STABILIZED OPTIMIZATION VIA AN NCL ALGORITHM*

DING MA†, KENNETH JUDD‡, DOMINIQUE ORBAN§, AND MICHAEL SAUNDERS†

Abstract. For optimization problems involving many nonlinear inequality constraints, we extend the bound-constrained (BCL) and linearly-constrained (LCL) augmented-Lagrangian approaches of LANCELOT and MINOS to an algorithm that solves a sequence of about 10 nonlinearly constrained augmented Lagrangian subproblems whose nonlinear constraints satisfy the LICQ everywhere. The NCL algorithm is implemented in AMPL and tested on large instances of a tax policy model that cannot be solved directly by the state-of-the-art solvers that we tested, because of singularity in the Jacobian of the active constraints. Algorithm NCL with IPOPT as subproblem solver proves to be effective, with IPOPT using second derivatives and successfully warm-starting each subproblem.

1. Introduction. We consider constrained optimization problems of the form

$$\text{NCO} \qquad \begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & \phi(x) \\ \text{subject to} & c(x) \ge 0, \quad Ax \ge b, \quad \ell \le x \le u, \end{array}$$

where $\phi(x)$ is a smooth nonlinear function, $c(x) \in \mathbb{R}^m$ is a vector of smooth nonlinear functions, and $Ax \ge b$ is a placeholder for a set of linear inequality or equality constraints, with x lying between lower and upper bounds ℓ and u.

In some applications where $m \gg n$, there may be more than n constraints that are essentially active at a solution. The constraints do not satisfy the linear independence constraint qualification (LICQ), and general-purpose solvers are likely to have difficulty converging. Some form of regularization is required. The stabilized SQP method of Gill et al. [9, 10] has been developed specifically for such problems. We achieve reliability more simply by adapting the augmented Lagrangian algorithm of the general-purpose optimization solver LANCELOT [4, 5, 15] to derive a sequence of regularized subproblems denoted in the next section by NC_k.

2. BCL, LCL, and NCL methods. The theory for the large-scale solver LANCELOT is best described in terms of the general optimization problem

$$\text{NECB} \qquad \begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & \phi(x) \\ \text{subject to} & c(x) = 0, \quad \ell \le x \le u \end{array}$$

with *nonlinear equality constraints* and bounds. We let x^* denote a local solution of NECB and (y^*, z^*) denote associated multipliers. LANCELOT treats NECB by solving a sequence of *bound-constrained subproblems* of the form

$$\text{BC}_k \qquad \begin{array}{ll} \underset{x}{\text{minimize}} & L(x, y_k, \rho_k) = \phi(x) - y_k^T c(x) + \tfrac{1}{2}\rho_k \|c(x)\|^2 \\ \text{subject to} & \ell \le x \le u, \end{array}$$

where y_k is an estimate of the Lagrange multipliers y^* for the equality constraints. This was called a bound-constrained Lagrangian (BCL) method by Friedlander and

*Version of January 23, 2018.

†Management Science and Engineering, Stanford University, Stanford, CA 94305-4026, USA ({dingma,saunders}@stanford.edu). Partially supported by NIH grant U01GM102098.

‡Hoover Institution, Stanford University, Stanford, CA 94305-6010, USA (judd@hoover.stanford.edu)

§GERAD and Dept of Mathematics and Industrial Engineering, École Polytechnique, Montréal, QC, Canada (dominique.orban@gerad.ca). Partially supported by an NSERC Discovery Grant.

Saunders [8], in contrast to the LCL (linearly constrained Lagrangian) methods of Robinson [18] and MINOS [16], whose subproblems LC_k contain bounds as in BC_k and also linearizations of the equality constraints at the current point x_k (including linear constraints).

In order to treat NCO with a sequence of BC_k subproblems, we convert the nonlinear inequality constraints to equalities to obtain

$$\boxed{\begin{array}{lll}\mathrm{NCO}' & \underset{x,\,s}{\text{minimize}} & \phi(x) \\ & \text{subject to} & c(x)-s=0, \quad Ax\ge b, \quad \ell\le x\le u, \quad s\ge 0\end{array}}$$

with corresponding subproblems (including linear constraints)

$$\boxed{\begin{array}{lll}\mathrm{BC}_k{}' & \underset{x,\,s}{\text{minimize}} & L(x,y_k,\rho_k)=\phi(x)-y_k^T(c(x)-s)+\tfrac{1}{2}\rho_k\|c(x)-s\|^2 \\ & \text{subject to} & Ax\ge b, \quad \ell\le x\le u, \quad s\ge 0.\end{array}}$$

We now introduce variables $r=-(c(x)-s)$ into $\mathrm{BC}_k{}'$ to obtain the *nonlinearly constrained Lagrangian* (NCL) subproblem

$$\boxed{\begin{array}{lll}\mathrm{NC}_k & \underset{x,\,r}{\text{minimize}} & \phi(x)+y_k^Tr+\tfrac{1}{2}\rho_k\|r\|^2 \\ & \text{subject to} & c(x)+r\ge 0, \quad Ax\ge b, \quad \ell\le x\le u,\end{array}}$$

in which r serves to make the nonlinear constraints independent. Assuming existence of finite multipliers and feasibility, for $\rho_k>0$ and larger than a certain finite value, the NCL subproblems should cause y_k to approach y^* and most of the solution $(x_k^*, r_k^*, y_k^*, z_k^*)$ of NC_k to approach (x^*, y^*, z^*), with r_k^* approaching zero.

