TUTORIALS IN
OPERATIONS RESEARCH
2014
Bridging Data and Decisions
Alexandra Newman and Janny Leung, Tutorials Co-Chairs and Volume Editors
J. Cole Smith, Series Editor
OR
Presented at the INFORMS Annual Meeting, November 9–12, 2014

INFORMS TutORials in Operations Research

Identification, Assessment, and Correction of Ill-Conditioning and Numerical Instability in Linear and Integer Programs

Ed Klotz

TUTORIALS IN
OPERATIONS RESEARCH
INFORMS 2014

© 2014 INFORMS | ISBN 978-0-9843378-5-9
http://dx.doi.org/10.1287/educ.2014.0130

Identification, Assessment, and Correction of Ill-Conditioning and Numerical Instability in Linear and Integer Programs

Ed Klotz
IBM, Incline Village, Nevada 89451, klotz@us.ibm.com

Abstract The implementation of linear programming (LP) and mixed-integer programming (MIP) algorithms on finite precision computers can create numerical challenges that are not addressed in the mathematical descriptions of these algorithms given in many introductory and more advanced textbooks and courses. Rounding errors associated with finite precision can be magnified because of ill-conditioning or numerical instability, resulting in unexpected, possibly inconsistent results. This tutorial helps the optimization practitioner identify sources of ill-conditioning and numerical instability, assess the cause, and take appropriate remedial action. After discussing some finite precision computing fundamentals, it considers different measures of ill-conditioning, each one of which provides the simplest explanation of ill-conditioning on certain types of LP and MIP models. We then consider remedies for these numerical challenges: (i) optimizer parameter settings that treat the symptoms and (ii) diagnostic tactics that resolve the underlying MIP or LP issue.

Keywords linear programming; integer programming; numerical stability; ill-conditioning; finite precision; numerical linear algebra; tutorials

The fluttering of a butterfly's wing in Rio de Janeiro, amplified by atmospheric currents, could cause a tornado in Texas two weeks later.
—Edward Lorenz, meteorology and chaos theory pioneer (Nader [24], p. 209)

1. Introduction

Operations research practitioners have been formulating and solving linear programs (LPs) since the 1940s (Dantzig [6]). State-of-the-art optimizers such as CPLEX (IBM [16]), Gurobi (Gurobi Optimization [14]), MOSEK (MOSEK ApS [22]), and Xpress-MP (FICO [12]) can solve most practical large-scale linear programs effectively. However, ill-conditioned linear programs and numerically unstable implementations of the underlying algorithms can magnify seemingly irrelevant round-off errors to a level that adversely affects run time or, even worse, yields inconsistent or inaccurate computed solutions. A proper understanding of sources of ill-conditioning can help the practitioner identify, diagnose, and correct these potential issues. This, in turn, results in better performance, more accurate results, and less time spent developing the linear programming model and any associated optimization application. The branch-and-bound algorithm is most frequently used to solve mixed-integer programs (MIPs); since it solves various LP subproblems, this tutorial can be useful when solving MIPs as well as LPs.

In its most general sense, ill-conditioning in a mathematical calculation occurs when a small change to the input results in a large change to the computed output. In other words, given vectors $x \in R^n$ and $y \in R^m$, and $f\colon R^n \to R^m$ with $y = f(x)$, we measure the change Δy in the output resulting from a change Δx in the input. Thus, for $y + \Delta y = f(x + \Delta x)$, we seek a measure κ that provides a bound on the change in y, i.e.,

$$\|\Delta y\| \le \kappa \|\Delta x\|. \tag{1}$$

Depending on the calculation, a quantitative measure of the level of ill-conditioning may be available. Fortunately, for the simplex and barrier algorithms—the most commonly used approaches to solving linear programs in practice—quantitative measures exist for the square linear systems of equations whose solutions are essential for these algorithms.

This paper is primarily intended for the industrial operations research practitioner, researcher, or student who already has some familiarity with the fundamental LP and MIP algorithms, but is less familiar with aspects of finite precision computing and numerical linear algebra. Therefore, we assume some basic familiarity with the simplex and barrier algorithms for linear programming, and branch and bound for integer programming, and do not discuss any algorithmic fundamentals here. The interested reader can refer to basic texts such as Chvátal [3], Dantzig and Thapa [7], Rardin [26], Winston [31], and Bazaraa et al. [2] for details. Or, refer to Klotz and Newman [18] for a quick overview of the key computational steps of the simplex and barrier algorithms. Nonetheless, familiarity with those algorithms is not essential for this paper. The practitioner simply needs to know that these algorithms rely heavily on solving square linear systems of equations for which accurately computed solutions are essential.

This tutorial describes how various characteristics of an LP or MIP model can affect the accuracy of computed solutions to those systems. It distinguishes between treating the symptoms and causes of ill-conditioned LPs and MIPs. Identifying the symptoms is essential, but treating the underlying causes, rather than just the symptoms, provides more robust remedies.

The remainder of the paper is organized as follows. Section 2 describes key fundamentals of finite precision computing that are relevant for the remainder of the paper, including machine precision and the definition of the condition number of a square matrix. We then consider sources of round-off error when solving linear and integer programs, and how they relate to the conditioning of the square linear systems solved by linear programming algorithms. Although the paper focuses on the square basis matrices associated with the simplex algorithms, the concepts and ideas apply to the square matrices associated with the normal equations solved by the barrier algorithm (Fiacco and McCormick [11]) and related interior point algorithms (Dikin [8], Gill et al. [13]). Section 3 then discusses optimizer features to identify and remedy symptoms of ill-conditioning and numerical instability. We consider the relevant features of the CPLEX optimizer, but most other state-of-the-art optimizers have similar features, as well as programming functionality that enables the users to implement any missing ones. Changing optimizer parameter settings typically provides more of a short-term remedy, since doing so treats the symptom of the problem more than the cause. Section 4 discusses some well-known sources of ill-conditioning and numerical instability in LP and MIP models. Practitioners can quickly consult this list of sources and assess if any of them pertain to the LP or MIP in question. For problematic models in which these known sources are not easy to identify, §5 describes effective tactics to help obtain additional information regarding the sources of the numerical problems. These tactics can help identify unstable subsets of the constraints and variables in the model that cause the trouble. These subsets can then be modified or reformulated to improve the conditioning or stability of the model. Section 6 describes some anomalies and contradictions that can occur when running optimizers on finite precision machines. Although practitioners may initially find them surprising, additional examination provides reasonable explanations. Section 7 discusses possible ways to automate the process of diagnosing the cause of ill-conditioning in square linear systems. Section 8 then provides some publicly available test problems that are considered ill-conditioned and uses the troubleshooting tactics described in this tutorial to remedy the problem and improve the accuracy of the computed solution, optimizer performance, or both.

The paper contains numerous segments of CPLEX output on practical LP and MIP models from publicly available models. Unless otherwise stated, these results were obtained by running version 12.6 of CPLEX on a computer with a Dual-Core Intel Xeon 5160 3.0 GHz CPU running Red Hat Linux 5 with 8 gigabytes of memory (a relatively inexpensive machine configuration).

2. Finite-Precision Computing Fundamentals for Solving LPs

A truly comprehensive discussion of finite precision scientific computing is a subject for books rather than papers. We focus here on fundamental finite precision computing concepts and issues specific to linear and integer programming. More detailed discussions can be found in Duff et al. [9] and Higham [15]. The former focuses on sparse linear equations that are essential to the simplex and barrier algorithms, but it also considers such equations in a broader context. The latter is truly comprehensive; it also considers computational issues arising in nonlinear calculations that are not essential to the subject matter in this tutorial.

Because most commonly used computers implement floating point computations in finite precision, arithmetic calculations can be affected by inaccuracies due to round-off errors. We focus the discussion on computing solutions to the square systems of linear equations solved in the primal and dual simplex algorithms. These linear solves cover all the important issues. By contrast, the other operations in the simplex algorithms (e.g., the ratio test and pricing operations) involve simpler computations with fewer potential sources of numerical instability, depend on the results of the linear solves, and are not as directly related to the model data that the practitioner can control.

The curious reader may wonder why this paper discusses remedies to computational challenges arising from finite precision computing rather than simply making use of exact rational calculations. This involves a different floating point model, where rational numbers are stored as integer numerators and denominators. The basic arithmetic operations result in rational numbers as well. Currently this is done via software emulation rather than in hardware. As of this writing, exact rational computations have been implemented, but they typically appear in software packages such as Maple (Cybernet Systems Co. [5]) and Mathematica (Wolfram [32]), which are not large-scale linear programming solvers. The QSopt_ex optimizer (Applegate et al. [1]) reflects significant progress in exact linear programming. It also provides the foundation for additional progress in integer programming (Cook et al. [4]). However, because of the significant extra time required for exact rational computations, even this solver still typically performs some calculations in finite precision. Currently, large-scale LPs and MIPs are solved using optimizers that operate in finite precision. Finite precision computing enables storage of floating point numbers in modest amounts of memory. In addition, basic arithmetic operations can be performed in a reasonable amount of time using floating point registers that require very little memory. In exchange for these advantages, small amounts of round-off error in these calculations can occur, and ill-conditioned models can magnify the errors to the point where they become problematic. At least for the short and medium term, the practitioner should be aware of the LP or MIP input data and the implications of using an optimization algorithm and a finite precision floating point implementation on a model instance with such data.

2.1. Machine Precision

Most commonly used computers implement floating point computations in finite precision. A finite number of bits is available to store each floating point value. Currently, 32-bit single and 64-bit double precision are the most common representations. We focus primarily on 64-bit double precision, the current choice for all state-of-the-art linear and integer programming optimizers. The concepts discussed in the upcoming sections illustrate the benefits of the additional accuracy of 64-bit double precision over 32-bit single precision. Although different computer chips can vary in their implementation of 64-bit doubles, most have similar levels of accuracy. Therefore, we consider the commonly used IEEE standard representation. The conclusions and recommendations derived from the analysis of the IEEE standard carry over to most other 64-bit implementations of floating point numbers.

A 64-bit IEEE double is implemented in base 2. One bit is reserved for the sign of the number. Eleven bits implement the exponent e of the number, providing a capacity of $2^{11} = 2048$ possible exponent values. The acceptable values range from -1021 to 1024. The remaining 52 bits, along with the low order bit of the exponent, define the 53-bit mantissa, which is a base-2 integer between 0 and $2^{53} - 1$. Letting $d_1, d_2, \ldots, d_{53}$ represent the 53 base-2 digits of the mantissa, a floating point number y can be represented as

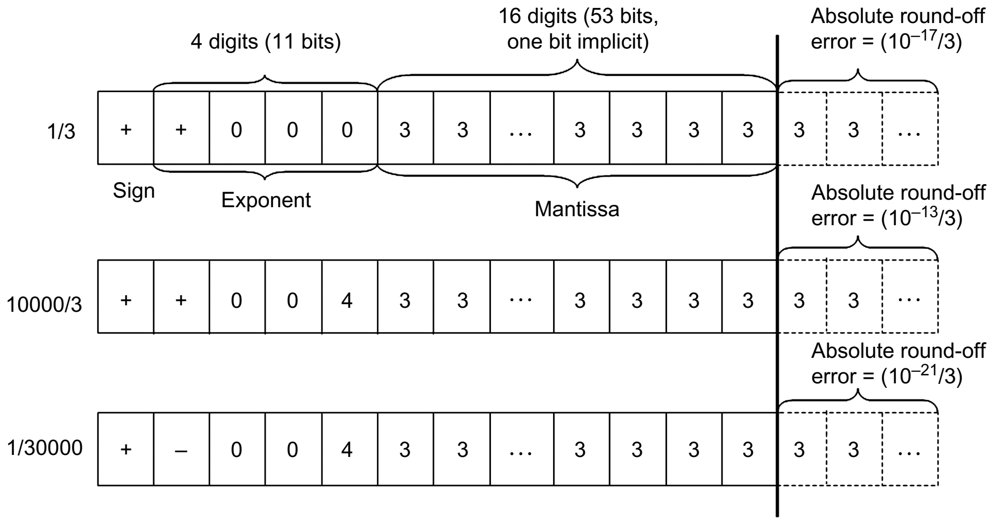
$$y = \pm 2^e (d_1/2 + d_2/2^2 + \cdots + d_{53}/2^{53}). \tag{2}$$

Equivalently, we can view $.d_1 d_2 \ldots d_{53}$ as a base-2 decimal string, and y is obtained by multiplying the string by $\pm 2^e$. For example, since $5/64 = 2^{-3}(1/2 + 1/2^3)$, the base-2 decimal string associated with an exponent of -3 is $0.10100000\ldots0$; i.e., the bits associated with 2^{-1} and 2^{-3} are 1, while the remaining 51 bits are all zero. Note that the IEEE single precision floating point implementation also uses base 2 but only has 24 bits of precision (with one bit shared with the exponent), an 8-bit exponent, and a single bit for the sign. See Higham [15] for a more general description of the precision and exponent range for arbitrary bases and bits.

