Stanford University, ICME

CME 338 Large-Scale Numerical Optimization

Instructor: Michael Saunders Spring 2019

Notes 6: **The Primal Simplex Method**

1 Linear optimization terminology

When it was first used in the context of operations research (around the time of World War II), the word *programming* conveyed the idea of *planning* or *scheduling*. For decades, *linear programming* and *mixed integer programming* have been synonymous with (very large) scheduling problems. The acronyms LP, QP, NLP, MIP, MINLP, SOCP, SDP have denoted increasingly complex problems in the field of *mathematical programming*.

It is hard to break away from such names, but in 2010 the Mathematical Programming Society officially changed its name to the Mathematical Optimization Society (MOS). Evidently we should now talk about LO problems (and QO, NLO, MILO, MINLO, SOCO, SDO problems)! Note that the main MOS journals are still called *Mathematical Programming* (Series A and B) and *Mathematical Programming Computation* (MPC)!

2 The LO problem

Mathematical optimization systems such as CLP, CPLEX, Gurobi, MOSEK, XpressMP, ... are designed to solve linear optimization problems in the form

$$\text{LO1} \qquad \begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & \phi(x) = c^T x \\ \text{subject to} & \ell \le \begin{pmatrix} x \\ Ax \end{pmatrix} \le u, \end{array}$$

where A is a sparse matrix ($m \times n$). Typically $m < n$ or $m \ll n$, but some problems have $m \gg n$. In practice, many of the variables are likely to have *vanilla bounds*: $\ell_j = 0$, $u_j = \infty$. Sometimes there are *free variables* with both bounds infinite: $\ell_j = -\infty$, $u_j = \infty$. There may also be *fixed variables* for which $\ell_j = u_j = \beta$, say (especially with equality constraints $a_i^T x = \beta$). Other variables are simply called *bounded*.

The main question is *Which bounds are active at a solution?* Clearly, free variables should be the *best kind* because we know in advance that their bounds are not active. Good implementations (and text books!) take advantage of free variables.

Since bounds on variables are easier to work with than general inequalities, implementations introduce a full set of "slack variables" that are *defined* by the relation $s = Ax$. They then treat problem LO1 in the following equivalent form:

$$\text{LO2} \qquad \begin{array}{ll} \underset{x\in\mathbb{R}^n,\, s\in\mathbb{R}^m}{\text{minimize}} & \phi(x) = c^T x \\ \text{subject to} & (A \;\; -I)\begin{pmatrix} x \\ s \end{pmatrix} = 0, \quad \ell \le \begin{pmatrix} x \\ s \end{pmatrix} \le u. \end{array}$$

The matrix $-I$ is usually represented implicitly (sometimes it is $+I$), and its presence helps ensure that certain matrices have full rank. Otherwise, it is important to understand that solution algorithms can treat x and s equally. *Those are the variables in the problem.*

As a result, there is no loss of generality if we describe algorithms in terms of the simpler problem

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

where x includes s above, and b could be zero any vector. If we still define A to be $m \times n$, we now have $m \le n$ and $\text{rank}(A) = m$.

Commercial mathematical optimization systems are likely to contain the following main algorithms for LO problems (Dantzig [2], Vanderbei [13]):

Primal simplex
Dual simplex
Barrier (usually a primal-dual interior method).

The simplex algorithms maintain $Ax = b$ throughout, and a tentative solution x is "infeasible" if the bounds $\ell \le x \le u$ are not satisfied (to within some tolerance). In contrast, Barrier maintains $\ell < x < u$ strictly (when $l_j < u_j$), and x is "infeasible" if $Ax \neq b$ (to within some tolerance).