Problem NC_k is analogous to Friedlander and Orban's formulation for convex quadratic programs [7, Eq. (3.2)]. See also Arreckx and Orban [2], where the motivation is the same as here, achieving reliability when the nonlinear constraints don't satisfy LICQ.

Note that for general problems NECB, the BCL and LCL subproblems contain linear constraints (bounds only, or linearized constraints and bounds). Our NCL formulation retains nonlinear constraints in the NC_k subproblems, but simplifies them by ensuring that they satisfy LICQ. On large problems, the additional variables $r\in\mathbb{R}^m$ in NC_k may be detrimental to active-set solvers like MINOS or SNOPT [11] because they increase the number of degrees of freedom (superbasic variables). Fortunately they are easily accommodated by interior methods, as our numerical results show for IPOPT [19, 12]. We trust that the same will be true for KNITRO [3, 14]. These solvers are most effective when second derivatives are available, as they are for our AMPL model.

2.1. The BCL algorithm. The LANCELOT BCL method is summarized in Algorithm BCL. Each subproblem BC_k is solved with a specified optimality tolerance ω_k, generating an iterate x_k^* and the associated Lagrangian gradient $z_k^*\equiv\nabla L(x_k^*, y_k, \rho_k)$. If $\|c(x_k^*)\|$ is sufficiently small, the iteration is regarded as "successful" and an update to y_k is computed from x_k^*. Otherwise, y_k is not altered but ρ_k is increased.

Key properties are that the subproblems are solved inexactly, the penalty parameter is increased only finitely often, and the multiplier estimates y_k need not be assumed bounded. Under certain conditions, all iterations are eventually successful, the ρ_k's remain constant, the iterates converge superlinearly, and the algorithm terminates in a finite number of iterations.

Algorithm 1 BCL (Bound-Constrained Lagrangian Method for NECB)

1: **procedure** BCL(x_0, y_0, z_0)
2: Set penalty parameter $\rho_1 > 0$, scale factor $\tau > 1$, and constants $\alpha, \beta > 0$ with $\alpha < 1$.
3: Set positive convergence tolerances $\eta_*, \omega_* \ll 1$ and infeasibility tolerance $\eta_1 > \eta_*$.
4: $k \leftarrow 0$, converged $\leftarrow$ false
5: **repeat**
6: $k \leftarrow k + 1$
7: Choose optimality tolerance $\omega_k > 0$ such that $\lim_{k\to\infty} \omega_k \le \omega_*$.
8: Find (x_k^*, z_k^*) that solves BC_k to within ω_k.
9: **if** $\|c(x_k^*)\| \le \max(\eta_*, \eta_k)$ **then**
10: $y_k^* \leftarrow y_k - \rho_k c(x_k^*)$
11: $x_k \leftarrow x_k^*,\ \ y_k \leftarrow y_k^*,\ \ z_k \leftarrow z_k^*$ *update solution estimates*
12: **if** (x_k, y_k, z_k) solves NECB to within ω_*, converged $\leftarrow$ true
13: $\rho_{k+1} \leftarrow \rho_k$ *keep* ρ_k
14: $\eta_{k+1} \leftarrow \eta_k/(1 + \rho_{k+1}^{\beta})$ *decrease* η_k
15: **else**
16: $\rho_{k+1} \leftarrow \tau\rho_k$ *increase* ρ_k
17: $\eta_{k+1} \leftarrow \eta_0/(1 + \rho_{k+1}^{\alpha})$ *may increase or decrease* η_k
18: **end if**
19: **until** converged
20: $x^* \leftarrow x_k,\ \ y^* \leftarrow y_k,\ \ z^* \leftarrow z_k$
21: **end procedure**

Algorithm 2 NCL (Nonlinearly Constrained Lagrangian Method for NCO)

1: **procedure** NCL(x_0, r_0, y_0, z_0)
2: Set penalty parameter $\rho_1 > 0$, scale factor $\tau > 1$, and constants $\alpha, \beta > 0$ with $\alpha < 1$.
3: Set positive convergence tolerances $\eta_*, \omega_* \ll 1$ and infeasibility tolerance $\eta_1 > \eta_*$.
4: $k \leftarrow 0$, converged $\leftarrow$ false
5: **repeat**
6: $k \leftarrow k + 1$
7: Choose optimality tolerance $\omega_k > 0$ such that $\lim_{k\to\infty} \omega_k \le \omega_*$.
8: Find $(x_k^*, r_k^*, y_k^*, z_k^*)$ that solves NC_k to within ω_k.
9: **if** $\|r_k^*\| \le \max(\eta_*, \eta_k)$ **then**
10: $y_k^* \leftarrow y_k + \rho_k r_k^*$
11: $x_k \leftarrow x_k^*,\ \ r_k \leftarrow r_k^*,\ \ y_k \leftarrow y_k^*,\ \ z_k \leftarrow z_k^*$ *update solution estimates*
12: **if** (x_k, y_k, z_k) solves NCO to within ω_*, converged $\leftarrow$ true
13: $\rho_{k+1} \leftarrow \rho_k$ *keep* ρ_k
14: $\eta_{k+1} \leftarrow \eta_k/(1 + \rho_{k+1}^{\beta})$ *decrease* η_k
15: **else**
16: $\rho_{k+1} \leftarrow \tau\rho_k$ *increase* ρ_k
17: $\eta_{k+1} \leftarrow \eta_0/(1 + \rho_{k+1}^{\alpha})$ *may increase or decrease* η_k
18: **end if**
19: **until** converged
20: $x^* \leftarrow x_k,\ \ r^* \leftarrow r_k,\ \ y^* \leftarrow y_k,\ \ z^* \leftarrow z_k$
21: **end procedure**

Note that at step 8 of Algorithm BCL, the inexact minimization would typically use the initial guess (x_k^*, z_k^*). However, other initial points are possible. At step 12, we say that (x_k, y_k, z_k) solves NECB to within ω_* if the largest dual infeasibility is smaller than ω_*.

