is invertible).

Algorithm 1 [Pantoja's stagewise Newton's method [16]]

Step 1. Given the current control variable x; calculate the current state variable y' via the transition function T'; calculate $P = (L_N)_{y'_N y'_N}, Q = (L_N)_{y'_N}, G = Q$.

Step 2. Perform backward recursion:

For $i = N-1, \ldots, 1$ **do**

(i). Calculate A_i, B_i, C_i, D_i and E_i according to

$$\begin{aligned}
A_i &= (\frac{\partial T'_i}{\partial y'_i})^T P (\frac{\partial T'_i}{\partial y'_i}) + \sum_{j=1}^{n_{y'}} G_j (T'_{i,j})_{y'_i y'_i} \\
B_j &= (\frac{\partial T'_i}{\partial y'_i})^T P (\frac{\partial T'_i}{\partial x_i}) + \sum_{j=1}^{n_{y'}} G_j (T'_{i,j})_{y'_i x_i} \\
C_i &= (\frac{\partial T'_i}{\partial x_i})^T P (\frac{\partial T'_i}{\partial x_i}) + \sum_{j=1}^{n_{y'}} G_j (T'_{i,j})_{x_i x_i} \\
D_i &= (\frac{\partial T'_i}{\partial x_i})^T Q \\
E_i &= (\frac{\partial T'_i}{\partial y'_i})^T Q.
\end{aligned}$$

(ii). Calculate:

$$\begin{aligned}
\alpha_i &= -C_i^{-1} D_i \\
\beta_i &= -C_i^{-1} B_i^T.
\end{aligned}$$

(iii). Update P, Q and G:

$$\begin{aligned}
P &\leftarrow A_i - \beta_i^T C_i \beta_i \\
Q &\leftarrow E_i + B_i \alpha_i \\
G &\leftarrow (\frac{\partial T'_i}{\partial y'_i})^T G.
\end{aligned}$$

End

Step 3. Calculate the Newton direction. Let $(\delta y)_1 = 0$.

For $i = 1, \ldots, N-1$ **do**

$$\begin{aligned} d_i &= \alpha_i + \beta_i (\delta y)_i \\ (\delta y)_{i+1} &= \frac{\partial T_i'}{\partial y_i'} (\delta y)_i + \frac{\partial T_i}{\partial x_i} d_i. \end{aligned}$$

End

At first glance it may be difficult to see the connection between Algorithm 1 and Newton's method. However, there is a clear relationship. As we shall see in the discussion in section 2.3, if we linearize the transition functions and expand function F up to second order derivatives, then the outcome of the DDP procedure for this approximation model is $-H_2^{-1} g$ where $H = H_1 + H_2$ is a partition of H. Pantoja's algorithm is a modification of the DDP algorithm so that the first part of H, H_1, can be included. Transforming (P) into (PP) makes a clear connection between matrices $\{C_i\}$ and the Hessian of f, H. It follows from Lemma 2.2 in section 2.3 that

$$C_{N-1} = H_{N-1,N-1}$$

where $H_{N-1,N-1}$ is the $(N-1)$-st diagonal block. If C_{N-1} is nonsingular then we can use it as the pivot element and one step of Gaussian elimination zeroes the first $N-2$ block-elements in the $(N-1)$-st block column. After this elimination, the matrix obtained by eliminating the $(N-1)$-st block column and row is just the Hessian of function f at the current point $\bar{x}$ where x_{N-1} is replaced by

$$x_{N-1} = \bar{x}_{N-1} + \sum_{j=1}^{N-2} f_{x_{N-1}, x_j} C_{N-1}^{-1} (x_j - \bar{x}_j).$$

This procedure is repeated recursively in the next step of Pantoja's algorithm. It follows from Lemma 2.2 again that C_{N-2} is the $(N-2)$nd diagonal block matrix of the matrix obtained from H by one step Gaussian elimination. This argument applies to all matrices $C_i, i = N-1, \ldots, 1$; thus we have that $H = UL$ where U is an upper triangular block matrix and L is a lower triangular block matrix with C_i being its diagonal block matrices. The proof of the equivalence of Algorithm 1 and Newton's method can be found in Pantoja [16].

Returning to problem (P), we can define a procedure for calculating Newton's step $d = -H^{-1}g$ as follows: we first transform (P) into (PP) and then use Algorithm 1 to obtain d. We denote this procedure by

$$\begin{aligned} d &= newton(\{x_i\}, \{(L_i)_{x_i}, (L_i)_{y_i}, (L_i)_{x_i x_i}, (L_i)_{x_i y_i}, (L_i)_{y_i y_i}\}, \\ &\quad \{(T_i)_{x_i}, (T_i)_{y_i}, (T_i)_{x_i x_i}, (T_i)_{x_i y_i}, (T_i)_{y_i y_i}\}) \end{aligned}$$

or simply $d = newton(x, L, T)$.

If we ignore the lower order terms the total number of operations per iteration of Pantoja's algorithm is the same as the DDP algorithm [7]:

$$N \cdot (2n_y^3 + \frac{7}{2}n_y^2 n_x + 2n_y n_x^2 + \frac{1}{3}n_x^3). \tag{1}$$

2.2 A trust region method

The trust region method of Nocedal and Yuan [15] is actually an algorithm that employs both trust region techniques and line searches. Consider an unconstrained minimization problem of the form

$$\min_{x \in R^n} f(x).$$

A trust region method calculates a trial step by solving the subproblem

$$\begin{array}{l} \min \phi_k(d) := (g^k)^T d + \frac{1}{2} d^T H_k d \\ \text{subject to} \quad \|d\| \leq \Delta_k, \end{array} \qquad \text{(subP)}$$

where $g^k = \nabla f(x^k)$, H_k is an $n \times n$ symmetric matrix which is either the Hessian of f or some approximation to it and $\Delta_k > 0$ is a trust region radius. (Throughout this paper we assume that H is the Hessian of f and $\|\cdot\|$ is the l_2 norm.) Then, based on the ratio between the actual reduction in the function and the predicted reduction, the step d^k –the solution of (subP)– is either accepted or rejected and H_k and Δ_k are updated.