3 The primal simplex method

All versions of the simplex method are *active-set methods* in which n constraints are active at the beginning and end of each iteration. The m constraints $Ax = b$ are always active, and an additional $n - m$ bounds (or temporary bounds) are active. This setup partitions x into $n - m$ *nonbasic variables* x_N that are temporarily frozen at their current values, and m *basic variables* that will be free to move at the next iteration after one of the nonbasic variables is chosen to move. For some permutation P, the columns of A are partitioned accordingly:

$$AP = \begin{pmatrix} B & N \end{pmatrix}, \qquad x = P\begin{pmatrix} x_B \\ x_N \end{pmatrix}, \qquad Ax = Bx_B + Nx_N = b, \tag{1}$$

where P must be chosen so that the $m \times m$ *basis matrix* B is nonsingular.[1] We assume that linear systems of the form $Bp = a$ and $B^T y = d$ can be solved for any given vectors a and d. Often a and d are *sparse right-hand sides*, and for extremely large problems the sparsity should be taken into account. In particular, most right-hand sides a are columns of A itself. Typically there are only 1 to 10 nonzeros in each column of A *regardless of the problem dimensions*.

The primal simplex method is an *active-set method*. Given a nonsingular B and values of x_N satisfying $\ell_N \le x_N \le u_N$, the *current active set* defines a *vertex* of the form

$$\begin{pmatrix} B & N \\ & I \end{pmatrix}\begin{pmatrix} x_B \\ x_N \end{pmatrix} = \begin{pmatrix} b \\ x_N \end{pmatrix}$$

(n active constraints and n variables). Primal simplex begins by solving $Bx_B = b - Nx_N$ and taking x_B to be new values for the basic variables. From (1), this ensures that $Ax = b$. If the new x_B satisfies its bounds, the new x is "feasible" and primal simplex may proceed normally. Otherwise, an almost identical Phase 1 procedure is employed to push x_B toward feasibility. Phase 1 is described in section 5.

The steps in section 3.1 are constructive. They illustrate how to generate a search direction p that is a *descent direction* for the objective $\phi(x)$. Primal simplex does this by considering "Shall we move one of the nonbasic variables either up or down".

If there is no such direction, the current x is an optimal solution, and the constraints $Ax = b$ along with the active bounds on the nonbasic variables are the *optimal active set*. Otherwise it is good to move as far as possible along the search direction because $\phi(x)$ is linear. Usually a basic variable reaches a bound and a basis exchange takes place. The process then repeats.

Similar steps are used by the *reduced-gradient (RG) method* for problems in which $\phi(x)$ is a nonlinear objective. The main difference is that the RG method moves several nonbasic variables simultaneously (we call them *superbasic*) and it might be better not to move too far along the search direction.

[1] For simplicity, let's assume that $P = I$ initially.

3.1 Primal simplex without a_{ij} or B^{-1}

Search direction p Assume $\ell \le x \le u$ and $Ax = b$. To maintain $A(x+p) = b$ for some search direction p, we require $Ap = 0$. Thus, if the nonbasic variables in (1) change in some direction p_N, we must solve

$$Bp_B = -Np_N.$$

Effect on objective In order to choose a "good" p_N, suppose y and z_N are defined by

$$B^T y = c_B, \qquad z_N = c_N - N^T y. \tag{2}$$

The objective changes from its current value $\phi = c^T x$ to $\bar{\phi}$ as follows:

$$\begin{aligned}\bar{\phi} &= c^T(x+p)\\ &= \phi + c_B^T p_B + c_N^T p_N\\ &= \phi + y^T B p_B + c_N^T p_N\\ &= \phi - y^T N p_N + c_N^T p_N\\ &= \phi + z_N^T p_N.\end{aligned}$$

The elements of z_N are the *reduced costs* or *reduced gradients*.

Choice of nonbasic to move Primal simplex moves only one nonbasic variable at a time. Suppose we try to move variable x_s. Let $\sigma = \pm 1$ because we're not sure yet if x_s should increase or decrease. Then

$$p_N = \sigma e_s, \qquad Bp_B = -\sigma a_s, \qquad \bar{\phi} = \phi + \sigma z_s. \tag{3}$$

Hence, the objective will improve if $\sigma = 1,\ z_s < 0$ or if $\sigma = -1,\ z_s > 0$. In general it is good if $|z_s|$ is large. The search for z_s is called *Pricing*.