This implementation of double precision has several implications regarding the representation of numeric data. First, floating point numbers that can be represented as a base-2 (within the IEEE exponent range) integer linear combination of $2^{-1}, 2^{-2}, \ldots, 2^{-53}$ can be represented exactly, whereas all other floating point numbers incur some round-off error in their representation. For example, $7/16 = 1/4 + 1/8 + 1/16 = 2^{-2} + 2^{-3} + 2^{-4}$ can be represented exactly, whereas 1/3 and 1/5 cannot. Second, the maximum relative rounding error between a number and the nearest representable one is 2^{-53}. This value is defined as the machine precision, and translates to approximately 10^{-16} in base 10. In other words, an IEEE double contains 53 digits of precision in base 2, and 16 digits of precision in base 10. Third, the representation in (2) does not uniquely define the representation of a value. For example, 5/64 can be represented (exactly) as $2^0(1/2^4 + 1/2^6)$, $2^{-1}(1/2^3 + 1/2^5)$, $2^{-2}(1/2^2 + 1/2^4)$, or $2^{-3}(1/2 + 1/2^3)$. To ensure a unique representation, the convention of normalized arithmetic assumes that $d_1 \neq 0$ in (2). Thus, for 5/64, the normalized representation is $2^{-3}(1/2 + 1/2^3)$. Normalized arithmetic has specific implications regarding round-off error in representation of and arithmetic operations on floating point numbers, which we consider in more detail. Although other (denormalized) representations exist, they too imply a certain level of round-off error. For the purpose of this tutorial, we need not consider the trade-offs in accuracy between different representations, so we illustrate the issues using normalized arithmetic.

Let us now consider the effects of the magnitude of floating point values on the potential round-off error arising from the representation and arithmetic operations using IEEE doubles. Letting $fl(\cdot)$ represent the floating point representation (IEEE double or otherwise) of a floating point expression, the absolute error of a numeric value y is $|y - fl(y)|$, while the relative error is $|y - fl(y)|/|y|$. For ease of exposition, we consider the base-10 approximation of the 53 binary digit mantissa, which means 16 base-10 digits. Figure 1 illustrates the relative and absolute round-off errors that can arise from floating point values that cannot be represented exactly. Specifically, consider the IEEE double representations of 1/3, $10000/3 = 10^4/3$, and $1/30000 = 10^{-4}/3$. The corresponding exact representations of these three numbers are $0.3333333\ldots$, $3333.3333333\ldots$, and $0.00003333\ldots$, respectively. In each case, the mantissa contains 16 base-10 digits of value 3. The sign bit is the same, and these three numbers have exponents of 0, 4, and -4, respectively. The 16 base-10 digit limitation on the mantissa means that the value 1/3 can only be represented to 16 decimal places. The lack of additional digits implies an absolute round-off error of $10^{-17}/3$. Since the exponent is zero in this case, the relative round-off error is $|fl(1/3) - 1/3|/(1/3) = (10^{-17}/3)/(1/3) = 10^{-17}$. Similarly, the 16-digit limit means that for 10000/3, the loss of precision implies an absolute round-off error

Figure 1. A 64-bit IEEE double floating point representation of 1/3, 10000/3, and 1/30000.


4 digits (11 bits)
16 digits (53 bits, one bit implicit)
Absolute round-off error = ($10^{-17}/3$)
1/3
Sign
Exponent
Mantissa
Absolute round-off error = ($10^{-13}/3$)
10000/3
Absolute round-off error = ($10^{-21}/3$)
1/30000

of $(10^{-17}\cdot 10^4)/3 = 10^{-13}/3$. Normalizing by the exponent yields a relative round-off error of $(10^{-13}/3)/(10^4/3) = 10^{-17}$. Similarly, the absolute round-off error for 1/30000 is $10^{-21}/3$, and the relative round-off error once again is 10^{-17}.

From the preceding discussion, any value that cannot be expressed as a sum of the powers of two described by Equation (2) has an IEEE double representation that creates round-off error. The relative round-off error is bounded by the machine precision (in this case, 10^{-16}). This means that most LP and MIP models already have some round-off error before the optimization algorithm begins. This is true even in the relatively uncommon situation for which all of the model data calculations or measurements were done exactly. Exceptions involve (typically combinatorial or network) models where all coefficients are ± 1. Less frequently in practice, if all model data values are powers of 2 (or, more generally, powers of the base of the floating point implementation if other than 2), no round-off error occurs.

Although the round-off error associated with finite precision representation of data is small, it can increase significantly once additional calculations are involved. Arithmetic operations such as addition, subtraction, multiplication, and division can increase the round-off error, particularly when the numeric operands are of different orders of magnitude. For addition and subtraction, normalized arithmetic requires shifting the exponent of one operand to match the exponent of the other. This increases the loss of precision in one of the operands. To illustrate, consider the addition of 1/3 and 1/30000 under the IEEE double representation. Shifting the exponent of 1/30000 from -4 to 0, as illustrated by Figure 2, introduces four zeros in the high order digits of the mantissa. The fixed length of the mantissa requires the removal of the four low order digits of 3 to compensate, resulting in a loss of four digits. This increases the absolute and relative round-off errors for 1/30000 during the addition by a factor of 10^4 to $10^{-17}/3$ and 10^{-13}, respectively. Fortunately, in this case, this loss of digits does not increase the absolute round-off error of the result of the addition, since 1/3 already has an absolute error of $10^{-17}/3$. However, the situation worsens when we consider the addition of 1/3 and 10000/3, as in Figure 3. Given that the mantissa consists of decimal values, once again we can only shift the exponent of the smaller number upward. This causes a loss of four digits of accuracy for the representation of 1/3, resulting in an absolute round-off error of $10^{-13}/3$. Had no shifting been required because the two operands had the same original exponent, the absolute round-off error would have been $10^{-17}/3$. So we see that performing arithmetic operations on numbers of different orders of magnitude increases the absolute round-off error

Figure 2. Addition of 1/3 and 1/30000.

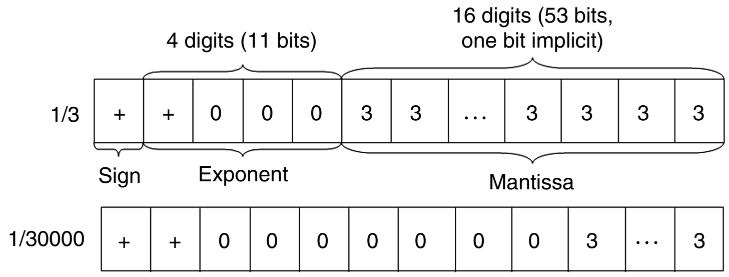
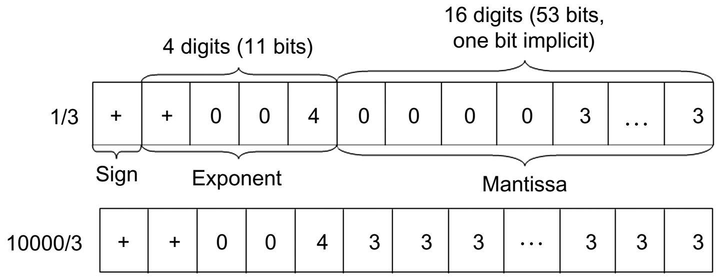


Figure 3. Addition of 1/3 and 10000/3.



that can accumulate compared with the corresponding operations on numbers of the same order of magnitude.

Division operations have additional potential for absolute round-off error. Unlike addition and subtraction, multiplication and division do not require shifting of the exponents. Instead, the exponents are added or subtracted and the mantissas are multiplied or divided. However, other sources of round-off error can arise. In particular, dividing a smaller number into a bigger number can result in more round-off error than the reverse. This is especially important in the basis factorization procedures of the simplex method, whose pivoting operations offer choices regarding the order of pivots performed. To illustrate, consider two floating point values a and b. For simplicity, we assume that a can be represented exactly in IEEE double precision, while $fl(b) = b + \epsilon$. The algebra is more complicated if the IEEE representation of a incurs round-off error, but the issue remains the same. For $a \gg b$, consider the round-off error associated with b/a and a/b. The latter quotient divides a smaller number into a bigger number, whereas the former quotient does the reverse. Let δ measure the round-off error resulting from the division of the mantissas. Then,

$$b/a \approx fl(b/a) = fl(fl(b)/a) = fl((b+\epsilon)/a) = b/a + \epsilon/a + \delta. \tag{3}$$

For a/b, we have

$$a/b \approx fl(a/b) = fl(a/fl(b)) = fl(a/(b+\epsilon)) = a/b - a\epsilon/((b+\epsilon)b) + \delta. \tag{4}$$

The round-off error is much larger in the second case because of the small denominator of the second term in the result of Equation (4). For example, assuming $\delta = 0$, letting $a = 2$ and $b = 1/30000$, the round-off error ϵ to represent b is $10^{-17}/3$. The round-off error from (3) is therefore $10^{-17}/6$, which is on the order of 10^{-17}. By contrast, for (4), the round-off error involves a division by $b^2 = 10^{-8}/9$, yielding a result on the order of 10^{-9}, a much more significant level of round-off error. Smaller values of b further reduce the value of b^2, which then increases the absolute round-off error of $fl(a/b)$.

Additional bits of precision provide one possible way to reduce the round-off errors associated with floating point calculations. With the cost of large amounts of memory having declined dramatically during the past two decades, support for 128-bit quadruple precision values has become commonplace. However, the additional precision does involve significant additional computation time. The nature of the quadruple precision calculation also depends on the operating system and computer chip involved. The size of the registers involved in the quadruple precision calculation, not the number of bits used to store the quadruple precision values in memory, determines the additional accuracy and speed of the calculation. Most computers as of this writing do not have 128-bit floating point registers on which to perform these calculations. For example, Intel chips have 80-bit extended precision registers. This approach compromises between the additional precision and the significant increase in computation time of 128-bit registers relative to 64-bit double precision calculations. Some other chips do use all 128 bits for quadruple precision calculations, but they do so via software that typically combines two 64-bit registers to emulate a 128-bit quadruple precision register. This emulation approach simplifies the chip configuration and offers the accuracy of all 128 bits in the calculation, but it can incur a significant performance degradation relative to doing the calculations in extended precision registers of 128 bits or fewer. The IBM System Z series offers true 128-bit floating point registers for quadruple precision calculations. Such registers perform the 128-bit floating point calculations faster than software emulation and with more accuracy than the 80-bit Intel extended precision registers. But the additional bits also require additional computation time compared with 64- or 80-bit registers.