2.2. The NCL algorithm. To derive a stabilized algorithm for problem NCO, we modify Algorithm BCL by introducing r and replacing the subproblems BC_k by NC_k. The resulting method is summarized in Algorithm NCL. The update to y_k becomes $y_k^* \leftarrow y_k - \rho_k(c(x_k^*) - s_k^*) = y_k + \rho_k r_k^*$, the value satisfied by an optimal y_k^* for subproblem NC_k. Step 8 of Algorithm NCL would typically use $(x_k^*, r_k^*, y_k^*, z_k^*)$ as initial guess, and that is what we use in our implementation below.

3. An application: optimal tax policy.

Some challenging test cases arise from the tax policy models described in [13]. With $x = (c, y)$, they take the form

$$\boxed{\begin{array}{llll} \text{TAX} & \underset{c,\,y}{\text{maximize}} & \sum_i \lambda_i U^i(c_i, y_i) & \\ & \text{subject to} & U^i(c_i, y_i) - U^i(c_j, y_j) \ge 0 & \text{for all } i, j \\ & & \lambda^T(y - c) \ge 0 & \\ & & c,\ y \ge 0, & \end{array}}$$

where c_i and y_i are the consumption and income of taxpayer i, and λ is a vector of positive weights. The utility functions $U^i(c_i, y_i)$ are each of the form

$$U(c, y) = \frac{(c - \alpha)^{1-1/\gamma}}{1 - 1/\gamma} - \psi \frac{(y/w)^{1/\eta+1}}{1/\eta + 1},$$

where w is the wage rate and α, γ, ψ and η are taxpayer heterogeneities. More precisely, the utility functions are of the form

$$U^{i,j,k,g,h}(c_{p,q,r,s,t}, y_{p,q,r,s,t}) = \frac{(c_{p,q,r,s,t} - \alpha_k)^{1-1/\gamma_h}}{1 - 1/\gamma_h} - \psi_g \frac{(y_{p,q,r,s,t}/w_i)^{1/\eta_j+1}}{1/\eta_j + 1},$$

where (i, j, k, g, h) and (p, q, r, s, t) run over na wage types, nb elasticities of labor supply, nc basic need types, nd levels of distaste for work, and ne elasticities of demand for consumption, with na, nb, nc, nd, ne determining the size of the problem, namely $m = T(T-1)$ nonlinear constraints, $n = 2T$ variables, with $T := na \times nb \times nc \times nd \times ne$.

Table 1 summarizes results for a 4D example ($ne = 1$ and $\gamma_1 = 1$). The first term of $U(c, y)$ becomes $\log(c - \alpha)$, the limit as $\gamma \to 1$. Problem NCO and Algorithm NCL were formulated in the AMPL modeling language [6]. The solvers SNOPT [11] and IPOPT [19] were unable to solve NCO itself, but Algorithm NCL was successful with IPOPT solving the subproblems NC_k. We use a default configuration of IPOPT with MUMPS [1] as symmetric indefinite solver to compute search directions. We set the optimality tolerance for IPOPT to $\omega_k = 10^{-6}$ throughout, and specified warm starts for $k \ge 2$ using options warm_start_init_point=yes and mu_init=1e-4. These options greatly improved the performance of IPOPT on each subproblem compared to cold starts, for which mu_init=0.1. It is helpful that only the objective function of NC_k changes with k.

For this example, problem NCO has $m = 39006$ nonlinear inequality constraints and one linear constraint in $n = 395$ variables $x = (c, y)$, and nonnegativity bounds. Subproblem NC_k has 39007 constraints and 39402 variables when r is included. Fortunately r does not affect the complexity of each IPOPT iteration, but greatly improves stability. In contrast, active-set methods like MINOS and SNOPT are very inefficient on the NC_k subproblems because the large number of inequality constraints leads to thousands of minor iterations, and the presence of r (with no bounds) leads to thousands of superbasic variables. About $3.2n$ constraints were within 10^{-6} of being active.

k	ρ_k	η_k	$\|r_k^*\|_\infty$	$\phi(x_k^*)$	Itns	Time
1	10^2	10^{-2}	3.1e-03	-2.1478532e+01	125	42.8
2	10^2	10^{-3}	1.3e-03	-2.1277587e+01	18	6.5
3	10^3	10^{-3}	6.6e-04	-2.1177152e+01	27	9.1
4	10^3	10^{-4}	5.5e-04	-2.1110210e+01	31	10.8
5	10^4	10^{-4}	2.9e-04	-2.1066664e+01	57	24.3
6	10^5	10^{-4}	6.5e-05	-2.1027152e+01	75	26.8
7	10^5	10^{-5}	5.2e-05	-2.1018896e+01	130	60.9
8	10^6	10^{-5}	9.3e-06	-2.1015295e+01	159	81.8
9	10^6	10^{-6}	2.0e-06	-2.1014808e+01	139	70.0
10	10^7	10^{-6}	2.1e-07	-2.1014800e+01	177	97.6