Nocedal and Yuan [15] propose the following algorithm to solve (subP) approximately:

Algorithm [Algorithm 2.6 of Nocedal and Yuan [15]]

Step 1. Given $\gamma > 1$, $\epsilon > 0$, set $\lambda := 0$. If H is positive definite go to Step 2; else find $\lambda \in [0, \|H\| + (1+\epsilon)\|g\|/\Delta]$ such that $H + \lambda I$ is positive definite, where I is the identity matrix.

Step 2. Factor $H + \lambda I = R^T R$, where R is upper triangular, and solve $R^T R d = -g$ for d.

Step 3. If $\|d\| \le \Delta$ stop; else solve $R^T q = d$ for q, and compute

$$\lambda := \lambda + \frac{\|d\|^2}{\|q\|^2} \frac{\gamma\|d\| - \Delta}{\Delta}; \tag{2}$$

go to Step 2.

The Nocedal-Yuan algorithm uses the Cholesky factorization of $H + \lambda I$; however, H is not available in the DTOC setting. To overcome this difficulty, we note that factoring $H + \lambda I = R^T R$ is not necessary: the update (2), which is based on Newton's method for equation

$$\frac{\gamma}{\Delta} - \frac{1}{\|d(\lambda)\|} = 0$$

where $d(\lambda) = -(H + \lambda I)^{-1} g$, is equivalent to

$$\lambda := \lambda + \frac{\|d\|^2}{d^T d'} \frac{\gamma\|d\| - \Delta}{\Delta}$$

where $d = -(H + \lambda I)^{-1} g$ and $d' = (H + \lambda I)^{-1} d$. We thus propose the following algorithm to solve (subP).

Algorithm 2

Initialization. Given $\gamma > 1$, $\epsilon > 0$ and a $\lambda \in [0, \|H\| + (1 + \epsilon)\|g\|/\Delta]$ such that $H + \lambda I$ is positive definite.

Until $\|d\| \le \Delta$, do

$$\begin{aligned} d &= -(H + \lambda I)^{-1} g \\ d' &= (H + \lambda I)^{-1} d \\ \lambda &\leftarrow \lambda + \frac{\|d\|^2}{d^T d'} \frac{\gamma\|d\| - \Delta}{\Delta}. \end{aligned}$$

End

The following algorithm is based on Algorithm 3.1 of Nocedal and Yuan [15] in which we use our modified algorithm to solve (subP).

Algorithm 3

Initialization. Given x^1 and $\Delta_1 > 0$, choose c_1, c_2 such that $0 < c_1 < 1$ and $0 < c_2 < 1$. Set $k = 1$.

Until convergence do

(i). Solve (subP) using Algorithm 2.

(ii). Calculate $f(x^k + d^k)$.
If $f(x^k + d^k) \geq f(x^k)$
use a simple binary search to find $0 < s_k < 1$ such that $f(x^k + s_k d^k) < f(x^k)$ and put $x^{k+1} = x^k + s_k d^k$, $\Delta_{k+1} = \|x^{k+1} - x^k\|$;
else
set $x^{k+1} = x^k + d^k$ and

$$\Delta_{k+1} = \begin{cases} \Delta_k & \text{if } \rho_k \geq c_1 \\ c_2 \Delta_k & \text{o.w.} \end{cases}$$

where

$$\rho_k = \frac{f(x^k) - f(x^{k+1})}{\phi_k(0) - \phi_k(d^k)}.$$

End if
Calculate g^{k+1}; set $k = k + 1$.

End

2.3 Combining the trust region method and the stage-wise Newton's method

To carry out the trust region method with the Newton step $d = newton(u, L, f)$ we need the following:

(a). A way to detect if $H + \lambda I$ is positive definite, where I is the identity matrix, and a procedure to find a $\lambda \in [0, \|H\| + (1 + \epsilon)\|g\|/\Delta]$ such that $H + \lambda I$ is positive definite.

(b). g: the gradient of $f(x)$.

(c). $d = -(H+\lambda I)^{-1}g$ and $d' = (H+\lambda I)^{-1}d$.

(d). $\phi(d) = g^T d + \frac{1}{2} d^T H d$.

We show below that these quantities can be calculated in $O(N)$ operations.

We first consider (b). It turns out the g can be obtained efficiently by means of a "backward" sequence of equations. The following procedure is essentially that of Dyer and Mcreynolds [4]. We assume that g is in the matrix form, i.e., $g = [g_1, \ldots, g_{N-1}]$, $g_i \in R^{n_x}, i = 1, \ldots, N-1$.

Note that we can extent the definition of f to all $i = 1, \ldots, N$:

$$f_i := f_i(y_i, [x_i, \ldots, x_{N-1}]), i = 1, \ldots, N-1$$

and $f_N := L_N(y_N)$. (Thus $f(x) = f_1(y_1, x)$.) Then we have

$$f_i = f_{i+1} + L_i(y_i, x_i). \tag{3}$$

Taking $\frac{\partial}{\partial x_i}$ on both sides of (3) yields

$$\frac{\partial f_i}{\partial x_i} = \frac{\partial f_{i+1}}{\partial y_{i+1}} \frac{\partial T_i}{\partial x_i} + \frac{\partial L_i(y_i, x_i)}{\partial x_i}. \tag{4}$$

It is easy to check that $\frac{\partial f_1}{\partial x_i} = \frac{\partial f_i}{\partial x_i}$. Taking $\frac{\partial}{\partial y_i}$ on both sides of (3) gives

$$\frac{\partial f_i}{\partial y_i} = \frac{\partial f_{i+1}}{\partial y_{i+1}} \frac{\partial T_i}{\partial y_i} + \frac{\partial L_i(y_i, x_i)}{\partial y_i}. \tag{5}$$

Also we have

$$\frac{\partial f_N}{\partial y_N} = \frac{\partial L_N(y_N)}{\partial y_N}. \tag{6}$$

Combining (3)-(6) we have the following algorithm for calculating the gradient g.