2.2. The Condition Number of a Square Matrix

Because current linear and integer programming algorithms rely on the solution of square linear systems of equations, a quantitative measure of the condition number κ in Equation (1) exists and can be computed for any basis matrix associated with a linear program or LP relaxation of an integer program. Basis condition numbers can help in assessing the level of ill-conditioning in an LP or MIP by providing a measure of the potential change in computed output relative to a change in the computed input. For example, loss of precision in the input data constitutes such a change in computed input, and the condition number measures the potential magnification of this change on the computed output. Let us consider the well-known derivation of the condition number of a square matrix and how ill-conditioning can affect the optimization of linear programs on finite precision computers. The text for this discussion draws heavily from Klotz and Newman [18]. One can find additional discussions in various numerical linear algebra textbooks, but the original derivation of the condition number of a square matrix goes back to Turing [30].

Consider the following linear program, where x is an $n \times 1$ column vector of continuous-valued, nonnegative decision variables, A is an $m \times n$ matrix of left-hand-side constraint coefficients, c is an $n \times 1$ column vector of objective function coefficients, and b is an $m \times 1$ column vector of right-hand-side data values for each constraint:

$$\begin{aligned} (P_{\text{LP}}): \qquad & \min \quad c^T x \\ & \text{subject to} \quad Ax = b, \\ & \qquad\qquad\quad x \geq 0. \end{aligned}$$

Let B and N represent the set of basic and nonbasic columns of A at a particular iteration of the simplex method. The basic variables x_B solve the linear system

$$A_B x_B = b. \tag{5}$$

The representation (P_{LP}) can be generalized to support nonzero bounds l and u on the variables. In that case, the right-hand side of (5) becomes $\hat{b} = b - A_N x_N$. But the derivation of the condition number remains the same, so we assume lower bounds of 0 and infinite upper bounds on all variables. If this linear system is ill-conditioned, the basic variable values, as well as other values that also depend on solving linear equations involving A_B, can have significant inaccuracies that cause problems for any simplex method implementation. The condition number of A_B therefore provides a useful metric for assessing the potential for unexpected or inconsistent optimizer results on a particular LP or MIP. Let us now derive the condition number of a square matrix and examine how it can be used.

The exact solution of the system of equations in (5) is given by

$$x_B = A_B^{-1} b. \tag{6}$$

Consider a small perturbation, Δb, to the right-hand side of equations (5). We wish to assess the relation between Δb and the corresponding change Δx_B to the computed solution of the perturbed system of equations:

$$A_B(x_B + \Delta x_B) = b + \Delta b. \tag{7}$$

The exact solution of this system of equations (7) is given by

$$(x_B + \Delta x_B) = A_B^{-1}(b + \Delta b). \tag{8}$$

Subtracting equations (6) from those given in (8), we obtain

$$\Delta x_B = A_B^{-1} \Delta b. \tag{9}$$

Applying the Cauchy–Schwarz inequality to equations (9), we obtain

$$\|\Delta x_B\| \le \|A_B^{-1}\| \|\Delta b\|. \tag{10}$$

In other words, the expression (10) gives an upper bound on the maximum absolute change in x_B relative to that of b. Similarly, we can get a relative change in x_B by applying the Cauchy–Schwarz inequality to equations (5):

$$\|b\| \le \|A_B\| \|x_B\|. \tag{11}$$

Multiplying the left- and right-hand sides of (10) and (11) together, and rearranging terms,

$$\frac{\|\Delta x_B\|}{\|x_B\|} \le \|A_B\| \|A_B^{-1}\| \left(\frac{\|\Delta b\|}{\|b\|} \right). \tag{12}$$

From (12), we see that the quantity $\kappa = \|A_B\| \|A_B^{-1}\|$ is a scaling factor for the relative change in the solution, $\|\Delta x_B\|/\|x_B\|$, given a relative change in the right-hand side, $\|\Delta b\|/\|b\|$. The quantity $\kappa \cdot (\|\Delta b\|/\|b\|)$ provides an upper bound on the relative change in the computed solution, $\|\Delta x_B\|/\|x_B\|$, for a given relative change in the right-hand side, $(\|\Delta b\|/\|b\|)$. Recall that ill-conditioning in its most general sense occurs when a small change in the input of a system leads to a large change in the output. The function $\kappa(A_B) = \|A_B\| \|A_B^{-1}\|$ defines the *condition number* of the matrix A_B and enables us to assess the ill-conditioning associated with the system of equations in (5). Larger values of κ imply greater potential for ill-conditioning in the associated square linear system of equations by indicating a larger potential change in the solution given a change in the (right-hand-side) inputs. Because both the simplex and interior point algorithms need to solve square systems of equations, the value of κ and its associated upper bound can help predict how ill-conditioning affects the computed solution of a linear program. The condition number $\kappa(A_B)$ has the same interpretation when applied to small perturbations in A_B; the logic used to derive (12) for a perturbation in the right-hand side can be analogously applied for a perturbation to the matrix.

Note that the preceding derivation of the condition number was performed under perfect precision. No assumptions about a finite precision computing implementation were made.

Therefore, ill-conditioning is a characteristic of the system (in this case, square linear equations) being modeled. It is not a by-product of a finite precision computing concept. However, the round-off errors that can occur in a finite precision computing environment constitute potential perturbations to b or A_B in (5) that can be magnified by a large condition number. Sources of such finite precision-based perturbations when solving LPs and MIPs include the following, most of which are discussed in greater detail in subsequent sections.

1. *Finite representation of exact data*. As seen in §2.1, exactly measured or calculated problem data can still cause round-off error when represented in finite precision. For the IEEE standard for double precision, data values that can be expressed as base-2 integer linear combinations of at most 53 contiguous powers of 2 can be represented exactly (see Equation (2)), whereas other values generate round-off error bounded by the machine precision of 10^{-16}.
2. *Calculation of problem data in finite precision*. Section 2.1 described how the arithmetic operations of addition, subtraction, multiplication, and division can increase the round-off error when performed under finite precision. Specifically, performing any of these operations on values of significantly different orders of magnitude creates more round-off error than the same operation on numbers of similar orders of magnitude. In addition, dividing smaller numbers into significantly larger ones introduces additional round-off error.
3. *Truncation of calculated data*. The practitioner may round or truncate the data after calculating it. Depending on the nature of the data, this practice may either improve or degrade the model numerics. If the practitioner determines the rounding or truncation based on knowledge of the physical system being modeled and the appropriate accuracy level of the data, the practitioner is cleaning up the data, which should reduce the round-off errors in the model. On the other hand, if such rounding or truncating is done arbitrarily, or without full consideration of the implications for the physical model (e.g., calculating data in single precision when the nature of the model requires double precision), it may result in more round-off errors and computed optimal solutions whose activities do not reflect the underlying model.
4. *Errors in physical measurements of data values*. Some LPs and MIPs involve data whose values are measured by a physical device or process (e.g., rates of loss in electrical power transmission lines, miles per gallon for vehicles used in shipping packages). In such cases, the precision of these measurements, along with the potential for ill-conditioning in the model to magnify such precision-based errors to levels above the optimizer tolerances, should be considered.
5. *Errors on algorithmic calculations of data*. For some LPs and MIPs, data values are calculated from other algorithmic computations (e.g., statistical methods to predict demand for production planning or returns for candidate assets of a financial portfolio). If so, the numerical stability of those algorithms, the conditioning of the problems they solve, and the accuracy of the output they provide as input for the LP or MIP model, can come into play.
6. *Other differences between calculations under perfect and finite precision*. The previous list of items is not comprehensive. In general, the practitioner should always assess how a concept translates from the perfect precision of the mathematical description to the corresponding finite precision representation. For example, under finite precision, addition and multiplication are no longer associative, and multiplication is no longer distributive over addition. Although the differences are small, they can be sufficient in the presence of ill-conditioning to yield significant differences in the final computed solution. This also explains why one can get different results within a program when applying different levels of compiler optimization to the source code, as simply reordering a calculation can yield slightly different results.

Equation (12) illustrates how round-off errors from any of these sources can be magnified when the LP or MIP has ill-conditioned basis matrices. Round-off errors on the order of machine precision (10^{-16} for IEEE double precision) arising from finite precision can be magnified to much larger values. For example, if a perturbation to the data is on the order of

10^{-16} and the condition number of one or more basis matrices is on the order of 10^{12}, then round-off errors as large as $10^{-16} \cdot 10^{12} = 10^{-4}$ can creep into the calculations, and linear programming algorithms may have difficulty distinguishing legitimate values from round-off error. The next section discusses this in more detail.

2.3. Optimizer Tolerances

State-of-the-art simplex algorithm implementations use tolerances to distinguish round-off error from meaningfully calculated values. The feasibility and optimality tolerances are particularly important. The feasibility tolerance provides the maximum acceptable bound violation for a basic variable for which the associated basic solution is considered feasible. Note that the feasibility tolerance does not constitute a relaxation of variable bounds. The variable bounds remain unchanged, but bound violations of the computed basic variable values within the feasibility tolerance are not considered infeasible. The optimality tolerance can be viewed as the tolerance on dual feasibility for the LP. With an objective to be minimized as in (P_{LP}), the optimality tolerance specifies the amount the nonbasic reduced costs can drop below 0 without being considered a potential entering basic variable. Equivalently, the optimality tolerance specifies the amount the slack variables to the dual model of (P_{LP}) can violate their lower bound of 0 without being considered infeasible.

Barrier algorithm implementations use a different set of tolerances. Unlike simplex algorithms, barrier algorithms are based on convergence, and they use a tolerance based on the complementary slackness of the primal and dual iterates to concurrently assess primal and dual feasibility. This tutorial focuses primarily on simplex algorithms, so it does not discuss barrier tolerances further.

MIPs consist of the linear program (P_{LP}) with additional integrality restrictions on some or all of the variables. State-of-the-art MIP optimizers use the branch-and-cut algorithm, which consists of the more fundamental branch-and-bound algorithm but can also add cuts at the nodes, including the root. Either algorithm solves various LP subproblems as well, typically by the simplex method since it currently has better support for restarts than the barrier and other interior point algorithms. Therefore, the choice of feasibility and optimality tolerances also influences MIP optimizations. In addition, the integrality tolerance specifies the amount by which a variable solution value in an LP subproblem can violate its integrality restriction without being considered fractional by the optimizer. In other words, a variable with fractional solution value within this tolerance is not eligible for branching. Although the integrality tolerance can play a significant role in addressing certain types of ill-conditioned or numerically unstable models (specifically, those with binary variables multiplied by so-called big-M coefficients), the simplex feasibility and optimality tolerances are more important because the LP subproblem solves form the foundation of the branch-and-cut algorithm.

CPLEX uses default feasibility and optimality tolerances of 10^{-6}. Most other state-of-the-art optimizers use the same or similar values. These tolerances are absolute, rather than relative. This means that they remain the same for models with data values on the order of 10^0 or much larger, e.g., 10^7. Section 2.1 illustrated how absolute round-off error potentially increases when larger values are used. A number on the order of 10^7 has 16 digits of accuracy, which means its double precision representation can incur round-off on the order of $10^7 \cdot 10^{-16} = 10^{-9}$. A moderately large basis condition number, or additional round-off errors from arithmetic operations, could quickly increase the round-off errors above CPLEX's default feasibility and optimality tolerance settings of 10^{-6}. Although using relative tolerances (i.e., dynamically scaling the feasibility and optimality tolerances based on the primal and dual values against which they are compared) offers some advantages in this regard, it has some drawbacks as well. CPLEX and most other LP solvers instead try to improve the scaling of the model so that the data values are closer to 10^0, making the default tolerances more appropriate. Therefore, the practitioner should be cautious using data values of 10^{10} and above when CPLEX's feasibility and optimality tolerances are at default values or smaller, as the round-off error arising just from representing these numbers in IEEE double precision could exceed the tolerances. This is particularly true if such values are mixed with other data values of 10^0 or less.