TABLE 1

NCL results on a 4D example with $na, nb, nc, nd = 11, 3, 3, 2$, giving $m = 39006$, $n = 395$. Itns refers to IPOPT's primal-dual interior point method, and Time is seconds on an Apple iMac with 2.93 GHz Intel Core i7.

k	ρ_k	η_k	$\|r_k^*\|_\infty$	$\phi(x_k^*)$	Itns	Time
1	10^2	10^{-2}	7.0e-03	-4.2038075e+02	95	41.1
2	10^2	10^{-3}	4.1e-03	-4.2002898e+02	17	7.2
3	10^3	10^{-3}	1.3e-03	-4.1986069e+02	20	8.1
4	10^4	10^{-3}	4.4e-04	-4.1972958e+02	48	25.0
5	10^4	10^{-4}	2.2e-04	-4.1968646e+02	43	20.5
6	10^5	10^{-4}	9.8e-05	-4.1967560e+02	64	32.9
7	10^5	10^{-5}	6.6e-05	-4.1967177e+02	57	26.8
8	10^6	10^{-5}	4.2e-06	-4.1967150e+02	87	46.2
9	10^6	10^{-6}	9.4e-07	-4.1967138e+02	96	53.6

TABLE 2

NCL results on a 5D example with $na, nb, nc, nd, ne = 5, 3, 3, 2, 2$, giving $m = 32220$, $n = 360$.

Table 2 summarizes results for a 5D example. The NC_k subproblems have $m = 32220$ nonlinear constraints and $n = 360$ variables, leading to 32581 variables including r. Again the options warm_start_init_point=yes and mu_init=1e-4 for $k \geq 2$ led to good performance by IPOPT on each subproblem. About $3n$ constraints were within 10^{-6} of being active.

For much larger problems of this type, we found that it was helpful to reduce mu_init more often, as illustrated in Table 3. The NC_k subproblems here have $m = 570780$ nonlinear constraints and $n = 1512$ variables, leading to 572292 variables including r. Note that the number of NCL iterations is stable ($k \leq 10$), and IPOPT performs well on each subproblem with decreasing mu_init. This time about $6.6n$ constraints were within 10^{-6} of being active.

Note that the LANCELOT approach allows early subproblems to be solved less accurately. It may save time to set $\omega_k = \eta_k$ (say) rather than $\omega_k = \omega_*$ throughout.

k	ρ_k	η_k	$\|r_k^*\|_\infty$	$\phi(x_k^*)$	mu_init	Itns	Time
1	10^2	10^{-2}	5.1e-03	-1.7656816e+03	10^{-1}	825	7763.3
2	10^2	10^{-3}	2.4e-03	-1.7648480e+03	10^{-4}	66	472.8
3	10^3	10^{-3}	1.3e-03	-1.7644006e+03	10^{-4}	106	771.3
4	10^4	10^{-3}	3.8e-04	-1.7639491e+03	10^{-5}	132	1347.0
5	10^4	10^{-4}	3.2e-04	-1.7637742e+03	10^{-5}	229	2450.9
6	10^5	10^{-4}	8.6e-05	-1.7636804e+03	10^{-6}	104	1096.9
7	10^5	10^{-5}	4.9e-05	-1.7636469e+03	10^{-6}	143	1633.4
8	10^6	10^{-5}	1.5e-05	-1.7636252e+03	10^{-7}	71	786.1
9	10^7	10^{-5}	2.8e-06	-1.7636196e+03	10^{-7}	67	725.7
10	10^7	10^{-6}	5.1e-07	-1.7636187e+03	10^{-8}	18	171.0

TABLE 3

NCL results on a 5D example with $na, nb, nc, ne, ne = 21, 3, 3, 2, 2$, giving $m = 570780$, $n = 1512$.

4. AMPL models, data, and scripts.

Algorithm NCL has been implemented in the AMPL modeling language [6] and tested on problem TAX. The following sections list each relevant file. The files are available from [17].

4.1. Tax model.

File `pTax5Dncl.mod` codes subproblem NC_k for problem TAX with five parameters w, η, α, ψ, γ, using $\mu := 1/\eta$. Note that for $U(c, y)$ in the objective and constraint functions, the first term $(c - \alpha)^{1-1/\gamma}/(1 - 1/\gamma)$ is replaced by a piecewise-smooth function that is defined for all values of c and α (see [13]).