Algorithm g

Initialization. Given $x = [x_1, \ldots, x_{N-1}]$, calculate $y = [y_1, \ldots, y_N]$ and $\frac{\partial f_N}{\partial y_N} = \frac{\partial L_N(y_N)}{\partial y_N}$.

For $i = N-1, \ldots, 1$ **do**

$$\begin{aligned} \frac{\partial f_i}{\partial y_i} &= \frac{\partial f_{i+1}}{\partial y_{i+1}} \frac{\partial T_i}{\partial y_k} + \frac{\partial L_i(y_i, x_i)}{\partial y_i} \\ \frac{\partial f_1}{\partial x_i} &= \frac{\partial f_{i+1}}{\partial y_{i+1}} \frac{\partial T_i}{\partial x_i} + \frac{\partial L_i(y_i, x_i)}{\partial x_i} \\ g_i &= (\frac{\partial f_1}{\partial x_i})^T. \end{aligned}$$

End

We note that the total number of operations for Algorithm g is

$$N \cdot (n_y^2 + n_y n_x). \tag{7}$$

We now consider (a) and (c). The key to our approach is the following observation.

Lemma 2.1 *H is positive definite if and only if C_i is positive definite for all $i = 1, \ldots, N-1$.*

Proof. According to Theorem 4.6 of Pantoja [16], if C_i is positive definite for all $i = 1, \ldots, N-1$ then H is positive definite. On the other hand, if H is positive definite then it follows from Theorem 4.5 of Pantoja [16] that C_i is invertible for all i. Thus the Gauss block-triangularization procedure in Pantoja [16] can be carried out. We thus have $H = UL$ where

$$U = \begin{bmatrix} M_{1,1} & M_{1,2} & \cdots & M_{1,N-1} \\ 0 & M_{2,2} & \cdots & M_{2,N-1} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & M_{N-1,N-1} \end{bmatrix}$$

and

$$L = \begin{bmatrix} C_1 & 0 & \cdots & 0 \\ L_{2,1} & C_2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ L_{N-1,1} & L_{N-1,2} & \cdots & C_{N-1} \end{bmatrix}$$

where $M_{j,j} = I, j = 1, \ldots, N-1$. It is easy to check that $H = U\bar{C}U^T$ with $\bar{C} = \text{diag}(C_1, \ldots, C_{N-1})$. Therefore C_i is positive definite for all i. $\square$

To our knowledge, the observation that if H is positive definite then C_i is positive definite (for all i) is a new one.

Lemma 2.1 provides us with a method to detect if H is positive definite: we just need to check if each C_i is positive definite. This can be done by attempting the Cholesky factorization of $C_i, i = 1, \ldots, N-1$.

Now consider how to calculate $(H + \lambda I)^{-1} h$ where h is an arbitrary vector and H is the Hessian of f at $\bar{x}$. Consider augmenting F:

$$\begin{aligned} F' &= F + \sum_{i=1}^{N-1} [\frac{\lambda}{2} \|x_i - \bar{x}_i\|^2 - (g+h)^T (x_i - \bar{x}_i)] + L_N(y_N) \\ &= \sum_{i=1}^{N-1} [L_i(y_i, x_i) + \frac{\lambda}{2} \|x_i - \bar{x}_i\|^2 - (g+h)^T (x_i - \bar{x}_i)] + L_N(y_N) \\ &=: \sum_{i=1}^{N-1} L'_i(y_i, x_i) + L'_N(y_N). \end{aligned}$$

In the following we show that if d' is the output of procedure *newton* for the problem with F' as the objective function, i.e., $d' = newton(x, L', T)$, then $d' = -H'^{-1} g' = (H + \lambda I)^{-1} h$. We need a technical result first.

Lemma 2.2 *The gradient and Hessian, g and H respectively, can be written,*

$$\begin{aligned} g &= [I, (\frac{\partial y}{\partial x})^T] \cdot \nabla F, \\ H &= [I, (\frac{\partial y}{\partial x})^T] \nabla^2 F \left[\begin{array}{c} I \\ \frac{\partial y}{\partial x} \end{array} \right] + \sum_{i=1}^{N-1} \sum_{j=1}^{n_y} [(\nabla F)_{y_{i,j}} \cdot \nabla^2_{xx} T_{i,j}], \end{aligned}$$

where I is the n by n identity matrix, $\frac{\partial y}{\partial x}$ is the Jacobian of y with respected to x and is an $n_y N$ by n matrix and $(\nabla F)_{y_{i,j}}$ is the component of the gradient of F corresponding to $y_{i,j}$. The block rows of $\frac{\partial y}{\partial x}$ can be calculated via the recursive formula:

$$\frac{\partial y_{i+1}}{\partial x} = \frac{\partial T_i}{\partial x} + \frac{\partial T_i}{\partial y_i} \frac{\partial y_i}{\partial x}$$

where

$$\frac{\partial T_i}{\partial x} = [0, \ldots, 0, \frac{\partial T_i}{\partial x_i}, 0, \ldots, 0],$$

with the boundary condition

$$\frac{\partial y_1}{\partial x} = 0 \in R^{n_y \times n_x (N-1)}.$$

Proof. By direct calculation. $\square$

Lemma 2.3 *Let $f' = f'(x)$ be the objective function reduced from F', H' be the Hessian of f' evaluated at $\bar{x}$ and g' be the gradient of f' evaluated at $\bar{x}$. Then, $H' = H + \lambda I$ and $g' = -h$.*