More generally, the practitioner should try to avoid situations in which cumulative round-off error in the model data or subsequently calculated values can exceed the tolerances that affect the optimizer's algorithmic decisions. Let us first consider an example regarding data values and tolerances. Very small numerical values force the algorithm to decide whether those smaller values are real or due to round-off error, and the optimizer's decisions can depend on the coordinate system (i.e., basis) with which it views the model. Consider the following feasibility problem, drawn from Klotz and Newman [18]:

$$\begin{aligned} c_1\colon\ & -x_1 + 24x_2 \le 21, \\ & -\infty < x_1 \le 3, \\ & x_2 \ge 1.00000008. \end{aligned} \tag{13}$$

The primal simplex method concludes infeasibility during the presolve:

CPLEX Iteration Log #1, Primal Simplex Method, Default Settings

```
CPLEX > primopt
Infeasibility row ‘‘c1’’: 0 ≤ -1.92e-06.
Presolve time = 0.00 sec.
Presolve - Infeasible.
```

Turning presolve off results in a similar conclusion from the primal simplex during the first iteration.

CPLEX Iteration Log #2, Primal Simplex Method, Presolve Disabled

```
Primal simplex - Infeasible: Infeasibility = 1.9199999990e-06
Solution time = 0.00 sec. Iterations = 0 (0)
CPLEX > display solution reduced -
Variable Name             Reduced Cost
x1                           -1.000000
x2                           24.000000
CPLEX > display solution slacks -
Constraint Name           Slack Value
slack c1                   -0.000002**
CPLEX > display solution basis variables -
There are no basic variables in the given range.
CPLEX > display solution basis slack -
Constraint ‘‘c1’’ is basic.
```

The asterisks on the slack value for constraint c1 signify that the solution violates the slack's lower bound of 0.

These two runs both constitute correct outcomes. In Iteration Log #1, CPLEX's presolve uses the variable bounds and constraint coefficients to calculate that the minimum possible value for the left-hand side of constraint c1 is $-3 + 24 \cdot 1.00000008 = 21 + 1.92 \cdot 10^{-6}$. This means that the left-hand side must exceed the right-hand side, and by a value of more than that of CPLEX's default feasibility tolerance of 10^{-6}. Iteration Log #2 shows that with presolve off, CPLEX begins the primal simplex method with the slack on constraint c1 in the basis, and the variables x_1 and x_2 at their respective bounds of 3 and 1.00000008. Given this basis, the reduced costs, i.e., the optimality criterion from phase I of the primal simplex method, indicate that there is no way to remove the infeasibility, so CPLEX declares the model infeasible. Note that most optimizers treat variable bound constraints separately from general linear constraints, and that a negative reduced cost on a variable at its upper bound such as x_1 indicates that decreasing that variable from its upper bound cannot decrease the objective. Now, suppose we run the primal simplex method with a starting basis of x_2, the slack variable nonbasic at its lower bound, and x_1 nonbasic at its upper bound. The resulting basic solution of $x_1 = 3$, $x_2 = 1$, slack on c1 $= 0$ satisfies constraint c1 exactly. The variable x_2 does not satisfy its lower bound of 1.00000008 exactly, but the violation is less than many optimizers' default feasibility tolerance of 10^{-6}. So, with this starting basis, an optimizer could declare the model feasible (and hence optimal, because the model has no objective function):

CPLEX Iteration Log #3, Primal Simplex Method, Starting Basis of x_2

```
Primal simplex - Optimal: Objective  = 0.0000000000e+00
Solution time = 0.00 sec. Iterations = 0 (0)
```

In this example, had we set the feasibility tolerance to 10^{-9}, we would have obtained consistent results with respect to both bases because the data do not possess values smaller than the relevant algorithm tolerance. Although the value of 0.00000008 is input data, this small numerical value could have just as easily been created during the course of the execution of the algorithm. This example illustrates the importance of verifying that the optimizer tolerances properly distinguish legitimate values from those arising from round-off error. When a model is on the edge of feasibility, different bases may prove feasibility or infeasibility relative to the optimizer's tolerances. Rather than relying on the optimizer to make such important decisions, the practitioner should ensure that the optimizer's tolerances are suitably set to reflect the valid precision of the data values in the model. In the example we just examined, one should first determine whether the lower bound on x_2 is really 1.00000008, or if, in fact, the fractional part is round-off error in the data calculation and the correct lower bound is 1.0. If the former holds, the practitioner should set the optimizer's feasibility and optimality tolerances to values smaller than 0.00000008. If the latter holds, the practitioner should change the lower bound to its correct value of 1.0 in the model. In this particular example, the practitioner may be inclined to deduce that the correct value for the lower bound on x_2 is 1.0, because all other data in the instance are integers. More generally, examination of the possible round-off error associated with the procedures used to calculate the input data may help to distinguish round-off error from meaningful values.

One particularly problematic source of round-off errors in the data involves the conversion of single precision values to their double precision counterparts used by most optimizers. Machine precision for an IEEE single precision value is $6 \cdot 10^{-8}$, which is almost as large as many of the important default optimizer tolerances, such as CPLEX's default feasibility and optimality tolerances of 10^{-6}. So, simply *representing* a data value in single precision can introduce round-off error of at least $6 \cdot 10^{-8}$, and additional single precision data calculations can increase the round-off error above the aforementioned optimizer tolerances. Hence, the optimizer may subsequently make decisions based on round-off error. Computing the data in double precision from the start avoids this problem. If that is not possible, setting the optimizer tolerances to values that exceed the largest round-off error associated with the conversion from single to double precision provides an alternative.

The previous example with the small problem (13) illustrated the importance of distinguishing legitimate values from round-off error in the LP problem data and how the practitioner can leverage his knowledge of the model to make better decisions than the optimizer about this distinction. But §2.2 illustrated how the condition number of the basis matrix can magnify round-off errors in the data to significantly larger values. Therefore, the basis condition number plays an important role in the distinction of round-off error from legitimate data values and the appropriate optimizer tolerances to set. Given the typical machine precision of 10^{-16} for double precision calculations and Equation (12) that defines the condition number, a condition number of 10^{10} provides an important threshold value. Given CPLEX's default feasibility and optimality tolerances of 10^{-6}, condition numbers of 10^{10} or greater imply a level of ill-conditioning that could cause the implementation of the algorithm to make decisions based on round-off error. Because (12) is an inequality, a condition number of 10^{10} does not guarantee this will happen, but it provides guidance as to when ill-conditioning is likely to be problematic. If condition numbers significantly exceed this threshold value, the likelihood of algorithmic decisions based on round-off error increases.

Figure 1 illustrated how double precision calculations involving numbers with orders of magnitude larger than 1 can introduce absolute round-off larger than the machine precision. Therefore, a basis condition number less than the aforementioned threshold of 10^{10} can magnify this error above the default feasibility and optimality tolerance of 10^{-6}. Note that because machine precision defines the smallest value that distinguishes two numbers, calculations involving numbers with orders of magnitude smaller than 1 still possess round-off errors at the machine precision level. Whereas the relative round-off error remains at machine precision, the absolute round-off error does not. Most state-of-the-art optimizers use absolute tolerances for assessing feasibility and optimality. To address this lack of relativity, they typically scale linear programs to try to keep the round-off error associated with double precision calculations close to the machine precision. Nonetheless, the practitioner can benefit from formulating LPs that are well scaled, avoiding mixtures of large and small coefficients that can introduce round-off errors significantly larger than machine precision. If this is not possible, the practitioner may need to consider solving the model with larger feasibility or optimality tolerances than the aforementioned defaults of 10^{-6}.

2.4. Numerical Stability of Algorithms

Although numerical instability and ill-conditioning are sometimes viewed as equivalent, they are not the same. Ill-conditioning can occur under perfect precision calculations and depends on the model. Finite precision calculations simply introduce numerous sources of potential perturbations to the input of an ill-conditioned model (see §2.2) that can result in larger changes to the output. By contrast, numerical instability involves procedures and algorithms implemented in finite precision. Formally, a procedure is numerically stable if its backward error analysis results in small, bounded errors on all data instances; i.e., if a small, bounded perturbation to the model would make the computed (under finite precision) solution to the unperturbed model an exact solution of the perturbed model. By contrast, the forward error consists of the actual error in the computed result under finite precision compared to the result under perfect precision. Mathematically, when computing $y = f(x)$, the absolute forward error is $\Delta y = |fl(f(x)) - f(x)|$. Note that this quantity cannot be computed under finite precision, since $f(x)$ is the result under perfect precision. Rather, one mathematically derives $fl(f(x))$ for a specific finite precision floating point representation, then compares it to $f(x)$ to assess the round-off error. Chapter 1 of Higham [15] provides some detailed examples of this, including one in which equivalent formulae to calculate the sample variance yield significantly different levels of forward error. Similarly, the absolute backward error is given by Δx: $f(x + \Delta x) = fl(f(x))$. Practically, calculations that can increase the round-off error relative to the error in the inputs can be considered numerically unstable. Here is a list of potentially numerically unstable calculations. The last two items come from a longer list of guidelines for designing numerically stable algorithms found in Higham [15].

1. *Performing arithmetic operations on numbers of dramatically different orders of magnitude*. This was shown by Figures 2 and 3. This issue is even more problematic when dividing a smaller number into a much larger one. This can be remedied when an equivalent calculation could avoid the problematic division. The flexibility of the pivot ordering when calculating the LU factorization for basis matrices in simplex algorithms illustrates this and is discussed in greater detail subsequently.

2. *Algorithms that rely on or create ill-conditioned subproblems*. Algorithms often create and solve subproblems or systems of equations. If the underlying logic or theory, or simply the nature of the finite precision calculations, that create them can generate ill-conditioned subproblems, the main algorithm can become numerically unstable. For example, as a cutting plane algorithm that uses Gomory cuts nears convergence, the cuts tend to become almost parallel (Sherali and Driscoll [29]). Almost parallel rows are nearly linearly dependent, resulting in high basis matrix condition numbers (as is described in more detail later) when the simplex method is used to solve the subproblems with the additional cuts. For this reason, branch-and-cut methods that add Gomory cuts must take precautions to avoid adding nearly parallel cuts.

3. *Ill-conditioned transformations of the problem*. Some algorithms perform transformations of the problem to create a more manageable problem. If these transformations are ill-conditioned, they can introduce numerical instability into this step of the algorithm. For example, with a linear transformation and linear constraints, multiplying by an ill-conditioned matrix introduces potential numerical instability.

4. *Calculations involving large intermediate values compared to final solution values*. Small relative round-off error for large intermediate values is significantly larger relative to the final computed values. Dividing smaller numbers into larger numbers is one example of this phenomenon.