Primal regularization $\frac{1}{2}\delta\|(c, y)\|^2$ with $\delta = 10^{-8}$ is added to the objective function to promote uniqueness of the minimizer. The vector r is called `R` to avoid a clash with subscript `r`.

```
# pTax5Dncl.mod
# An NLP to solve a taxation problem with 5-dimensional types of tax payers.
#
# 29 Mar 2005: Original AMPL coding for 2-dimensional types by K. Judd and C.-L. Su.
# 20 Sep 2016: Revised by D. Ma and M. A. Saunders.
# 08 Nov 2016: 3D version created.
# 08 Dec 2016: 4D version created.
# 10 Mar 2017: Piece-wise smooth utility function created.
# 12 Nov 2017: pTax5Dncl.mod derived from pTax5D.mod.
# 08 Dec 2017: pTax5Dncl files added to multiscale website.

# Define parameters for agents (taxpayers)
param na;                 # number of types in wage
param nb;                 # number of types in eta
param nc;                 # number of types in alpha
param nd;                 # number of types in psi
param ne;                 # number of types in gamma
set A := 1..na;           # set of wages
set B := 1..nb;           # set of eta
set C := 1..nc;           # set of alpha
set D := 1..nd;           # set of psi
set E := 1..ne;           # set of gamma
set T = {A,B,C,D,E};      # set of agents

# Define wages for agents (taxpayers)
param wmin;               # minimum wage level
param wmax;               # maximum wage level
param w {A};              # i, wage vector
param mu{B};              # j, mu = 1/eta# mu vector
param mu1{B};             # mu1[j] = mu[j] + 1
param alpha{C};           # k, ak vector for utility
param psi{D};             # g
param gamma{E};           # h
param lambda{A,B,C,D,E}; # distribution density
param epsilon;
param primreg     default 1e-8;  # Small primal regularization

var c{(i,j,k,g,h) in T} >= 0.1;  # consumption for tax payer (i,j,k,g,h)
var y{(i,j,k,g,h) in T} >= 0.1;  # income      for tax payer (i,j,k,g,h)
var R{(i,j,k,g,h) in T, (p,q,r,s,t) in T:
      !(i=p and j=q and k=r and g=s and h=t)} >= -1e+20, <= 1e+20;

param kmax      default 20;         # limit on NCL itns
param rhok      default 1e+2;       # augmented Lagrangian penalty parameter
param rhofac    default 10.0;       # increase factor
param rhomax    default 1e+8;       # biggest rhok
param etak      default 1e-2;       # opttol for augmented Lagrangian loop
param etafac    default  0.1;       # reduction factor for opttol
param etamin    default 1e-8;       # smallest etak
```

```
param rmax      default    0;       # max r (for printing)
param rmin      default    0;       # min r (for printing)
param rnorm     default    0;       # ||r||_inf
param rtol      default 1e-6;       # quit if biggest |r_i| <= rtol

param nT        default    1;       # nT = na*nb*nc*nd*ne
param m         default    1;       # nT*(nT-1) = no. of nonlinear constraints
param n         default    1;       # 2*nT      = no. of nonlinear variables

param ck{(i,j,k,g,h) in T} default 0;        # current variable c
param yk{(i,j,k,g,h) in T} default 0;        # current variable y
param rk{(i,j,k,g,h) in T, (p,q,r,s,t) in T: # current variable r = - (c(x) - s)
   !(i=p and j=q and k=r and g=s and h=t)} default 0;
param dk{(i,j,k,g,h) in T, (p,q,r,s,t) in T: # current dual variables (y_k)
   !(i=p and j=q and k=r and g=s and h=t)} default 0;

minimize f:
   sum{(i,j,k,g,h) in T}
   (
      (if c[i,j,k,g,h] - alpha[k] >= epsilon then
          - lambda[i,j,k,g,h] *
              ((c[i,j,k,g,h] - alpha[k])^(1-1/gamma[h]) / (1-1/gamma[h])
               - psi[g]*(y[i,j,k,g,h]/w[i])^mu1[j] / mu1[j])
       else
          - lambda[i,j,k,g,h] *
         (-   0.5/gamma[h] * epsilon^(-1/gamma[h]-1) * (c[i,j,k,g,h] - alpha[k])^2
          + ( 1+1/gamma[h])* epsilon^(-1/gamma[h]  ) * (c[i,j,k,g,h] - alpha[k])
          + (1/(1-1/gamma[h]) - 1 - 0.5/gamma[h]) * epsilon^(1-1/gamma[h])
               - psi[g]*(y[i,j,k,g,h]/w[i])^mu1[j] / mu1[j])
      )
   + 0.5 * primreg * (c[i,j,k,g,h]^2 + y[i,j,k,g,h]^2)
   )
 + sum{(i,j,k,g,h) in T, (p,q,r,s,t) in T: !(i=p and j=q and k=r and g=s and h=t)}
       (dk[i,j,k,g,h,p,q,r,s,t] * R[i,j,k,g,h,p,q,r,s,t]
                   + 0.5 * rhok * R[i,j,k,g,h,p,q,r,s,t]^2);

subject to

Incentive{(i,j,k,g,h) in T, (p,q,r,s,t) in T:
          !(i=p and j=q and k=r and g=s and h=t)}:
   (if c[i,j,k,g,h] - alpha[k] >= epsilon then
      (c[i,j,k,g,h] - alpha[k])^(1-1/gamma[h]) / (1-1/gamma[h])
       - psi[g]*(y[i,j,k,g,h]/w[i])^mu1[j] / mu1[j]
    else
       -  0.5/gamma[h] *epsilon^(-1/gamma[h]-1)*(c[i,j,k,g,h] - alpha[k])^2
       + (1+1/gamma[h])*epsilon^(-1/gamma[h]  )*(c[i,j,k,g,h] - alpha[k])
       + (1/(1-1/gamma[h]) - 1 - 0.5/gamma[h])*epsilon^(1-1/gamma[h])
       - psi[g]*(y[i,j,k,g,h]/w[i])^mu1[j] / mu1[j]
   )
 - (if c[p,q,r,s,t] - alpha[k] >= epsilon then
      (c[p,q,r,s,t] - alpha[k])^(1-1/gamma[h]) / (1-1/gamma[h])
       - psi[g]*(y[p,q,r,s,t]/w[i])^mu1[j] / mu1[j]
    else
       -  0.5/gamma[h] *epsilon^(-1/gamma[h]-1)*(c[p,q,r,s,t] - alpha[k])^2
       + (1+1/gamma[h])*epsilon^(-1/gamma[h]  )*(c[p,q,r,s,t] - alpha[k])
       + (1/(1-1/gamma[h]) - 1 - 0.5/gamma[h])*epsilon^(1-1/gamma[h])
       - psi[g]*(y[p,q,r,s,t]/w[i])^mu1[j] / mu1[j]
   )
 + R[i,j,k,g,h,p,q,r,s,t] >= 0;

Technology:
   sum{(i,j,k,g,h) in T} lambda[i,j,k,g,h]*(y[i,j,k,g,h] - c[i,j,k,g,h]) >= 0;
```