Optimality No improvement is possible if one of the following conditions holds for every nonbasic variable x_j:

$$\begin{aligned}\ell_j < x_j < u_j \quad &\text{and} \quad z_j = 0,\\ x_j = \ell_j \quad &\text{and} \quad z_j \ge 0,\\ x_j = u_j \quad &\text{and} \quad z_j \le 0.\end{aligned}$$

We know that $z_B \equiv c_B - B^T y = 0$ by definition of y. Thus we can say that a triple (x, y, z) solves problem LO if x is feasible ($Ax = b$ and $\ell \le x \le u$) and if

$$z = c - A^T y, \qquad \min(x - \ell, z) = 0, \qquad \min(u - x, -z) = 0.$$

Similar tests are suitable when the objective or constraints are nonlinear.

Steplength If (x, y, z) is not optimal, the search direction is computed as in (3). The objective improves as much as possible in that direction if we choose a steplength α to solve the one-dimensional problem

$$\max\ \alpha \ \text{ subject to } \ \ell \le x + \alpha p \le u.$$

This is often called *the ratio test*. One of the variables x_r will reach its bound first. Note that the winner is sometimes x_s: it moves from its current value to one of its bounds. In this case, B is unaltered.

Basis change In general, $r \ne s$ and one of the *basic* variables is first to reach a bound. Further improvement might be possible if we switch x_s and x_r in B and N. The basis is *updated*, $x \leftarrow x + \alpha p$, and all steps are repeated.

4 Simplex facts

- Variables may have any bounds, including ones where $\ell_j = u_j$.
- x includes a full set of slack variables. There are no "artificial variables".
- Nonbasic variables need not be zero—they may have any value satisfying their bounds. "Nonbasic" means *temporarily frozen at current value*. Admittedly, most nonbasic variables are *equal* to one of their bounds (though not if both bounds are infinite!).
- Simplex methods start from a specified basis and solution (B, N, x_B, x_N). If B is nonsingular, it enables us satisfy $Ax = b$.
- Free variables are usually in the basis. They are "best" for the simplex method because they always satisfy their (infinite) bounds and therefore won't generate basis changes.
- (Partial Pricing) If $n \gg m$, the computation of the reduced-gradient vector $z_N = c_N - N^T y$ is expensive. We may compute only part of z_N each iteration and choose any element z_s that is reasonably large and has appropriate sign.
- (Steepest-Edge Pricing) Alternatively, the improvement per unit step is greater if $c^T p/\|p\|$ is large (and negative). This means $|z_s|/\omega_s$ should be large, where ω_j is defined as follows for each nonbasic variable x_j:

  $$Bp_j = a_j, \qquad \omega_j = \left\| \begin{pmatrix} -\sigma p_j \\ \sigma \end{pmatrix} \right\| = \left\| \begin{pmatrix} p_j \\ -1 \end{pmatrix} \right\|.$$

  When the chosen a_s replaces the rth column of B, it would be too expensive to re-solve for each p_j in order to compute $\omega_j^2 = (\|p_j\|_2^2 + 1)$ [5, 3]. Instead, the systems

  $$B^T z = p_s \quad \text{and} \quad B^T w = e_r$$

  are solved and the steepest-edge weights are updated according to

  $$\theta_j = (w^T a_j)/p_{rs}, \qquad \bar{\omega}_j = \omega_j - 2\theta_j\, z^T a_j + \theta_j^2\, \omega_s.$$