Let us now consider numerical stability of simplex algorithms for LPs. The linear solves involving the basis matrix and the associated LU factorization are the foundation upon which simplex algorithm implementations are built. The remaining steps, such as the computation of the reduced costs and the ratio tests to preserve primal or dual feasibility, rely on the results of these solves. So numerically stable calculations for these linear solves are essential. The following example, also drawn from Higham [15], illustrates the stability benefits provided by the permuting of rows allowed with partial pivoting. We consider two matrices, $\tilde{A}$ and the corresponding matrix A with the rows of $\tilde{A}$ permuted, and then compare their factorizations $\tilde{A} = \tilde{L}\tilde{U}$ and $A = LU$.

$$\tilde{A} = \begin{pmatrix} \epsilon & -1 \\ 1 & 1 \end{pmatrix} = \overbrace{\begin{pmatrix} 1 & 0 \\ \dfrac{1}{\epsilon} & 1 \end{pmatrix}}^{\tilde{L}} \overbrace{\begin{pmatrix} \epsilon & -1 \\ 0 & 1 + \dfrac{1}{\epsilon} \end{pmatrix}}^{\tilde{U}}. \tag{14}$$

$$A = \begin{pmatrix} 1 & 1 \\ \epsilon & -1 \end{pmatrix} = \overbrace{\begin{pmatrix} 1 & 0 \\ \epsilon & 1 \end{pmatrix}}^{L} \overbrace{\begin{pmatrix} 1 & 1 \\ 0 & -(1 + \epsilon) \end{pmatrix}}^{U}. \tag{15}$$

Since permuting rows does not affect matrix norms, $\tilde{A}$ and A have the same condition number. To compute that condition number, note that

$$\tilde{A}^{-1} = \begin{pmatrix} \dfrac{1}{\epsilon + 1} & \dfrac{1}{\epsilon + 1} \\ -\dfrac{1}{\epsilon + 1} & \dfrac{\epsilon}{\epsilon + 1} \end{pmatrix}. \tag{16}$$

Therefore, the condition number of either $\tilde{A}$ or A using either the 1 or ∞ matrix norm (i.e., maximum absolute column or row sum, respectively) in (12) is $4/(1 + \epsilon)$, so $\tilde{A}$ and A are well conditioned for all small positive values of ϵ (and, in fact, are well conditioned for all values of

ϵ of modest order of magnitude except those close to -1). However, as ϵ drops below the machine precision of 10^{-16}, $\tilde{u}_{22} = 1 + 1/\epsilon$ moves above 10^{16}; its 16 digits of accuracy are on the order of 10^1 or higher. Thus, $fl(\tilde{u}_{22}) = 1/\epsilon$ and, even if we assume the calculation of $1/\epsilon$ is performed exactly,

$$\tilde{A} - fl(\tilde{L})fl(\tilde{U}) = \begin{pmatrix} \epsilon & -1 \\ 1 & 1 \end{pmatrix} - \begin{pmatrix} 1 & 0 \\ \frac{1}{\epsilon} & 1 \end{pmatrix} \begin{pmatrix} \epsilon & -1 \\ 0 & \frac{1}{\epsilon} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}. \tag{17}$$

By contrast, in the LU factorization of A in (15), no divisions by ϵ occur, which, in turn, results in no mixtures of large and small numbers in the factorization, and no resulting loss of accuracy in the calculation. Another interpretation of this is that although $\tilde{A}$ and A are both well conditioned, the condition numbers of $\tilde{L}$ and $\tilde{U}$ are much larger than the condition numbers of L and U for small values of ϵ; $\tilde{L}$ and $\tilde{U}$ both have terms of $1/\epsilon$, which result in large matrix norms. Thus, even if their inverses have modest norms, (12) indicates that these two matrices have large condition numbers. The large matrix norms of $\tilde{L}$ and $\tilde{U}$ relative to $\tilde{A}$ illustrate the potential numerical instability associated with calculations involving intermediate values much larger than final solution values. In fact, this comparison of the stability of regular pivoting with the row interchange capability of partial pivoting also illustrates three other items in the aforementioned list of potentially numerically unstable calculations, namely, arithmetic operations with numbers of dramatically different orders of magnitude, smaller numbers being divided into larger numbers, and linear transformations involving ill-conditioned matrices.

2.5. Alternate Measurements of Conditioning of a Square Matrix

Although (12) provides an exact, quantitative measure of the condition number of a square matrix, other metrics exist. Familiarity with these different metrics can help the practitioner effectively locate the constraints or variables that cause the high condition number.

Gastinel and Kahan (Kahan [17]) established an inverse relationship between the condition number of a square matrix and its distance to singularity. Formally, for a square matrix B and an associated perturbation ΔB, if the distance to singularity is defined by

$$\operatorname{dist}(B) := \min_{\Delta B} \left\{ \frac{\|\Delta B\|}{\|B\|} \colon B + \Delta B \text{ singular} \right\}, \tag{18}$$

then

$$\operatorname{dist}(B) = \frac{1}{\kappa(B)}. \tag{19}$$

This result measures distance to singularity in terms of a perturbation matrix ΔB applied to the elements of B. A related measure of distance to singularity involves linear combinations of the rows or columns of B that results in a vector with a small norm. The above result also implies that a linear combination of the rows or columns of B that is close to zero implies ill-conditioning. In other words, for any matrix B with $\|B\| \geq 1$, B is ill-conditioned if

$$B\lambda = v, \tag{20}$$

$$\|v\| < \epsilon, \tag{21}$$

$$\|\lambda\| \gg \epsilon. \tag{22}$$

Specifically, since $B\lambda = v$, $\lambda = B^{-1}v$, application of the Cauchy–Schwarz inequality gives

$$\frac{\|\lambda\|}{\|v\|} \leq \|B^{-1}\|. \tag{23}$$

Since $\|\lambda\| \gg \epsilon$ and $\|B\| \geq 1$, multiplying the left-hand side of (23) by $\|B\|$ yields

$$\kappa(B) = \|B\|\|B^{-1}\| \geq \frac{\|\lambda\|}{\|v\|}. \tag{24}$$

As ϵ approaches 0, this lower bound on the condition number approaches infinity. Thus, the vector λ can be considered a certificate of ill-conditioning of B, and the columns of B corresponding to zero elements of λ can be excluded as potential sources of the ill-conditioning. The same result holds for linear combinations of the rows of B.

A special case of the perturbation in (18) occurs when ΔB is a nonzero multiple of the identity matrix. This special case provides another metric for measuring ill-conditioning, namely, eigenvalues. Recall that a nonzero eigenvalue λ and eigenvector x for the matrix B satisfy

$$Bx = \lambda x. \tag{25}$$

Equivalently, $(B - \lambda I)x = 0$. Thus, if λ is small, $\|\lambda I\|$ is small as well. If $\|\lambda I\| \ll \|B\|$, then (18) and (19) imply that $\kappa(B) \geq 1/\lambda$. In other words, as long as $\|B\| \geq 1$, a small eigenvalue implies that B is ill-conditioned. Furthermore, multiplying (25) by B^{-1} and rearranging, $B^{-1}x = (1/\lambda)x$. Therefore, for any λ that is an eigenvalue of B, $1/\lambda$ is an eigenvalue of B^{-1}.

The derivation of (12) was for arbitrary matrix norm. It was illustrated with the ∞-norm, which is the maximum absolute row sum of the matrix. The matrix 1-norm is the maximum absolute column sum of the matrix. The matrix 2-norm is the maximum absolute eigenvalue of B^TB. Let $\kappa_p(B)$ denote the condition number of the matrix B using the p-norm in equation (12). When using the matrix 2-norm for the condition number of B, the inverse relationship between the eigenvalues of B^TB and its inverse imply that $\kappa_2(B)$ is the ratio of the maximum to the minimum eigenvalue of B^TB. When B is a normal matrix (i.e., $B^TB = BB^T$), then the eigenvalues of B^TB are the same as those of B, so the condition number of B is the same ratio.

Methods for computing eigenvalues is a lengthy topic that is beyond the scope of this tutorial. However, mathematical toolkits such as MATLAB (MathWorks [21]) can compute them for sparse matrices of nontrivial size. And most state-of-the-art linear, integer, and also nonlinear programming solvers now offer interfaces to MATLAB, either directly as part of the software package or indirectly (at additional cost) through third-party vendors such as TOMLAB Optimization (http://www.tomopt.com).

Basis matrices in LPs are asymmetric, which means they can have complex eigenvalues. Although they typically are not normal matrices, the condition number metrics in (19) or (20) imply that B is ill-conditioned when B^TB is. Nonetheless, a large 2-norm for B provides less insight into the submatrices of B responsible for the ill-conditioning. Therefore, using eigenvalues to assess ill-conditioning of basis matrices used by simplex algorithms has limited practicality. However, for the barrier algorithm and related interior point methods that solve scaled symmetric square linear systems involving the constraint matrix multiplied by its transpose, the eigenvalues are real and can provide useful information.

So far, this section has discussed quantities that provide an exact measure of the potential ill-conditioning. Approximate metrics are also possible, and we consider one involving the LU factorization of the basis matrix. In almost all practical models, L and U are sparse, as is the basis matrix. As long as numerically stable threshold pivoting methods are used to compute the factorization, L has a low condition number; L is lower triangular and has diagonal elements all equal to 1. The elements below the diagonal are bounded by the reciprocal of the threshold pivoting tolerance (called the Markowitz tolerance in CPLEX). For the default tolerance in most LP solvers, this means the absolute value of the off-diagonal elements is bounded by 100. Similarly, L^{-1} is also lower triangular and has diagonal elements of 1. Its off-diagonal elements equal the negative of the corresponding elements of L. Combining this with the sparsity, $\|L\|$ and $\|L^{-1}\|$ consist of the sum of a small number of absolute values bounded by 100. Hence, they are quite modest, as is the condition number derived from their product. Hence, any ill-conditioning of the basis matrix is reflected in U. U is upper triangular and can have positive or negative diagonal elements. We can express $U = D\hat{U}$, where D is a diagonal matrix with elements of absolute values of the diagonals of U. The diagonals of $\hat{U}$ are therefore ± 1. $\hat{U}$ is the upper triangular matrix obtained by rescaling the rows of U by the reciprocals of the diagonals in D. D can now be used to assess ill-conditioning in U. The eigenvalues of a diagonal matrix are the diagonal elements. Thus, a large ratio of maximum to minimum diagonal element of D indicates that the smallest eigenvalue $d_{\min} \ll \|D\|$, which, by (18) and (19), implies that $\kappa(D) \geq 1/d_{\min}$. If D is ill-conditioned in this way, then the associated LU factorization involves an ill-conditioned transformation, one of the potentially unstable calculations listed in §2.4. Thus, the largest and smallest absolute elements of the diagonals of U provide a proxy for the condition number of the basis matrix associated with the LU factorization.

Unfortunately, many of the state-of-the-art LP and MIP solvers do not provide information about the diagonals of U in computed factorizations. One exception is MINOS (Murtagh and Saunders [23]), which is a general-purpose nonlinear solver that also has a very numerically stable primal simplex code. The potential for the nonlinear algorithms to generate numerically challenging LP subproblems, combined with a user base including many experienced math programming practitioners, prompted its designers to provide higher levels of debugging and diagnostic output.

3. Identification and Treatment of Symptoms of Ill-Conditioning and Numerical Instability

Regarding ill-conditioning and numerical instability, the practitioner should take care to distinguish between symptoms and causes. The symptoms help the practitioner determine when ill-conditioning or numerical instability affects optimizer performance (in terms of either run time or accuracy of the final solution). Most optimizers have parameter settings that can improve optimizer performance in such cases. However, relying on parameter settings to resolve issues with truly ill-conditioned problems is less robust than eliminating the underlying causes. Optimizer parameter settings can reduce or limit the adverse effects, but as long as the settings affect the optimization algorithm rather than the model being solved, they cannot remove the root cause of the ill-conditioning. So, although the parameters discussed in this section can improve performance on ill-conditioned problems, one should typically view changing parameters as a short-term remedy and consider the more long-term approaches to identifying and removing the causes of ill-conditioning described subsequently in this tutorial.

3.1. Identification of Symptoms

Reiterating, all optimizer output is illustrated using IBM ILOG CPLEX. Other state-of-the-art optimizers may offer similar functionality. CPLEX has numerous utilities to help its users identify symptoms of numerical instability or ill-conditioning. We illustrate using the interactive CPLEX optimizer, but most of these features are available from CPLEX's programming application programming interfaces (APIs) as well. CPLEX offers the "display problem stats" command to help assess the range of coefficients in the problem data. Here is some sample output from the LP model `ns1687037`, publicly available at Hans Mittelmann's Benchmarks for Optimization Software website (http://plato.asu.edu/ftp/lptestset/). These problem statistics show a wide range of coefficients in the constraint matrix and right-hand side vectors, indicating the potential for ill-conditioning, or simply numerical instability as a result of unnecessarily large round-off errors.