4.2. Tax model data. File `pTax5Dncl.dat` provides data for a specific problem.

```
# pTax5Dncl.dat
# 08 Dec 2017: pTax5Dncl files added to multiscale website.

data;

let na := 5;
let nb := 3;
let nc := 3;
let nd := 2;
let ne := 2;

# Set up wage dimension intervals
let wmin := 2;
let wmax := 4;
let {i in A}  w[i]   := wmin + ((wmax-wmin)/(na-1))*(i-1);

data;

param  mu :=
    1   0.5
    2   1
    3   2 ;

# Define mu1
let {j in B} mu1[j] := mu[j] + 1;

data;

param alpha :=
    1   0
    2   1
    3   1.5;

param psi :=
    1   1
    2   1.5;

param gamma :=
    1   2
    2   3;

# Set up 5 dimensional distribution
let {(i,j,k,g,h) in T} lambda[i,j,k,g,h] := 1;

# Choose a reasonable epsilon
let epsilon := 0.1;
```

4.3. Initial values. File `pTax5Dinitial.run` solves a simplified model to compute starting values for Algorithm NCL. The nonlinear inequality constraints are removed, and $y = c$ is enforced. This model solves easily with MINOS or SNOPT on all cases tried. Solution values are output to file `p5Dinitial.dat`.

```
# pTax5Dinitial.run
# 08 Dec 2017: pTax5Dncl files added to multiscale website.

# Define parameters for agents (taxpayers)
param na := 5;       # number of types in wage
param nb := 3;        # number of types in eta
param nc := 3;        # number of types in alpha
param nd := 2;        # number of types in psi
param ne := 2;        # number of types in gamma
set A := 1..na;       # set of wages
set B := 1..nb;       # set of eta
set C := 1..nc;       # set of alpha
set D := 1..nd;      # set of psi
set E := 1..ne;       # set of gamma
set T = {A,B,C,D,E};     # set of agents

# Define wages for agents (taxpayers)
param  wmin := 2;        # minimum wage level
param  wmax := 4;        # maximum wage level
param  w {i in A} := wmin + ((wmax-wmin)/(na-1))*(i-1);  # wage vector

# Choose a reasonable epsilon
param epsilon := 0.1;

# mu vector
param mu {B};            # mu = 1/eta
param mu1{B};            # mu1[j] = mu[j] + 1
param alpha {C};
param gamma {E};
param psi {D};

var c {(i,j,k,g,h) in T} >= 0.1;
var y {(i,j,k,g,h) in T} >= 0.1;

maximize f: sum{(i,j,k,g,h) in T}
   if c[i,j,k,g,h] - alpha[k] >= epsilon then
     (c[i,j,k,g,h] - alpha[k])^(1-1/gamma[h]) / (1-1/gamma[h])
      -  psi[g] * (y[i,j,k,g,h]/w[i])^mu1[j] / mu1[j]
   else
      -  0.5/gamma[h] *epsilon^(-1/gamma[h]-1)*(c[i,j,k,g,h] - alpha[k])^2
      + (1+1/gamma[h])*epsilon^(-1/gamma[h])  *(c[i,j,k,g,h] - alpha[k])
      + (1/(1-1/gamma[h]) -1 - 0.5/gamma[h])*epsilon^(1-1/gamma[h])
      -  psi[g] * (y[i,j,k,g,h]/w[i])^mu1[j] / mu1[j];

subject to
   Budget {(i,j,k,g,h) in T}: y[i,j,k,g,h] - c[i,j,k,g,h] = 0;

let {(i,j,k,g,h) in T} y[i,j,k,g,h] := i+1;
let {(i,j,k,g,h) in T} c[i,j,k,g,h] := i+1;

data;

param  mu :=
    1   0.5
    2   1
    3   2 ;

```

```
# Define mu1
let {j in B} mu1[j] := mu[j] + 1;

data;

param alpha :=
    1   0
    2   1
    3   1.5;

param psi :=
    1   1
    2   1.5;

param gamma :=
    1   2
    2   3;

 option solver minos;
 option solver snopt;
 option show_stats 1;

option minos_options  ' \
   summary_file=6       \
   print_file=9         \
   scale=no             \
   print_level=0        \
  *minor_iterations=200 \
   major_iterations=2000\
   iterations=50000     \
   optimality_tol=1e-7  \
  *penalty=100.0        \
   completion=full      \
  *major_damp=0.1       \
   superbasics_limit=3000\
   solution=yes         \
  *verify_level=3       \
';

option snopt_options  ' \
   summary_file=6       \
   print_file=9         \
   scale=no             \
   print_level=0        \
   major_iterations=2000\
   iterations=50000     \
   optimality_tol=1e-7  \
  *penalty=100.0        \
   superbasics_limit=3000\
   solution=yes         \
  *verify_level=3       \
';


display na,nb,nc,nd,ne;
solve;
display na,nb,nc,nd,ne;
display y,c >p5Dinitial.dat;
close p5Dinitial.dat;
```

4.4. NCL implementation. File `pTax5Dnclipopt.run` uses files

```
pTax5Dinitial.run
pTax5Dncl.mod
pTax5Dncl.dat
pTax5Dinitial.dat
```

to implement Algorithm NCL. Subproblems NC_k are solved in a loop until $\|r_k^*\|_\infty \leq$ `rtol = 1e-6`, or η_k has been reduced to parameter `etamin = 1e-8`, or ρ_k has been increased to parameter `rhomax = 1e+8`. The loop variable k is called `K` to avoid a clash with subscript `k` in the model file.