Proof. It follows from Lemma 2.2 that

$$g' = [I, (\frac{\partial y}{\partial x})^T] \cdot \nabla F' = [I, (\frac{\partial y}{\partial x})^T] \cdot \nabla F + \lambda(x - \bar{x}) - (g + h).$$

Thus

$$g' = g'|_{x=\bar{x}} = [I, (\frac{\partial y}{\partial x})^T] \cdot \nabla F - g - h = -h.$$

On the other hand, by Lemma 2.2,

$$\begin{aligned}
H' &= [I, (\frac{\partial y}{\partial x})^T] \nabla^2 F' \begin{bmatrix} I \\ \frac{\partial y}{\partial x} \end{bmatrix} + \sum_{i=1}^{N-1} \sum_{j=1}^{n_y} [(\nabla F')_{y_{i,j}} \cdot \nabla^2_{xx} T_{i,j}] \\
&= [I, (\frac{\partial y}{\partial x})^T] \left(\nabla^2 F + \begin{bmatrix} \lambda I & 0 \\ 0 & 0 \end{bmatrix} \right) \begin{bmatrix} I \\ \frac{\partial y}{\partial x} \end{bmatrix} + \sum_{i=1}^{N-1} \sum_{j=1}^{n_y} [(\nabla F)_{y_{i,j}} \cdot \nabla^2_{xx} T_{i,j}] \\
&= [I, (\frac{\partial y}{\partial x})^T] \nabla^2 F \begin{bmatrix} I \\ \frac{\partial y}{\partial x} \end{bmatrix} + \lambda I + \sum_{i=1}^{N-1} \sum_{j=1}^{n_y} [(\nabla F)_{y_{i,j}} \cdot \nabla^2_{xx} T_{i,j}] \\
&= H + \lambda I.
\end{aligned}$$

$\square$

Therefore $(H + \lambda I)^{-1} h$ can be calculated by applying the procedure *newton* to (P) with F being replaced by F'. We note that the total number of operations needed for this calculation is the same as (1):

$$N \cdot (2n_y^3 + \frac{7}{2} n_y^2 n_x + 2 n_y n_x^2 + \frac{1}{3} n_x^3). \tag{8}$$

It is clear that this method can be used to calculate the vectors $-(H + \lambda I)^{-1} g$ and $(H + \lambda I)^{-1} d$, both of which are needed in (c).

Now we consider (d). Note that $H = H_1 + H_2$ with

$$\begin{aligned}
H_1 &= \sum_{i=1}^{N-1} \sum_{j=1}^{n_y} [(\nabla F)_{y_{i,j}} \cdot \nabla^2_{xx} T_{i,j}] \\
H_2 &= [I, (\frac{\partial y}{\partial x})^T] \nabla^2 F \begin{bmatrix} I \\ \frac{\partial y}{\partial x} \end{bmatrix}.
\end{aligned}$$

Consider the following DTOP problem:

$$
\begin{array}{ll}
\min F_1(z,d) := \sum_{i=1}^{N-1}[(L_i)_{y_i}^T z_i + (L_i)_{x_i}^T d_i] + (L_N)_{y_N}^T z_N & \\
z_{i+1} = w_i(z_i,d_i) := \frac{\partial T_i}{\partial y_i} z_i + \frac{\partial T_i}{\partial x_i} d_i + \frac{1}{2} d_i^T (T_i)_{x_i x_i} d_i, \quad i = 1,\ldots,N-1 & \qquad (P_1) \\
z_1 = 0, &
\end{array}
$$

where $d_i^T(T_i)_{x_ix_i}d_i$ is an n_y-vector whose j-th component is $d_i^T(T_{i,j})_{x_ix_i}d_i$. Let $f_1(d)$ be the reduced function. Then obviously, $f_1(d)$ is a quadratic function. Using Lemma 2.2 simple calculations lead to:

$$
\begin{aligned}
f_1(0) &= 0 \\
\nabla f_1(0) &= [I, (\frac{\partial z}{\partial d})^T] \cdot \nabla F_1 = [I, (\frac{\partial y}{\partial x})^T] \cdot \nabla F = g \\
\nabla^2 f_1(0) &= [I, (\frac{\partial z}{\partial d})^T] \nabla^2 F_1 \begin{bmatrix} I \\ \frac{\partial z}{\partial d} \end{bmatrix} + \sum_{i=1}^{N-1} \sum_{j=1}^{n_y} [(\nabla F_1)_{z_{i,j}} \cdot \nabla_{dd}^2 w_{i,j}] \\
&= \sum_{i=1}^{N-1} \sum_{j=1}^{n_y} [(\nabla F)_{y_{i,j}} \cdot \nabla_{xx}^2 T_{i,j}] = H_1
\end{aligned}
$$

Therefore

$$
f_1(d) = g^T d + \frac{1}{2} d^T H_1 d
$$

which can be calculated via (P_1) using (ignoring the lower order terms)

$$
N \cdot (n_y n_x^2)
$$

operations.