- We *don't invert* B. We have to solve systems involving B and B^T, using a factorization of B and updating the factors when one column of B is replaced by a column of N.
- After some number of updates (typically 50 to 100), the current B is factorized directly and the basic variables x_B are recomputed so that $Ax = b$. Any number of basic variables may prove to be infeasible. Phase 1 iterations are initiated or continued if necessary. (Thus, Phase 1 may be initiated more than once.)
- In general, x is regarded as feasible if $Ax = b$ to high accuracy and

  $$\ell - \delta_P \le x \le u + \delta_P, \qquad \text{or equivalently} \qquad \min(x - \ell, u - x) \ge -\delta_P,$$

  where δ_P is a small *feasibility tolerance*. Typically $\delta_P = 10^{-6}$, 10^{-7}, or 10^{-8}. It helps prevent simplex from jumping in and out of Phase 1. It also allows for noisy data that might otherwise have no feasible x.
- If x is feasible, (x, y, z) is regarded as optimal if nonbasic variables x_j satisfy

  $$x_j > \ell_j \ \text{ and } \ z_j \le \delta_D \qquad \text{or} \qquad x_j < u_j \ \text{ and } \ z_j \ge -\delta_D,$$

  where δ_D is a small *optimality tolerance*. Typically $\delta_D = 10^{-6}$, 10^{-7}, or 10^{-8}. (Almost nobody needs 9 or more digits of precision in the optimal objective value—except for large examples of Flux Balance Analysis (FBA) and Flux Variability Analysis (FVA), as mentioned in the NEOS Metabolic Engineering Problem! For such cases, we can now apply SoPlex [12] or a quadruple-precision version of MINOS [9].
- Absolute tolerances of this nature assume that the problem is *well scaled*. This means that the largest elements in A, x, y and z should be of order 1. Automatic scaling of the rows and columns of A can usually achieve this (but wise modelers choose good units from the beginning).

5 Phase 1

Primal simplex performs a "Phase 1 iteration" whenever some of the current basic variables are infeasible. Phase 1 defines a *new problem every iteration*. The new problem is feasible (the violated bounds on x_B are relaxed to $\pm\infty$) and has a *new objective vector* $\widetilde{c}$ that seeks to reduce the *sum of infeasibilities* for the original problem. The elements of $\widetilde{c}$ are zero except for those corresponding to infeasible basic variables:

$$\widetilde{c}_j = \begin{cases} +1 & \text{if } x_j > u_j + \delta_P, \\ -1 & \text{if } x_j < \ell_j - \delta_P. \end{cases}$$

A benefit is that *any number of infeasibilities* may be removed in a single iteration (not just 0 or 1). Special care is needed in defining the steplength α when the *last* infeasibility is removed.

In many implementations, the constraints $Ax = b$ include the objective vector as a constraint (with infinite bounds on the associated slack variable). For problem LO2 above, if the objective is in the last row of A, the constraint has the form $c^Tx - s_m = 0$, where $-\infty \le s_m \le \infty$. The system "$B^Ty = c_B$" is therefore always of the form

$$\widetilde{B}^T\widetilde{y} = \begin{pmatrix} B^T & c_B \\ & -1 \end{pmatrix} \begin{pmatrix} y \\ y_m \end{pmatrix} = \begin{pmatrix} \widetilde{c}_B \\ \gamma \end{pmatrix},$$

where $\gamma = 0$ in Phase 1 and $\widetilde{c}_B = 0$ in Phase 2. Note the flexibility of this arrangement. The same factorization of $\widetilde{B}$ may be used for both phases, because only the right-hand side is being "juggled". A single scan of the basic variables is used to define $\widetilde{c}_B$, and then γ is set to 0 or 1 appropriately (or to -1 if x_B is feasible and the objective should be maximized).

6 Basis factorization

Simplex and reduced-gradient implementations factorize B periodically using some kind of sparse LU procedure—a sparse form of Gaussian elimination: $B = LU$, where L and U are permuted sparse triangular matrices.