CPLEX Problem Statistics, ns1687037

```
 Variables            :    43749 [Nneg: 36001, Box: 874, Free: 6874]
 Objective nonzeros   :    24000
 Linear constraints   :    50622 [Greater: 38622, Equal: 12000]
   Nonzeros           :  1406739
   RHS nonzeros       :    24000

 Variables            :  Min LB:  0.000000      Max UB:  3.000000
 Objective nonzeros   :     Min:  1.000000         Max:  100.0000

 Linear constraints   :
   Nonzeros           :     Min:  1.987766e-08     Max:  1364210.
   RHS nonzeros       :     Min:  0.0005000000     Max:  5.030775e+07
```

The problem statistics provide information about the model before the optimization algorithm has started. Many models with a wide range of coefficients as in the output above solve without any performance problems or limited accuracy in the computed optimal solution. Nonetheless, practitioners with such models can benefit from assessing optimizer performance for wasted computations and solutions for residuals that could compromise their meaning in the system being modeled. To assess performance, look at the iteration log of the run. In particular, numerical performance problems typically manifest themselves with frequent basis refactorizations after 10 or fewer simplex iterations, loss of primal or dual feasibility after it had previously been attained, and messages indicating that CPLEX's internal logic selected more numerically conservative settings than the defaults. The following iteration log, from a dual simplex run on ns1687037 with CPLEX 12.6 and default settings, illustrates several of these indicators.

CPLEX Iteration Log #4: Loss of Feasibility After Basis Refactorization, ns1687037

```
Iteration: 674222 Dual objective     = 913.318204
Iteration: 674228 Dual objective     = 913.318204
Iteration: 674239 Dual objective     = 913.318204
Iteration: 674246 Dual objective     = 913.318205
Iteration: 674254 Dual objective     = 913.318205
Iteration: 674256 Dual objective     = 913.318205
Iteration: 674258 Dual objective     = 913.318205
Removing perturbation.
Iteration: 674259 Scaled dual infeas = 12123.146176
Iteration: 674772 Scaled dual infeas = 595.276887
Elapsed time = 16959.32 sec. (6667412.82 ticks, 674876 iterations)
Iteration: 674876 Scaled dual infeas = 0.636846
                  ...
Elapsed time = 17138.76 sec. (6737439.02 ticks, 681930 iterations)
Iteration: 681949 Scaled dual infeas = 0.000002
Iteration: 682053 Scaled dual infeas = 0.000001
                  ...
Iteration: 682542 Scaled dual infeas = 0.000000
Iteration: 682624 Dual objective     = -19559.930294
Iteration: 682896 Dual objective     = -18109.597465
Elapsed time = 17160.88 sec. (6747443.75 ticks, 682941 iterations)
```

Each log line that prints the objective corresponds to a basis refactorization. In the first seven such lines, we see refactorizations occurring every 2 to 11 iterations. This indicates numerical troubles; CPLEX normally needs to refactorize no more than once every 100 iterations. After

CPLEX removes the small perturbations designed to address dual degeneracy, we see a massive loss of dual feasibility at iteration 674259. CPLEX must now iterate to restore dual feasibility, which requires more than 8000 iterations and three minutes of wasted effort. Even worse, since the dual simplex algorithm is maximizing in this case, iteration 682624 reveals that restoring feasibility has substantially degraded the objective of the dual feasible solution that is obtained after iterating to remove the infeasibilities. So additional iterations and time are needed simply to regain the previous level of dual objective. Regarding CPLEX's automatic use of more conservative settings, the following segment of the iteration log from the same run illustrates. In this case, the Markowitz tolerance was increased to its maximum value to invoke the most conservative threshold pivoting strategy during the basis factorization operations. Note that iterations 783543–783556 preceding this decision involved basis factorizations every three to seven iterations. That indicator of numerical difficulties influenced the decision to use a more conservative approach during the basis factorization.

CPLEX Iteration Log #5: Increase of Markowitz Tolerance, ns1687037

```
Iteration: 783543 Dual objective = 3.635840
Iteration: 783548 Dual objective = 3.635840
Iteration: 783550 Dual objective = 3.635840
Iteration: 783553 Dual objective = 3.635840
Iteration: 783556 Dual objective = 3.635840
Removing shift (209).
Markowitz threshold set to 0.99999
Iteration: 783558 Dual objective = 3.635695
```

In some cases, numerical issues like those described in Iteration Logs #4 and #5 occur for a subset of bases, and the optimal basis CPLEX finds does not have any problems. In that case, the final solution is accurate, but the practitioner has an opportunity to reduce optimization run time by addressing these issues, thus removing wasted computational effort. However, in other cases, these same problems may prevent the optimizer from finding an optimal solution or result in an optimal solution of limited accuracy. CPLEX provides extensive measures of solution quality to help the practitioner assess the accuracy of computed solutions. Solution quality also helps the practitioner identify potential aspects of the model that are involved in the numerical problems. Here is the solution quality output from the same run as the above logs, for which CPLEX declared an optimal solution after more than five hours of run time. A detailed discussion of each category of the solution quality follows subsequently.

Optimal Basis Solution Quality, ns1687037

```
Max. unscaled (scaled) bound infeas.        = 8.39528e-07 (8.39528e-07)
Max. unscaled (scaled) reduced-cost infeas. = 2.31959e-08 (2.31959e-08)
Max. unscaled (scaled) Ax - b resid.        = 3.51461e-07 (1.16886e-12)
Max. unscaled (scaled) c - B'pi resid.      = 1.18561e-13 (1.18561e-13)
Max. unscaled (scaled) |x|                  = 24139.1 (24139.1)
Max. unscaled (scaled) |slack|              = 48278.2 (48278.2)
Max. unscaled (scaled) |pi|                 = 76.2637 (76.2637)
Max. unscaled (scaled) |red-cost|           = 100 (100)
Condition number of scaled basis            = 2.2e+12
```

Note that when CPLEX terminates because of an optimal solution or some other termination criterion, it refactorizes the basis. Therefore, solution quality displays correspond to solutions based on a fresh factorization. The first four lines provide measures of the accuracy of the computed primal and dual solutions; the next four lines provide information about the solution values.

1. *Bound infeasibilities.* Since the simplex method always sets nonbasic variables to their bounds (and any superbasic values lie between bounds), only basic variables can register bound infeasibilities. Immediately after a basis factorization, basic variables are recalculated based on a linear solve of (5) using the freshly computed LU factorization. At iterations between factorizations, basic variables are updated using the representation of the incoming nonbasic column relative to the current basis. This column also involves a linear solve with the LU factorization, which includes the updates that account for the pivots since the last basis refactorization. But solution quality is displayed for basic variables calculated from a fresh factorization. Bound violations for a basic feasible solution lie within CPLEX's feasibility tolerance (with the rare exception of unscaled infeasibilities, discussed later). However, bound infeasibilities close to the feasibility tolerance may indicate the presence of ill-conditioning. They could also indicate that more conservative parameter settings regarding stability are needed or the presence of relatively large numerical values in the data. If the absolute round-off error in the associated simplex method calculations is too large for the selected feasibility tolerance (which is also absolute), a larger feasibility tolerance may be appropriate. In the solution quality output above, the optimal solution found is feasible for CPLEX's default feasibility tolerance of 10^{-6}, but it would not be for a feasibility tolerance of 10^{-7}. By itself, this is not an indication of a problem; CPLEX has computed a solution within the specified tolerances. But an accurately computed solution would typically have bound infeasibilities several orders of magnitude below the feasibility tolerance.

2. *Reduced cost infeasibilities.* Reduced cost infeasibilities measure violations of dual feasibility. In the dual of P_{LP}, dual variables associated with the constraints are all free, so bound violations can only occur in the dual slack variables, which correspond to the reduced costs in the primal simplex method. For an optimal basis to P_{LP}, reduced cost infeasibilities measure the reduced costs that fall below zero but have absolute values less than the optimality tolerance and hence do not alter the declaration of optimality. For nonoptimal solutions, reduced cost infeasibilities measure the most negative reduced costs. In the solution quality output above, changing the optimality tolerance from its default of 10^{-6} to 10^{-8} or less would prompt CPLEX to perform additional iterations.

3. *Residuals on $Ax-b$.* In finite precision computing implementations, the simplex algorithms solve the linear system $A_Bx_B = b - A_Nx_N$ after each refactorization of A_B. Let $\hat{x} = fl(x\colon A_Bx_B = b - A_Nx_N)$, i.e., the computed solution to this linear system. The residuals on $Ax-b$ are the finite precision calculation of $A\hat{x}-b$. Although the finite precision nature of the additions, subtractions, and multiplications in this calculation can contribute to these residuals, most of the residual values typically result from the difference between $\hat{x}$, the calculated solution, and the solution x that would be obtained under perfect precision. Residual values that exceed the feasibility tolerance typically arise from large basis condition numbers, or large right-hand-side or variable bound values that lead to nontrivial absolute round-off error in the calculations.

4. *Residuals on $c - B'pi$.* Whereas the residuals on $Ax-b$ provide an assessment of the accuracy of primal solutions, the residuals on $c - B'pi$ provide the corresponding assessment for dual solutions. In other words, (P_{LP}) has dual constraints $A^T\pi \le c$, and the residuals for the dual solution values are the residuals on the square linear system $A_B^T\pi = c_B$. Primal and dual simplex method implementations either explicitly or implicitly solve this system to calculate the dual variables subsequently used to calculate reduced costs. Residuals on this linear system that exceed the optimality tolerance (i.e., the feasibility tolerance for the dual problem that is implicitly present when invoking the primal or dual simplex algorithms) are an indication of ill-conditioning or large values in the objective coefficients that can lead to nontrivial absolute round-off error in the calculations.

5. *Primal and dual solution values.* Primal and dual solution values, including slacks and reduced costs, by themselves do not shed much light on sources of ill-conditioning or numerical instability. But the presence of extremely large values in any solution often can help the practitioner isolate the source of numerical problems. By starting with a small group of

the variables or constraints with such large values and working through, respectively, the intersecting constraints and variables involved, one can often locate the source of the problem. However, we see from the preceding solution quality output for the model `ns1687037` that the solution values are all reasonable and therefore have limited potential to aid the diagnosis the numerical difficulties with this model.

6. *Basis condition number*. This is an estimate of the basis condition number defined in (12). Computing the 1- or ∞-norm for the explicit inverse of a basis from P_{LP} requires m linear solves, which is roughly equal to $m/4$ simplex iterations. Although this is not computationally intractable, it can consume significant amounts of time on large models. Fortunately, accurate, inexpensive estimation procedures provide a proxy much faster. Given the aforementioned threshold of 10^{10} for basis condition numbers, the value of $2.2 \cdot 10^{12}$ reported here indicates that ill-conditioned basis matrices have potential to be involved in the loss of feasibility seen in the iteration logs for this model.

Items 3 and 4, the residuals on primal and dual feasibility, are the most important metrics for solution quality. If they exceed the feasibility and optimality tolerances, the simplex method implementations may make algorithmic decisions based on round-off error. Once that happens, inconsistent or unstable results can occur, regardless of whether the underlying cause is ill-conditioning in the model, numerical instability in the model (i.e., unnecessarily large or small coefficients that create larger amounts of round-off error), or the need for more numerically stable calculations in the algorithm. In such cases, particularly if the residuals are several orders of magnitude larger than the feasibility and optimality tolerances, some remedial action is advised. It may be as simple as using larger tolerances, provided that makes sense for the context of the model. For example, in a model that has numerous large data values on the order of 10^{10}, the 16 base-10 digits of accuracy with IEEE double precision imply round-off errors associated with simply representing the data to already be on the order of CPLEX's default feasibility and optimality tolerances of 10^{-6}. Additional calculations could easily magnify the errors, so in such cases larger feasibility and optimality tolerances make sense. The basis condition number is an important metric as well. However, keep in mind that it provides an upper bound on the maximum possible magnification of round-off error in the input in the computed output. It does not provide a calculation of the error in the output. If a large condition number is a problem, it typically manifests itself with large residuals on primal or dual feasibility. If the condition number is large but those residuals are significantly smaller than the feasibility and optimality tolerance, then, at least for the basis in question, the bound in (12) is probably not tight, and the ill-conditioning is not problematic.