We now consider another DTOP problem:

$$
\begin{array}{ll}
\min F_2(z,d) := \frac{1}{2}(\sum_{i=1}^{N-1}[z_i^T(L_i)_{y_iy_i}z_i + 2z_i^T(L_i)_{y_ix_i}d_i + d_i^T(L_i)_{x_ix_i}d_i]) & \\
\quad + \frac{1}{2} z_N^T (L_N)_{y_N y_N} z_N & \\
z_{i+1} = w_i(z_i,d_i) := \frac{\partial T_i}{\partial y_i} z_i + \frac{\partial T_i}{\partial x_i} d_i, \quad i = 1,\ldots,N-1 & \qquad (P_2) \\
z_1 = 0. &
\end{array}
$$

Let $f_2(d)$ be the reduced function which is also a quadratic function. Using Lemma 2.2 we have

$$
\begin{aligned}
f_2(0) &= 0 \\
\nabla f_2(0) &= 0
\end{aligned}
$$

$$
\begin{aligned}
\nabla^2 f_2(0) &= [I, (\frac{\partial z}{\partial d})^T]\nabla^2 F_2 \left[\begin{array}{c} I \\ \frac{\partial z}{\partial d} \end{array} \right] \\
&= [I, (\frac{\partial y}{\partial x})^T]\nabla^2 F \left[\begin{array}{c} I \\ \frac{\partial y}{\partial x} \end{array} \right] = H_2.
\end{aligned}
$$

Hence,

$$f_2(d) = \frac{1}{2} d^T H_2 d$$

which can be calculated via (P_2) using

$$N \cdot (n_y^2 + n_y n_x + n_x^2)$$

operations. Putting them together, the value of $\phi(d) = f_1(d) + f_2(d)$ can be calculated in two forward recursive sweeps and the total number of operations is (ignoring the lower order terms):

$$N \cdot (n_y n_x^2). \tag{9}$$

It then follows from (1), (8) and (9) – ignore the operations for calculating the gradient (7) which is regarded as a lower order term – that the total number of operations per simple-iteration, i.e., no shifting or line search is needed, is

$$N \cdot (6n_y^3 + \frac{21}{2} n_y^2 n_x + 7 n_y n_x^2 + n_x^3).$$

Finally, we propose a procedure to find a $\lambda \in (0, \|H\| + (1+\epsilon)\|g\|/\Delta)$ such that $H + \lambda I$ is positive definite.

Algorithm λ

Initialization. Take $\lambda = \|g\|/\Delta$ and let $\delta\lambda = (\|g\|/\Delta)/2$.

Until $H + \lambda I$ is positive definite do

$$\lambda \leftarrow \lambda + \delta\lambda$$

End

Note: checking if $H + \lambda I$ is positive definite can be carried out by checking if all $C_i, i = 1, \ldots, N-1$ are positive definite during the procedure *newton*.

For this algorithm we have

Lemma 2.4 *If H is not positive definite then Algorithm λ terminates in at most $\lceil (2\Delta\|H\|)/\|g\|\rceil + 1$ steps.*

Proof. If H is not positive definite, then it follows from Lemma 2.3 of Nocedal and Yuan [15] that the solution of (subP) is on the boundary, i.e., if d^* is a solution of (subP) then $\|d^*\| = \Delta$, and there is a λ^* such that $H + \lambda^* I$ is a positive semi-definite and $0 \leq \lambda^* \leq \|H\| + \|g\|/\Delta$. Therefore for any given $\epsilon > 0$ $H + \lambda I$ is positive definite for any $\lambda \in (\|H\| + \|g\|/\Delta, \|H\| + (1+\epsilon)\|g\|/\Delta)$. Hence, Algorithm λ terminates in at most $\lceil (2\Delta\|H\|)/\|g\|\rceil + 1$ steps. □

Our algorithm is the trust region method of Nocedal and Yuan [15] tailored to the DTOC setting by adapting the efficient stagewise Newton idea due to Pantoja [16]. Clearly, the convergence properties of the Nocedal-Yuan method carry over here.

Theorem 2.5 *Assume that $f : R^n \to R$ is twice continuously differentiable on the level set $\Omega = \{x : f(x) \leq f(x^0)\}$ and the sequence $\{x^k\}$ be generated by our algorithm. If Ω is a compact set and $\nabla^2 f$ is bounded on Ω, then:*

1. *The sequence x^k satisfies*

$$\lim_{k\to\infty} \inf \|g^k\| = 0.$$

2. *If f is convex it follows that*

$$\lim_{k\to\infty} \|g^k\| = 0.$$

3. *If x^k converges to a point x^* then $\nabla^2 f(x^*)$ is positive semi-definite.*

4. *If x^k converges to a point x^* such that $\nabla^2 f(x^*)$ is positive definite, the rate of convergence is quadratic.*

Proof. See Nocedal and Yuan [15]. □

We note that it follows from Theorem 4.14 of Moré [11] that if, in addition to the assumptions of Theorem 2.5, f is bounded below on Ω and ∇f is uniformly continuous then we have

$$\lim_{k\to\infty} \|g^k\| = 0.$$

Table 1: Numerical results for problem 1 ($n_y = 4, n_x = 2$)

μ	N	Iter	Feva	F^*	$\|g\|$	θ
0	10	10	10	7.3594539E-02	5.3041E-11	3.3180E-10
	50	9	9	2.2994188E-01	5.1724E-07	8.2697E-08
1/200	10	10	10	7.5313313E-02	6.1530E-10	9.8368E-11
	50	9	9	2.3950010E-01	7.9095E-07	1.2655E-07
1/20	10	8	9	9.8483463E-02	3.0286E-07	7.6949E-08
	50	9	9	3.8293257E-01	5.0803E-08	7.8325E-09
1/2	10	7	7	3.1230671E-01	4.4406E-10	8.1275E-11
	50	7	7	1.7218828E+00	4.4103E-09	8.3894E-10
1	10	6	6	4.1425647E-01	2.1047E-07	6.2088E-08
	50	6	6	2.3208717E+00	1.9211E-07	5.6377E-08

3 Numerical results

We tested our algorithm with the problems collected in the Appendix. Our algorithm was written in MATLAB and all runs were performed on a SUN Sparcstation. We take $\gamma = 1.05$ and $\epsilon > 0.5$ in Algorithm 2 and $c_1 = 0.1$ and $c_2 = 0.5$ in Algorithm 3. The line search algorithm in Algorithm 3 is carried out so that

$$f(x^{k+1}) \leq f(x^k) + 0.0001(x^{k+1} - x^k)^T g^k.$$

The convergence criterion is $\|g\| < 10^{-6}$. We also give the corresponding value of θ which is often used as the convergence criterion for DDP algorithms.