Optimality tolerance $\omega_k = 10^{-6}$ is used throughout to ensure that the solution of the final subproblem NC_k will be close to a solution of the original problem if $\|r_k^*\|_\infty$ is small enough for the final k ($\|r_k^*\|_\infty \leq$ `rtol = 1e-6`).

IPOPT is used to solve each subproblem NC_k, with runtime options set to implement increasingly warm starts.

```
# pTax5Dnclipopt.run
# 08 Dec 2017: pTax5Dncl files added to multiscale website.

reset;   model pTax5Dinitial.run;
reset;   model pTax5Dncl.mod;
data  pTax5Dncl.dat;
data; var include p5Dinitial.dat;

model;
option solver ipopt;
option show_stats 1;

option ipopt_options '\
   dual_inf_tol=1e-6  \
   max_iter=5000      \
';
option opt2 $ipopt_options ' warm_start_init_point=yes';

# NCL method.
# kmax, rhok, rhofac, rhomax, etak, etafac, etamin, rtol
# are defined in the .mod file.

printf "NCLipopt log for pTax5D\n" > 5DNCLipopt.log;
display na, nb, nc, nd, ne, primreg  > 5DNCLipopt.log;
printf "   k      rhok      etak     rnorm        Obj\n" > 5DNCLipopt.log;

for {K in 1..kmax}
{  display na, nb, nc, nd, ne, primreg, K, kmax, rhok, etak;
   if K == 2 then {option ipopt_options $opt2 ' mu_init=1e-4'};
   if K == 4 then {option ipopt_options $opt2 ' mu_init=1e-5'};
   if K == 6 then {option ipopt_options $opt2 ' mu_init=1e-6'};
   if K == 8 then {option ipopt_options $opt2 ' mu_init=1e-7'};
   if K ==10 then {option ipopt_options $opt2 ' mu_init=1e-8'};
   display $ipopt_options;
   solve;

   let rmax := max({(i,j,k,g,h) in T, (p,q,r,s,t) in T:
      !(i=p and j=q and k=r and g=s and h=t)} R[i,j,k,g,h,p,q,r,s,t]);
   let rmin := min({(i,j,k,g,h) in T, (p,q,r,s,t) in T:
      !(i=p and j=q and k=r and g=s and h=t)} R[i,j,k,g,h,p,q,r,s,t]);
   display na, nb, nc, nd, ne, primreg, K, rhok, etak, kmax;
   display K, kmax, rmax, rmin;
   let rnorm := max(abs(rmax), abs(rmin));   # ||r||_inf
```

```
   printf "%4i %9.1e %9.1e %9.1e %15.7e\n", K, rhok, etak, rnorm, f >> 5DNCLipopt.log;
   close 5DNCLipopt.log;

   if rnorm <= rtol then
   { printf "Stopping: rnorm is small\n"; display K, rnorm; break; }

   if rnorm <= etak then # update dual estimate dk; save new solution
   {let {(i,j,k,g,h) in T, (p,q,r,s,t) in T:
         !(i=p and j=q and k=r and g=s and h=t)}
            dk[i,j,k,g,h,p,q,r,s,t] :=
            dk[i,j,k,g,h,p,q,r,s,t] + rhok*R[i,j,k,g,h,p,q,r,s,t];
    let {(i,j,k,g,h) in T} ck[i,j,k,g,h] := c[i,j,k,g,h];
    let {(i,j,k,g,h) in T} yk[i,j,k,g,h] := y[i,j,k,g,h];
    display K, etak;
    if  etak == etamin then { printf "Stopping: etak = etamin\n"; break; }
    let etak := max(etak*etafac, etamin);
    display etak;
   }
   else # keep previous solution; increase rhok
   { let {(i,j,k,g,h) in T} c[i,j,k,g,h] := ck[i,j,k,g,h];
     let {(i,j,k,g,h) in T} y[i,j,k,g,h] := yk[i,j,k,g,h];
     display K, rhok;
     if  rhok == rhomax then { printf "Stopping: rhok = rhomax\n"; break; }
     let rhok := min(rhok*rhofac, rhomax);
     display rhok;
   }
}

display c,y;  display na, nb, nc, nd, ne, primreg, rhok, etak, rnorm;

# Count how many constraint are close to being active.
data;
let nT   := na*nb*nc*nd*ne;   let m := nT*(nT-1);   let n := 2*nT;
let etak := 1.0001e-10;
printf "\n m = %8i\n n = %8i\n", m, n >> 5DNCLipopt.log;
printf "\n Constraints within tol of being active\n\n" >> 5DNCLipopt.log;
printf "    tol      count    count/n\n" >> 5DNCLipopt.log;

for {K in 1..10}
{ let kmax := card{(i,j,k,g,h) in T, (p,q,r,s,t) in T:
                   !(i=p and j=q and k=r and g=s and h=t)
                   and Incentive[i,j,k,g,h,p,q,r,s,t].slack <= etak};
  printf "%9.1e %8i %8.1f\n", etak, kmax, kmax/n >> 5DNCLipopt.log;
  let etak := etak*10;
}
printf "Created 5DNCLipopt.log\n";
```

5. Conclusions. This work has been illuminating in several ways as we sought to improve our ability to solve examples of problem TAX.