Certain updating methods allow the basis factorization to be a "black box", but some methods update the LU factors directly and therefore affect the way in which L and U are computed and stored. For example, MA28 and MA48 are able to permute B to *block-triangular form* and then factorize each block separately, but this is not suitable for methods that update a single L and U.

The choice of permutations (the "pivot order") is a delicate balance between sparsity and stability. A *stable* factorization ensures that at least one of the factors (typically L) is well-conditioned. The other factor then tends to reflect the condition of B.

If B is close to singular (especially the very first B), the factorization should be able to signal singularity and report which rows and columns of B appear to be the cause. Such *rank-revealing properties* cannot be guaranteed (there are pathological examples that defeat most practical strategies), but LUSOL (to be discussed later) has options that are increasingly successful at the expense of greater run-time and density of the factors.

If singularity is apparent, the offending basic variables (columns) are replaced by slack variables corresponding to the offending rows. Hence the need for a complete set of slack variables.

Ideally, the numerical values of the rejected basic variables are preserved in order to maintain $Ax = b$ as closely as possible. Hence the wish for nonbasic variables to take any value (within their bounds).

7 Basis updating

Suppose a column $\bar{a}$ replaces the rth column of B to give the next basis $\bar{B}$. Thus,

$$\bar{B} = B + (\bar{a} - a)e_r^T,$$

where e_r is the rth column of the identity. Given some kind of factorization of B, we need to obtain a factorization of $\bar{B}$. We study several methods that have varying degrees of stability, efficiency, and ease of implementation:

The Product-Form update
The Bartels-Golub update
The Forrest-Tomlin update
The block-LU update.

8 Crash procedures

This strange term refers to choosing an initial basis matrix B when none is provided by the user. Several approaches are described in [6, 10].

Triangular crash procedures make several passes through the columns of A, using the sparsity pattern to find a set of columns that form a permuted triangle. Each effective diagonal element should be reasonably large compared to other elements in the same column (say, no smaller than 0.1 times the largest nonzero in the column). The aim is to find an initial basis that is nonsingular and not too ill-conditioned.

Other procedures try to do more by using the simplex method itself. For example, a number of "cheap iterations" can be performed in which the Pricing step does very little searching to find a reduced gradient z_j that has suitable sign and is larger than a modest threshold.

9 Scaling

Unfortunately, some problems are formulated without concern about the size of the elements of A. Row and column scaling may be needed to maintain robustness of the simplex solver. Scaling often reduces the number of simplex iterations (although there are notorious cases that solve much more quickly without such treatment).

Nominally, the nonzero elements of A should be of order 1. We mean that there is no reason to have very large entries, but it is hard to know if *small* entries are small through poor scaling or because they intentionally have little effect on the associated constraints. As suggested by Robert Fourer (private communication, 1979), a good general approach is to make several passes through the columns and rows of A, computing the geometric mean of the nonzeros in each (scaled) column or row and using that as the new scale factor. (The geometric mean is approximated by the square root of the product of the largest and smallest nonzero in each column.)

On problem `pilotja` [11, 8], the scaling procedure in MINOS proceeds as follows:

		Min elem	Max elem	Max col ratio
After	0	2.00E-06	5.85E+06	189805175.80
After	1	1.27E-03	1.00E+02	78436.79
After	2	1.59E-02	6.30E+01	3962.95
After	3	2.02E-02	4.94E+01	2441.93
After	4	2.11E-02	4.74E+01	2247.57

The `Min` and `Max` columns refer to the extreme nonzero $|a_{ij}|$ in all of A. The last column `Max col ratio` means $\max_j\{\max_i |a_{ij}| / \min_i |a_{ij}|\}$ among nonzero $|a_{ij}|$. The MATLAB function `gmscale.m` implements the same scaling procedure.

10 Anti-degeneracy

The steplength procedure may return an embarrassing value—namely $\alpha = 0$. The objective function then does not improve. If a sequence of zero steplengths repeats indefinitely, the simplex method is said to be *cycling* (and convergence does not occur).