For problem 1 we take $n_y = 4$, $n_x = 2$ and $\mu = 0, 1/200, 1/20, 1/2, 1$. The larger μ, the more nonlinear the transition functions. For each μ we choose $N = 10$ and $N = 50$. The initial point is $x^1 = 0$. The numerical results are presented in Table 1.

The numerical results for problem 2 are presented in Table 2. This problem is very nonlinear. The initial point is $x^1 = 0$.

Numerical results for problems 3–5 are presented in Table 3. We take $s = 1/N$ in problem 3. The initial point for problem 5 is $x^1 = e$ where e is the all-one vector.

Problem 6 is the DTOC form of the so-called "sum of exponentials" problem [13]. We solve it using both our algorithm and the conventional method–using

Table 2: Numerical results for problem 2 ($n_y = 4, n_x = 2$)

N	Iter	Feva	F^*	$\|g\|$	θ
10	13	13	4.8598254E-01	5.3958E-08	1.5343E-08
20	12	12	4.8621209E-01	1.1622E-07	5.8311E-11
30	15	16	4.8644162E-01	3.1724E-07	1.2261E-10
40	15	16	4.8667115E-01	8.1564E-08	1.3184E-11
50	15	16	4.8690068E-01	2.0792E-08	2.0315E-11

Table 3: Numerical results for problem 3–5

problem	N	Iter	Feva	F^*	$\|g\|$	θ
Prob. 3	10	17	27	2.2459038E+02	3.5404E-07	7.1388E-10
	100	19	29	2.3428772E+02	4.1082E-08	2.7744E-10
	500	25	36	2.3508445E+02	5.6098E-07	7.7792E-10
	1000	27	33	2.3518341E+02	4.5233E-07	4.2776E-09
Prob. 4	10	14	14	3.7508235E+00	8.6445E-11	1.2410E-10
	100	12	12	2.9473466E+00	2.3395E-07	1.4338E-06
	500	25	25	2.8828510E+00	1.5253E-09	2.7640E-08
	1000	35	35	2.8748904E+00	1.1013E-11	1.8059E-10
Prob. 5	10	4	4	1.4519006E+00	8.9753E-08	7.0031E-07
	100	9	9	1.5325863E+00	4.0516E-10	3.8686E-09
	500	17	17	1.5347290E+00	3.5983E-11	7.3166E-10
	1000	23	23	1.5349460E+00	1.2683E-11	3.6280E-10

Table 4: Numerical results for problem 6

n $(N-1)$	Iter (Feva)	F^*	$\|g\|$	t_N	t_P	$t_N/iter$	$t_P/iter$
10	9	1.9804145E+01	7.9E-09	1.4	5.1	0.16	0.57
20	13	6.2495269E+01	5.0E-07	7.7	19.2	0.60	1.48
30	17	1.1903301E+02	1.7E-07	23.9	40.6	1.41	2.39
40	21	1.8589622E+02	4.3E-10	60.0	70.1	2.86	3.34
50	24	2.6111329E+02	1.7E-07	124.2	101.4	5.18	4.23
70	30	4.3184514E+02	7.1E-07	434.1	186.4	14.47	6.21
90	36	6.2461932E+02	6.2E-08	1,225.2	301.9	34.03	8.39
100	39	7.2798132E+02	6.6E-10	1,892.6	361.2	48.53	9.26

the Hessian **explicitly** in the trust region algorithm. Results are presented in Table 4 where t_N (t_P) is the cpu time (in seconds) for the conventional Newton's method (our method). It is obviously from the last 2 columns that the cpu time per iteration for our algorithm is a linear function of N while it is a cubic function of N for the conventional method. Hence if a nonlinear optimization problem with a dense Hessian matrix can be transformed into the DTOC form it can yield an enormous reduction in computational cost.

4 Concluding remarks

We have proposed a trust region method for the unconstrained DTOC problem. Our method possesses advantages of both the trust region method and the stagewise Newton's method. It has strong convergence properties yet remains economical.

There are other trust region algorithms, such as those proposed in Moré and Sorensen [12] and Gay [5], which have stronger convergence properties. In the Nocedal-Yuan algorithm the subproblem (subP) is not solved very accurately: it only guarantees that

$$\phi_k(d^k) \leq \beta \min\{\phi_k(d) : d = \mu g^k, \|d\| \leq \Delta_k\}, \quad \|d^k\| \leq \Delta_k$$

where β is some positive constant. In the Moré-Sorensen and Gay's algorithms the subproblem (subP) is solved more accurately; i.e.,

$$\phi_k(d^k) \leq \beta_1 \min\{\phi_k(d) : \|d\| \leq \Delta_k\}, \quad \|d^k\| \leq \beta_2 \Delta_k,$$

for some positive constants β_1 and β_2. Therefore, the Moré-Sorensen and Gay's algorithms possess stronger convergence properties [11]: if $\{x^k\}$ is bounded then there is a limit point x^* which satisfies the first and second order necessary conditions; if $\nabla^2 f(x^*)$ is nonsingular for some limit point x^* then $\nabla^2 f(x^*)$ is positive definite and $\{x^k\}$ converges to x^*. However both the Moré-Sorensen algorithm and Gay's algorithm assume that the explicit Hessian is available which is not the case for most discrete-time optimal control problems.