- Small examples of the tax model solve efficiently with MINOS and SNOPT, but eventually fail to converge as the problem size increases.
- IPOPT also solves small examples efficiently, but eventually starts requesting additional memory for the MUMPS sparse linear solver. The solver may freeze, or the iterations may diverge.
- The NC_k subproblems are not suitable for MINOS or SNOPT because of the large number of variables (x, r) and the resulting number of superbasic variables (although warm-starts are natural).
- It is often said that interior methods cannot be warm-started. Nevertheless, IPOPT has several runtime options that have proved to be extremely helpful

for implementing Algorithm NCL. For the results obtained here, it has been sufficient to say that warm starts are wanted for $k > 1$, and that the IPOPT barrier parameter should be initialized at decreasing values for later k (where only the objective of subproblem NC_k changes with k).

- The numerical examples of section 3 had $3n$, $3n$ and $6.6n$ constraints essentially active at the solution, yet were solved successfully. They suggest that the NCL approach with an interior method as subproblem solver can overcome LICQ difficulties on problems that could not be solved directly.

Acknowledgments. We are extremely grateful to the developers of AMPL and IPOPT for making the development and evaluation of Algorithm NCL possible. We are especially grateful to Mehiddin Al-Baali and other organizers of the NAO-IV conference *Numerical Analysis and Optimization* at Sultan Qaboos University, Muscat, Oman, which brought the authors and AMPL developers together in January 2017.

REFERENCES

[1] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM Journal on Matrix Analysis and Applications, 23 (2001), pp. 15–41, doi:10.1137/S0895479899358194.

[2] S. Arreckx and D. Orban, *A regularized factorization-free method for equality-constrained optimization*, Technical Report GERAD G-2016-65, GERAD, Montréal, QC, Canada, 2016, doi:10.13140/RG.2.2.20368.00007.

[3] R. H. Byrd, J. Nocedal, and R. A. Waltz, *Knitro: An integrated package for nonlinear optimization*, in Large-Scale Nonlinear Optimization, G. Di Pillo and M. Roma, eds., Springer US, Boston, MA, 2006, pp. 35–59, doi:10.1007/0-387-30065-1_4.

[4] A. R. Conn, N. I. M. Gould, and P. Toint, *A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds*, SIAM J. Numer. Anal., 28 (1991), pp. 545–572, doi:10.1137/0728030.

[5] A. R. Conn, N. I. M. Gould, and P. Toint, *LANCELOT: A Fortran Package for Large-scale Nonlinear Optimization (Release A)*, Lecture Notes in Computation Mathematics 17, Springer Verlag, Berlin, Heidelberg, New York, London, Paris and Tokyo, 1992.

[6] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*, Brooks/Cole, Pacific Grove, second ed., 2002.

[7] M. P. Friedlander and D. Orban, *A primal–dual regularized interior-point method for convex quadratic programs*, Math. Prog. Comp., 4 (2012), pp. 71–107, doi:10.1007/s12532-012-0035-2.

[8] M. P. Friedlander and M. A. Saunders, *A globally convergent linearly constrained Lagrangian method for nonlinear optimization*, SIAM J. Optim., 15 (2005), pp. 863–897, doi:10.1137/S1052623402419789.

[9] P. E. Gill, V. Kungurtsev, and D. P. Robinson, *A stabilized SQP method: global convergence*, IMA J. Numer. Anal., 37 (2017), pp. 407–443.

[10] P. E. Gill, V. Kungurtsev, and D. P. Robinson, *A stabilized SQP method: superlinear convergence*, Math. Program., Ser. A, 163 (2017), pp. 369–410.

[11] P. E. Gill, W. Murray, and M. A. Saunders, *SNOPT: An SQP algorithm for large-scale constrained optimization*, SIAM Review, 47 (2005), pp. 99–131, doi:10.1137/S0036144504446096. SIGEST article.

[12] *IPOPT open source NLP solver*. https://projects.coin-or.org/Ipopt.

[13] K. L. Judd, D. Ma, M. A. Saunders, and C.-L. Su, *Optimal income taxation with multidimensional taxpayer types*. Working paper, Hoover Institution, Stanford University, 2017.

[14] *KNITRO optimization software*. https://www.artelys.com/tools/knitro_doc/2_userGuide.html.

[15] *LANCELOT optimization software*. http://www.numerical.rl.ac.uk/lancelot/blurb.html.

[16] B. A. Murtagh and M. A. Saunders, *A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints*, Math. Program. Study, 16 (1982), pp. 84–117.

[17] *NCL*. http://stanford.edu/group/SOL/multiscale/models/NCL/.

[18] S. M. Robinson, *A quadratically-convergent algorithm for general nonlinear programming problems*, Math. Program., 3 (1972), pp. 145–156, doi:10.1007/BF01584986.

[19] A. Wächter and L. T. Biegler, *On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming*, Math. Program., 106 (2006), doi:10.1007/s10107-004-0559-y.