When solving MIPs, the branch-and-cut algorithm used by state-of-the-art optimizers solves numerous LP relaxations and subproblems, arising from branching on variables, adding cuts, and various node heuristics. So, assessment of ill-conditioning involves more than just examining optimal basis condition numbers from the root node relaxation, the node or heuristic that found the optimal solution, or the fixed LP obtained from the MIP by fixing the integer variables at the final integer solution values. Proper assessment of the level of ill-conditioning requires statistics that summarize the overall profile of the run. To that end, CPLEX offers a MIP Kappa feature that accumulates such statistics. It samples LP subproblem optimal basis condition numbers during branch and cut, making use of the threshold of 10^{10} for basis condition numbers discussed in §2.3 to classify basis condition numbers. Basis condition numbers of less than 10^7 are classified as stable. Those with condition numbers in the interval $[10^7, 10^{10})$ are classified as suspicious. Although they typically are harmless, they can be problematic in the presence of other sources of nontrivial round-off errors (e.g., large values in the problem data). Bases with condition numbers ranging from $[10^{10}, 10^{13})$ are classified as unstable. They can be problematic just because of the round-off errors associated with the finite representation of the problem data. However, because the condition number provides a (possibly weak) bound on the error in the computed output rather than the value, CPLEX still frequently manages to solve MIPs with some basis condition numbers in this category, providing accurate solutions and showing no indications of an avoidable degradation in performance. Nonetheless, with condition numbers in this range, good solution quality and the absence of any signs of numerical performance problems do not guarantee a successful run; an unstable node LP could result in the pruning of a node that had a child node that contained a better solution than the one declared to be optimal. Bases with condition numbers of 10^{13} or higher are classified as ill-posed; a significant number of these are likely to cause problems for 64-bit finite precision calculations and CPLEX's default tolerances. The practitioner should consider larger tolerance settings, parameter settings that provide more accuracy in the calculations, or assessment of the ill-conditioning in the model to obtain satisfactory performance or accuracy of solutions.

Here are a few samples of output from CPLEX's MIP Kappa feature. We begin with output indicating no problems at all, with the result after solving stein45, from the MIPLIB 2003 test set (http://miplib.zib.de/miplib3/miplib3_cat.txt).

CPLEX MIP Kappa Output, stein45

```
Max condition number:                     5.6120e+03
Percentage (number) of stable bases:      100.00%    (66567)
Percentage (number) of suspicious bases:  0.00%      (0)
Percentage (number) of unstable bases:    0.00%      (0)
Percentage (number) of ill-posed bases:   0.00%      (0)
```

This output is ideal; all basis condition numbers fall well below the stable threshold of 10^7.

CPLEX also displays solution quality output for MIPs. Sample output for stein45 appears below. Since a solution to a MIP lacks a uniquely associated basis matrix, no measures of dual variable or reduced cost solution quality are available. Norms of primal variable values, bound violations, and associated residuals are provided analogously to the previously discussed LP solution quality. In addition, the integrality error describes the maximum amount a solution value for a variable violates its integrality requirement. CPLEX uses a default integrality tolerance of 10^{-5}. The slack bound error recomputes the slack or artificial variable values based on the structural variable values in the solution. Any resulting violations on the bounds of these slack variables that exceed the feasibility tolerance indicates potential for better solution quality by addressing ill-conditioning in the model. Note that the slack bound error differs from the primal residual solution error $Ax - b$. The former involves recomputing the slacks, whereas the latter computes residuals based on the structural and slack values obtained when the final integer solution was computed. These two metrics, along with the integrality error, are the most important measures of MIP solution quality. Here is the solution quality output for stein45.

CPLEX Optimal Solution Quality, stein45

```
MILP objective                              3.0000000000e+01
MILP solution norm |x| (Total, Max)         3.00000e+01 1.00000e+00
MILP solution error (Ax = b) (Total, Max)   0.00000e+00 0.00000e+00
MILP x bound error (Total, Max)             0.00000e+00 0.00000e+00
MILP x integrality error (Total, Max)       0.00000e+00 0.00000e+00
MILP slack bound error (Total, Max)         0.00000e+00 0.00000e+00
```

Given the previous MIP Kappa statistics, the perfect solution quality in this output is not particularly surprising. In addition, this is a combinatorial problem with a constraint matrix, objective, and right-hand-side nonzero coefficients all equal to 1 except for a single right-hand-side coefficient of 22.

By contrast, the results for neos-1225589, from the unstable test set for the MIPLIB 2010 test set (http://miplib.zib.de/miplib2010-unstable.php; see Koch et al. [20] for details) are much worse.

CPLEX MIP Kappa Output, neos-1225589

```
Max condition number:                     2.9345e+12
Percentage (number) of stable bases:      3.70%       (8)
Percentage (number) of suspicious bases:  25.46%      (55)
Percentage (number) of unstable bases:    70.83%      (153)
Percentage (number) of ill-posed bases:   0.00%       (0)
Attention level:                          0.215046
```

At first glance, this output looks highly problematic, with a vast majority of basis condition numbers classified as unstable. However, note that none are classified as ill-posed. Furthermore, CPLEX solves this model within a second with no signs in the node log of performance problems, and with good solution quality:

CPLEX Optimal Solution Quality, neos-1225589

```
MILP objective                              1.2310651918e+09
MILP solution norm |x| (Total, Max)         9.40754e+06   1.82096e+06
MILP solution error (Ax = b) (Total, Max)   5.42059e-10   1.16415e-10
MILP x bound error (Total, Max)             1.98270e-10   1.98270e-10
MILP x integrality error (Total, Max)       0.00000e+00   0.00000e+00
MILP slack bound error (Total, Max)         3.23137e-09   2.91038e-09
```

So, despite the presence of a significant number of optimal node LP bases with condition numbers in the range $[10^{10}, 10^{13})$, this model shows no signs of degradation in performance or solution quality.

In contrast to neos-1225589, the model momentum2, also from the MIPLIB 2010 unstable test set, has less than 1% of bases above the unstable threshold of 10^{10}, but more indications from the MIP Kappa and solution quality output for potential improved performance and accuracy. In this case, we see a few ill-posed bases and a higher maximum condition number. Also, the MILP slack bound indicator, which is similar to the residuals on $Ax - b$ for LPs, indicates violations on the order of the feasibility tolerance.

CPLEX Solution Quality and MIP Kappa Output, momentum2

```
MILP objective                              1.2314296334e+04
MILP solution norm |x| (Total, Max)         1.83089e+02   7.17892e+00
MILP solution error (Ax = b) (Total, Max)   1.55974e-13   1.77636e-15
MILP x bound error (Total, Max)             0.00000e+00   0.00000e+00
MILP x integrality error (Total, Max)       0.00000e+00   0.00000e+00
MILP slack bound error (Total, Max)         8.09544e-06   1.00000e-06

Branch-and-cut subproblem optimization:
Max condition number:                       3.3366e+17
Percentage (number) of stable bases:        89.49%    (27004)
Percentage (number) of suspicious bases:    9.68%     (2921)
Percentage (number) of unstable bases:      0.79%     (239)
Percentage (number) of ill-posed bases:     0.03%     (10)
Attention level:                            0.003676
```

These metrics indicate that although the percentage of models with suspicious or unstable condition numbers for neos-1225589 is much more than the corresponding percentage for momentum2, the latter model has potentially significant numerical problems, and the former does not.

3.2. Treatment of Symptoms

Although treating the symptoms of numerical problems may not be as robust as identifying and addressing the actual cause, it can provide easier and faster resolutions. CPLEX offers some parameters that can improve performance on numerically challenging models. Specifically, using the nondefault setting of 1 for the scaling parameter, although less effective on the majority of models, works well on models with a wide range of problem data coefficients. The numerical emphasis parameter, when enabled, invokes more numerically conservative approaches within the optimization algorithm. Although these can increase the computational effort associated with various operations, if a numerically challenging problem suffers from wasted computations (e.g., the extra simplex method iterations on ns1687037 arising from loss of dual feasibility in Iteration Log #4), then the time saved by eliminating the wasted computations usually outweighs the additional time required for the more precise calculations. The Markowitz tolerance controls the threshold pivoting used when calculating the LU factorization for the basis matrix. When performing a pivot to compute a column of L, threshold pivoting promotes a numerically stable calculation by ensuring that the pivot element is within a threshold of the maximum element in the pivot column. The Markowitz tolerance defines the threshold. For example, CPLEX's default Markowitz tolerance of 0.01 restricts the pivot column element selected to be within 0.01 of the maximum element. This limits the round-off errors that can accumulate from dividing smaller values into larger ones that was discussed previously in Equation (4). Increasing the Markowitz tolerance from this default value further reduces the round-off error in the individual pivots, yielding tighter bounds on the precision of the solution of the associated linear systems of equations. However, this can come at the significant computational cost of substantially increasing the nonzeros in the factorization.

3.3. Example: ns1687037

Let us now attempt to improve the performance and solution quality for the aforementioned model ns1687037. As discussed in §3.1, the primary problem was solution time. The quality of the optimal solution showed no residuals or infeasibilities larger than the default feasibility tolerances, although the unscaled primal residuals and primal bound infeasibilities were just barely below those tolerances. So, the primary challenge with this model involves improving solution time by avoiding the wasted iterations arising from loss of feasibility as illustrated in Iteration Log #4. The most serious concern arising from assessing the symptoms involves the wide range of nonzero coefficients in the constraint matrix, with a minimum of $1.987766 \cdot 10^{-8}$ and a maximum of 1364210. Therefore, a reasonable approach consists of simply setting the scaling parameter to 1, which is better suited for models with a wide range of coefficients. Indeed, doing so drops CPLEX's dual simplex run time from 21094 seconds to 903 seconds. Overall the solution quality is much better, with a smaller optimal basis condition number and no primal or dual violations. But the $Ax - b$ residuals increase slightly.

CPLEX Solution Quality, ns1687037, scale parameter set to 1

```
There are no bound infeasibilities.
There are no reduced-cost infeasibilities.
Max. unscaled (scaled) Ax-b resid.  = 1.3177e-06 (3.21704e-10)
Max. unscaled (scaled) c-B'pi resid.= 7.69362e-13 (4.1181e-15)
Max. unscaled (scaled) |x|          = 24139.1 (256.095)
Max. unscaled (scaled) |slack       = 48278.2 (256.401)
Max. unscaled (scaled) |pi|         = 75.9485 (0.195312)
Max. unscaled (scaled) |red-cost|   = 100 (0.390625)
Condition number of scaled basis    = 4.3e+06
```

Although this simple parameter change dramatically reduced solution time, fundamentally it addressed the symptom rather than the cause of the numerical challenges in this model. Let us now take a closer look at the model and try to address the cause rather than the symptom. The problem statistics (listed previously near the start of §3.1) indicate matrix coefficients on the order of 10^{-8}, below CPLEX's default feasibility and optimality tolerances. As illustrated in the small example in (13), one must determine whether coefficients smaller than CPLEX's tolerances involve round-off error that should be set to 0, or if they have meaning in the physical system being modeled. Changing the small coefficients on the order of 10^{-8} to zero makes it trivial to solve.

Optimal Objective, `ns1687037`, coefficients below 10^{-7} set to 0

```
CPLEX > tranopt
LP Presolve eliminated 49748 rows and 42875 columns.
Aggregator did 874 substitutions.
All rows and columns eliminated.
...
Dual simplex - Optimal: Objective = 0.0000000000e+00
Solution time = 0.50 sec. Iterations = 0 (0)
```

This suggests that the small coefficients have significance in the model, which is confirmed by examination of the constraints containing them. Each of these small coefficients occurs in a subset of constraints similar to the those starting with R0002624 listed below. The names of listed variables and constraints for this and all subsequent models discussed in this paper come from the original publicly available model. If a closer look at this particular collection of constraints reveals an equivalent formulation that eliminates the small coefficients, it will apply to the other similar groups of constraints as well. The variables in the constraint R0002624 except for C0025749 appear in other constraints that also restrict their activity level. Variables C0025749, C0025750, and C0025751 only appear in this group of constraints and do not appear in the objective. Variables C0000001, C0000002, C0000003, and C0000004 are penalty variables. They only appear in this group of constraints and with positive coefficients in the objective to be minimized.