Another algorithm, in the trust region camp, is the dogleg algorithm of Powell [18]. The convergence results for the dogleg algorithm do not guarantee both the first and second order necessary conditions in the case that $\{x^k\}$ converges to x^*; therefore, they are weaker than those for the Nocedal-Yuan algorithm. We also note that the mechanisms for the dogleg algorithm and the Nocedal-Yuan algorithm are different. For the dogleg algorithm the search direction is the improved steepest descent direction with the help of Newton's direction and is not expected to be the solution of the subproblem (subP); while for the Nocedal-Yuan algorithm the search direction is based on Newton's method for equation

$$\frac{\gamma}{\Delta} - \frac{1}{\|d(\lambda)\|} = 0$$

where $\gamma > 1$ is some constant and $d(\lambda) = -(H + \lambda I)^{-1} g$. Therefore, if γ is near 1, it is the solution to the subproblem (subP) except for the "hard case". To compare the dogleg algorithm with the Nocedal-Yuan algorithm numerically we solve our test problems using the dogleg algorithm. Our numerical experiments show that the Nocedal-Yuan algorithm performs better. We present in Table 5 the results for problems 3–5 using the dogleg algorithm to highlight the behavior of this algorithm for the DTOC problems.

Another popular method for solving DTOC problems is the DDP method. Both the DDP and the stagewise Newton methods are locally quadratically convergent. We note that the stagewise Newton method is independent of the transition functions as long as they give the same function f, while the DDP method depends on the transition functions. Sometimes the DDP performs better for $(\bar{P})$ by solving the corresponding DTOC problem (P) if the transition functions are chosen properly, but it seems that there are no general rules to follow. It would be of interest to generalize our approach to the DDP method.

Table 5: Numerical results for problem 3–5 using the dogleg algorithm

problem	N	Iter	Feva	F^*	$\|g\|$
Prob. 3	10	101	101	2.2459038E+02	8.9568E-07
	100	124	124	2.3428772E+02	8.8156E-07
	500	90	90	2.3508445E+02	7.8271E-07
	1000	96	96	2.3518341E+02	9.7264E-07
Prob. 4	10	18	18	3.7508235E+00	2.5192E-07
	100	12	12	2.9473466E+00	1.6331E-08
	500	20	20	2.8828510E+00	3.7078E-07
	1000	27	27	2.8748904E+00	2.7201E-07
Prob. 5	10	16	16	1.4519006E+00	6.7223E-07
	100	21	21	1.5325863E+00	7.1019E-07
	500	33	33	1.5347290E+00	9.9071E-07
	1000	34	34	1.5349460E+00	9.1366E-07

We also would like to point out that although the DDP method requires less computation than the stagewise Newton method during each iteration, Pantoja [16] provides an example showing that the stagewise Newton method gives the exact solution in one iteration while DDP only provides an iterative solution.

Finally, while our work makes use of the dynamic structure in series, Ralph [19] and Wright [20] propose some parallel algorithms for DTOC problems that explore the dynamic structure in parallel.

Acknowledgment: We thank Christine Shoemaker and Li-Zhi Liao for many illuminating discussions on discrete-time optimal control problems.

Appendix

We collect some test problems in this section.

Problem 1. [8]

$$\begin{array}{l} \min F := \sum_{i=1}^{N-1}(\sum_{j=1}^{n_y}(y_{i,j}+\frac{1}{4})^4 + \sum_{j=1}^{n_x}(x_{i,j}+\frac{1}{2})^4) + \sum_{j=1}^{n_y}(y_{N,j}+\frac{1}{4})^4 \\ y_{i+1} = T_i(y_i, x_i), \quad i = 1, \dots, N-1 \end{array}$$

where

$$T_i(y_i, x_i) = Ay_i + Bx_i + (y_i^T C x_i)e, i = 1, \dots, N-1,$$

$y_1 = 0$, and $A \in R^{n_y \times n_y}$ given by:

$$(A)_{i,j} = \begin{cases} 0.5 & \text{if } i = j \\ 0.25 & \text{if } j = i+1 \\ -0.25 & \text{if } j = i-1; \end{cases}$$

$B \in R^{n_y \times n_x}$ given by:

$$(B)_{i,j} = \frac{i+j}{n_y + n_x};$$

$C \in R^{n_y \times n_x}$ given by:

$$(C)_{i,j} = \mu \frac{i+j}{n_y + n_x};$$

e is the all-one vector in R^{n_y}.

Problem 2. [8]

$$\begin{array}{l} \min F := \sum_{i=1}^{N-1}(\|y_i\|^2 \cdot [\sin^2(\frac{\|x_i\|^2}{n_x}) + 1]) + \|y_N\|^2 \\ y_{i+1} = T_i(y_i, x_i), \quad i = 1, \dots, N-1 \end{array}$$

where

$$T_i(y_i, x_i) = \sin(y_i) + C \cdot W(x_i)$$

and $C \in R^{n_y \times n_x}$ given by

$$C_{i,j} = \frac{i+j}{2n_y}, \ i = 1, \dots, n; \ j = 1, \dots, m.$$

$W(x_i) = (\sin(x_{i,1}), \dots, \sin(x_{i,n_x}))^T$. The initial state is given by

$$y_{1,j} = \frac{j}{2n_y}, \quad j = 1, \dots, n_y.$$

The initial point is $x^1 = 0$.

Note: this problem is the modification of that in [21] where the the single loss function is given by

$$L(y,x) = \exp(\|y\|^2)\left[\sin^2(\frac{\|x\|^2}{n_x}) + 1\right].$$

Problem 3. [1] ($n_y = 2$ $n_x = 1$)

$$\begin{aligned} &\min F := \tfrac{s}{2}\textstyle\sum_{i=1}^{N-1}(y_{i+1}^T Q y_{i+1} + Rx_i^2) \\ &y_{i+1} = \begin{bmatrix} 1 & s \\ -s & 1 \end{bmatrix} y_i + \begin{bmatrix} 0 \\ s \end{bmatrix} x_i, i = 1,\ldots,N-1. \end{aligned}$$

where $R = 6$ and

$$Q = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}.$$

$s \in (0,1)$ and $y_1 = (15,5)^T$. The initial point is $x^1 = 0$.