Indeed it is common for *many* basic variables to be essentially equal to one of their bounds. The current solution is then called *primal degenerate*. The search direction p might want to move several basic variables outside their bounds, and care is needed to ensure that $|p_r|$ is sufficiently large compared to other possible choices.

A naive but effective approach is to make small perturbations to the problem data (typically to the vector b) to reduce the probability that any basic variable would be on its bound.

A more elaborate approach was introduced as the EXPAND procedure in [4]. The key idea is to relax all of the bounds $\ell \leq x \leq u$ by a tiny amount every simplex iteration. In particular, when a variable leaves the basis, it may became nonbasic at a *slightly infeasible value* (no greater than δ_P). Over a period of K iterations, where $K = 10,000$ or 100,000 say, there may be many tiny steplengths α (at which points the simplex method is effectively *stalling*), but it is unlikely that any sequence of iterations will recur.

Unfortunately, the bound relaxation cannot continue indefinitely. After K iterations, infeasible nonbasic variables must be put on their bounds, the basic variables recomputed, and the EXPAND process restarted. There is a danger of the same K iterations repeating, and indeed, Hall and McKinnon [7] have constructed examples that cause EXPAND to cycle. Nevertheless, EXPAND continues to be used successfully within MINOS and SQOPT.

11 Commentary

Many things contribute to efficient implementations of the simplex method. We have touched on some. Bixby [1] describes the remarkable progress made during 1987–2001. For example, a significant item is to take advantage of *sparse right-hand sides* in systems $Bp = a$ and $B^T y = d$. Various forms of simplex remain vital for solving increasingly large linear and mixed-integer models. From cold starts, dual simplex with steepest-edge pricing is often the most efficient solver, but if huge numbers of iterations are required, a barrier solver could well be faster (because it will never take more than $O(100)$ iterations). Similarly for branch-and-bound mixed-integer optimization, dual simplex is good at warm-starting after each branch, but occasionally a switch to barrier is needed for the current subproblem.

References

[1] R. E. Bixby. Solving real-world linear programs: a decade and more of progress. *Operations Research*, 50(1):3–15, 2002.

[2] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

[3] J. J. Forrest and D. Goldfarb. Steepest-edge simplex algorithms for linear programming. *Math. Program.*, 57:341–374, 1992.

[4] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. A practical anti-cycling procedure for linear and nonlinear programming. *Math. Program.*, 45:437–474, 1989.

[5] D. Goldfarb and J. K. Reid. A practicable steepest-edge simplex algorithm. *Math. Program.*, 12:361–371, 1977 (where the title should have read *practical!*).

[6] N. I. M. Gould and J. K. Reid. New crash procedures for large-scale systems of linear constraints. *Math. Program.*, 45:475–501, 1989.

[7] J. A. J. Hall and K. I. M. McKinnon. The simplest examples where the simplex method cycles and conditions where EXPAND fails to prevent cycling. *Math. Program.*, 100(1):133–150, 2004.

[8] LPnetlib collection of LP problems in MATLAB format. `http://faculty.cse.tamu.edu/davis/matrices.html`.

[9] D. Ma, L. Yang, R. M. T. Fleming, I. Thiele, B. O. Palsson, and M. A. Saunders. Reliable and efficient solution of genome-scale models of Metabolism and macromolecular Expression. *Sci. Rep.*, 7, Article number: 40863, http://rdcu.be/oCpn, 2017.

[10] I. Maros and G. Mitra. Strategies for creating advanced bases for large-scale linear programming problems. *INFORMS J. on Computing*, 10:248–260, 1998.

[11] Netlib collection of LP problems in MPS format. `http://www.netlib.org/lp/data`.

[12] SoPlex linear programming solver. `http://soplex.zib.de/`.

[13] R. J. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer, Boston, London and Dordrecht, second edition, 2001.