Problem 4. [17] ($n_y = 2$ $n_x = 1$)

$$\begin{aligned} &\min F := (\tfrac{5h}{2}\|y_1\|^2 + 5hx_1^2) + 5h\textstyle\sum_{i=2}^{N-1}(\|y_i\|^2 + x_i^2) + \tfrac{5h}{2}\|y_N\|^2 \\ &y_{i+1} = y_i + 5h\begin{bmatrix} (1-(y_{i,2})^2)(y_{i,1}) - (y_{i,2}) + x_i \\ (y_{i,1}) \end{bmatrix}, \quad i = 1,\ldots,N-1 \end{aligned}$$

where $h = 1/N$ and $y_1 = (0,1)^T$. The initial point is $x^1 = 0$.

Problem 5. [17] ($n_y = 1$ $n_x = 1$)

$$\begin{aligned} &\min F := h\textstyle\sum_{i=1}^{N-1}(y_i^2 + x_i^2) \\ &y_{i+1} = y_i + h(y_i^2 - x_i), \; i = 1,\ldots,N-1 \end{aligned}$$

where $h = 1/N$ and $y_1 = 1$. The initial point is $x^1 = e$.

Problem 6. [13] ($n_y = 1$ $n_x = 1$)

$$\begin{aligned} &\min F := \textstyle\sum_{i=1}^{N-1}[\frac{1}{2}(y_i + \exp(x_i))^2 + \frac{1}{2}(x_i)^2] \\ &y_{i+1} = y_i + \exp(x_i), \; i = 1,\ldots,N-1 \end{aligned}$$

where $y_1 = 0$. The initial point is $x_1 = 0$.

Note: this is the DTOC formulation of the so-called "Sum of exponentials:"

$$f(x) = \sum_{i=1}^{n} [\frac{1}{2}(x_i)^2 + \frac{1}{2}(\sum_{j=1}^{i} \exp(x_j))^2]$$

with $n = N - 1$. There are some other nonlinear optimization problems in [13] that can be reduced to the DTOC form.

References

[1] D. P. Bertsekas. Projected Newton methods for optimization problems with simple constraints. *SIAM J. Control and Optimization*, 20:221–246, 1982.

[2] B. D. Craven. *Mathematical Programming and Control Theory*. Chapman and Hall, London, 1978.

[3] T. B. Culver and C. A. Shoemaker. Dynamic optimal control for groundwater remediation with flexible management periods. *Water Resources Research*, 28:629–641, 1992.

[4] P. Dyer and S. R. Mcreynolds. *The computation and theory of optimal control*. Academic Press, 1970.

[5] D. M. Gay. Computing optimal local constrained steps. *SIAM J. Sci. Stat. Comput.*, 2:186–197, 1981.

[6] D. Jacobson and D. Mayne. *Differential dynamic programming*. Elsevier Sci. Publ., 1970.

[7] L.-Z. Liao and C. A. Shoemaker. Convergence in unconstrained discrete-time differential dynamic programming. *IEEE Trans. Automat. Contr.*, 36:692–706, 1991.

[8] L.-Z. Liao and C. A. Shoemaker. Advantages of differential dynamic programming over Newton's method for discrete-time optimal control problems. Technical Report ctc92tr97, Advanced Computing Research Institute, Cornell University, 1992.

[9] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, Reading, Mass., 1984.

[10] D. Mayne. A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems. *Intnl. J. Control*, 3:85–95, 1966.

[11] J. J. Moré. Recent developments in algorithms and software for trust region methods. In A. Bachem, M. Grötschel, and B. Korte, editors, *Mathematical Programming*, pages 258–287, New York, 1983. Springer-Verlag.

[12] J. J. Moré and D. C. Sorensen. Computing a trust region step. *SIAM J. Sci. Stat. Comput.*, 4:553–572, 1983.

[13] D. M. Murray and S. J. Yakowitz. The application of optimal control methodology to nonlinear programming problems. *Mathematical Programming*, 21:331–347, 1981.

[14] D. M. Murray and S. J. Yakowitz. Differential dynamic programming and Newton's method for discrete optimal control problems. *J. of Optimization Theory and Applications*, 43:395–414, 1984.

[15] J. Nocedal and Y. Yuan. Combining trust region and line search techniques. Technical Report, Dept of EE and Computer Science, Northwestern University, 1992.

[16] J. F. A. De O. Pantoja. Differential dynamic programming and Newton's method. *Intnl. J. Control*, 47:1539–1553, 1988.

[17] G. Di Pillo, L. Grippo, and F. Lampariello. A class of structured quasi-Newton algorithms for optimal control problems. In H. E. Rauch, editor, *IFAC Applications of nonlinear programming to optimization and control*, pages 101–107. International Federation of Automatic Control, Pergamon Press, New York, 1983.

[18] M. J. D. Powell. A new algorithm for unconstrained optimization. In J. B. Rosen, O. Mangasarian, and K. Ritter, editors, *Nonlinear Programming*, pages 31–65, New York, 1970. Academic Press.

[19] D. Ralph. A parallel method for discrete-time optimal control problems. Technical Report ctc93tr118, Advanced Computing Research Institute, Cornell University, 1993.

[20] S. J. Wright. Partitioned dynamic programming for optimal control. *SIAM Journal on optimization*, 1:620–642, 1991.

[21] S. Yakowitz and B. Rutherford. Computational aspects of discrete-time optimal control. *Applied Mathematics and Computation*, 15:29–45, 1984.