Sample group of constraints in `ns1687037`, involving coefficients below 10^{-7}

```
R0002624: 50150 C0024008 + 50150 C0024010 + 50150 C0024012
            + 50150 C0024014 + 50150 C0024016 + 113600 C0024020
            + 50150 C0024024 + 113600 C0024026 + 113600 C0024038
            + 113600 C0024040
...
            + 227200 C0025718 + 227200 C0025722 + 97735 C0025726
            + 69070 C0025728 + 69070 C0025734 + 47585 C0025738
            + 50150 C0025742 + 50150 C0025744 + 69070 C0025748
            + C0025749 = 50307748
R0002625: -C0025749 + C0025750 ≥ 0
R0002626: C0025749 + C0025750 ≥ 0
R0002627: 1.9877659e-8 C0025750 - C0025751 = 0
R0002628: C0000001 - C0025751 ≥ 0
R0002629: C0000002 - C0025751 ≥ -0.0005
R0002630: C0000003 - C0025751 ≥ -0.0008
R0002631: C0000004 - C0025751 ≥ -0.0009
```

These constraints fall into three groups. The first group consists of just R0002624, a soft constraint governed by the variables with nonunit coefficients. C0025749 is a free variable that measures the violation of R0002624. Therefore, a positive value measures the shortfall for the constraint, and a negative value measures the surplus. Constraints R0002625 through R0002627 constitute the second group. They measure the scaled absolute value of the violation for the soft constraint R0002624. Specifically, constraints R0002625 and R0002626 introduce a nonnegative variable C0025750 that measures the absolute value of the violation in C0025749. Specifically, if C0025749 > 0, then R0002625 can be active and ensures that C0025750 is at least the positive violation. In that case, R0002626 is nonbinding. If C0025749 < 0, the reverse holds, with R0002625 nonbinding, and R0002626 pushing C0025750 to be at least the positive surplus value associated with $-$C0025749. R0002627 then scales C0025750 by a value on the order of 10^{-8}, assigning the result to C0025751. This was first observed, independently, by E. D. Anderson in his blog at http://erlingdandersen.blogspot.com/2011/12/observation-about-ns1687037.html. Constraints R0002628 through R0002631 comprise the third group of constraints. They introduce the actual nonnegative penalty variables in the objective, C0000001 through C0000004. It follows that C0025750 equals the absolute value of the violation of constraint R0002624, rather than exceeds it. If C0025750 exceeded the absolute violation given by |C0025749|, the excess would propagate into values in the penalty variables C0000001, C0000002, C0000003, and C0000004 that could be reduced without changing the actual activities in the LP. Therefore, C0025751 measures the absolute value of the violation of constraint R0002624, but scaled by this value of $1.9877659 \cdot 10^{-8}$.

Because C0000001 through C0000004 are penalty variables, they only appear in the objective and constraints R0002628 through R0002631. So the scaling in constraint R0002627 really is unnecessary. It merely scales down the violations to a level that challenges CPLEX to distinguish between legitimate reductions in the objective value and round-off error. Instead, one can change the units on the penalty variables C0000001 through C0000004 but leave the objective coefficients unchanged, as the objective consists only of such penalty variables. In other words, one can rewrite the above constraints with C0025751 measuring the same order of magnitude of the absolute value of the violation of R0002624 and then adjust the last four constraints to do likewise for C0000001 through C0000004. The objective coefficients remain unchanged. This reformulation implicitly scales the objective by 10^8, but one can recapture the actual objective and variable values for C0025751 and C0000001 through C0000004 by dividing them by 10^8 if needed. One can also view this reformulation as a rescaling of constraints R0002627 through R0002631 by 10^8, followed by a change of variables $x' = 10^8 \cdot x$ for variables C0025751 and C0000001 through C0000004. Regardless of the interpretation, the modified constraints no longer contain any problematic small coefficients below 10^{-7}:

Rescaled Formulation, ns1687037

```
R0002624: 50150 C0024008 + 50150 C0024010 + 50150 C0024012
          + 50150 C0024014 + 50150 C0024016 + 113600 C0024020
          + 50150 C0024024 + 113600 C0024026 + 113600 C0024038
          + 113600 C0024040
...
          + 227200 C0025718 + 227200 C0025722 + 97735 C0025726
          + 69070 C0025728 + 69070 C0025734 + 47585 C0025738
          + 50150 C0025742 + 50150 C0025744 + 69070 C0025748
          + C0025749 = 50307748
R0002625: -C0025749 + C0025750 ≥ 0
R0002626: C0025749 + C0025750 ≥ 0
R0002627: 1.9877659 C0025750 - C0025751 = 0
R0002628: C0000001 - C0025751 ≥ 0
R0002629: C0000002 - C0025751 ≥ -50000
R0002630: C0000003 - C0025751 ≥ -80000
R0002631: C0000004 - C0025751 ≥ -90000
```

Table 1. CPLEX 12.6 performance, ns1687037, original formulation.

ns1687037 (original formulation)			
	Algorithm		
Settings	Primal	Dual	Barrier
Default	14214.6	21094.2	1258.23
Scaling = 1	11164.64	907.5	83.52

Table 2. CPLEX 12.6 performance, ns1687037, modified formulation.

ns1687037 (modified formulation)			
	Algorithm		
Settings	Primal	Dual	Barrier
Default	2310.9	2926.5	41.4
Scaling = 1	6890.8	1054.7	68.2

This modified formulation has two primary advantages. First, the change of units removed the small coefficients from the matrix. That increased the right-hand side values in constraints R0002628 through R0002631 by a factor of 10^8. But, although larger values in the right-hand side can introduce round-off error resulting from floating point arithmetic operations, they do not affect the *LU* factorization that is essential for virtually all important simplex algorithm calculations. Second, the model already included much larger right-hand-side values such as the one for R0002624, so the range of coefficients in the right-hand side does not increase, yet the range of coefficients in the constraint matrix has improved dramatically.

Tables 1 and 2 compare CPLEX 12.6 performance on the original and modified formulation for all three of CPLEX's LP algorithms, using both default and nondefault scaling. All run times are reported in seconds of wall clock time with no competing jobs running. The results indicate that the modified model solves at least *six* times faster across all algorithms with default scaling. With nondefault scaling, the results are mixed, but the modifed model solves faster for the primal simplex and barrier algorithms and slightly slower for the dual simplex algorithm. Overall, CPLEX solves the modified formulation in significantly less time.

Summarizing the experience with ns1687037, examination of the problem statistics sheds light on a nondefault parameter setting that improved performance dramatically for the three fundamental LP algorithms. Subsequent examination of the constraints involving the numerically challenging aspect of the model led to a simple reformulation that yielded additional performance improvements. Thus, although parameter settings to treat the symptoms of the performance problems sufficed to improve results, addressing the root cause in the model formulation worked even better.

4. Common Sources of Ill-Conditioning

Ill-conditioning in a model can involve an arbitrarily large number of constraints or variables. Or it can involve a small collection of constraints or variables that are difficult to locate. In either case, diagnosis and removal of the ill-conditioning can be difficult. This is particularly true if changing parameters or examining the problem statistics, as described in the previous section, do not resolve the problem. Section 5 subsequently describes some tactics to help resolve such challenging cases, and this section examines some common sources of ill-conditioning that appear frequently in practice. Before trying to identify a specific source of ill-conditioning in the model, the practitioner may save time by assessing if his model contains any of the common characteristics discussed in this section. Proactively removing such characteristics may remove the source of ill-conditioning without requiring the practitioner to perform the potentially time consuming task of identifying individual parts of the model that cause the trouble.

Here is a list of common sources of ill-conditioning to consider.

1. *Mixtures of large and small coefficients in the problem data*. Section 2 described how arithmetic operations on numbers of much different orders of magnitude introduce more round-off error than the corresponding operations on numbers of similar magnitude. Although a wide range of coefficients does not necessarily imply the basis matrices associated with the model have higher condition numbers, the increase in round-off error associated with arithmetic operations can magnify the error in the computed solutions associated with those matrices. For example, consider an LP with coefficients of similar orders of magnitude. A basis condition number of 10^8 lies below the condition number threshold of 10^{10} discussed in §2.3. Round-off error associated with arithmetic operations involving these numbers of similar orders of magnitude will probably remain at the level of machine precision (10^{-16}). If so, the potential error in computed solutions to the linear systems of equations involving the basis matrices given by (12) remains bounded by $10^{-16} \cdot 10^8 = 10^{-8}$, two orders of magnitude below CPLEX's default feasibility and optimality tolerances of 10^{-6}. So there is little danger of the optimizer making algorithmic decisions based on round-off error. By contrast, for an LP with coefficients of dramatically different orders of magnitude, the round-off error from arithmetic operations could easily be on the order of 10^{-11} instead of 10^{-16}. For the same basis condition number of 10^8, the potential error in computed solutions rises to $10^{-11} \cdot 10^8 = 10^{-3}$, which means CPLEX could (although is not guaranteed to) make algorithmic decisions based on round-off error.

2. *Imprecise data calculations resulting in near singular matrices*. Gastinel and Kahan's result described in (19) showed that near-singular matrices are ill-conditioned. State-of-the-art simplex algorithm implementations have no trouble handling matrix rows or columns that are truly linearly dependent. For linearly dependent rows, the slack variable of at least one of the rows remains basic throughout the simplex algorithm iterations. For linearly dependent columns, the simplex method variable selection rules and ratio test ensure that both columns are never both in the basis simultaneously. However, nearly linearly dependent rows or columns can result in near singular basis matrices that are ill-conditioned. The following small example, drawn from Klotz and Newman [18], illustrates this:

$$\begin{aligned}
\text{P1:}\quad & \text{maximize} \quad && x_1 + x_2 \\
& \text{subject to} && c_1\colon\ x_1 + 2x_2 = 3, \\
& && c_2\colon\ \tfrac{1}{3}x_1 + \tfrac{2}{3}x_2 = 1, \\
& && x_1, x_2 \geq 0.
\end{aligned} \tag{26}$$

The LP P1 has data in exact precision, and its two constraints are linearly dependent. The dependency of the rows means that under exact precision the simplex algorithm never constructs a basis consisting of x_1 and x_2; at least one of the two slack variables always remains in the basis. However, neither $\frac{1}{3}$ nor $\frac{2}{3}$ can be stored exactly in IEEE single or double precision, so storing the data does introduce some round-off error. If the practitioner rounds to single precision, the LP provided to the optimizer is

$$\begin{aligned}
\text{P2:}\quad & \text{maximize} \quad && x_1 + x_2 \\
& \text{subject to} && c_1\colon\ x_1 + 2x_2 = 3, \\
& && c_2\colon\ 0.33333333x_1 + 0.66666667 = 1, \\
& && x_1, x_2 \geq 0.
\end{aligned} \tag{27}$$

Now, the two constraints are no longer linearly dependent. A basis matrix consisting of x_1 and x_2 is just barely nonsingular. Its inverse is

$$\begin{pmatrix} 66666667 & -2\cdot 10^8 \\ -33333333 & 10^8 \end{pmatrix}. \tag{28}$$

Thus, the norm of the inverse, as well as the condition number of this basis, is on the order of 10^8, whereas all other basis matrices have condition numbers less than 10. The rounding to single precision legitimizes this relatively ill-conditioned basis matrix. In a larger model, this type of rounding can introduce ill-conditioned submatrices, whose associated pivots in the basis factorization result in larger condition numbers for complete basis matrices. Note also that rounding the fractions to double precision actually increases the order of magnitude of the corresponding basis condition number to 10^{16}. Paradoxically, however, this actually resolves the problem. As of this writing, CPLEX uses a singularity threshold of 10^{-13}, so it would ignore any nonsingularity below that threshold. This means that rounding should be done at a precision level exceeding 13 decimal places to avoid introducing ill-conditioning. Other sources of rounding include text files that impose limits on the precision of numerical values. For example, the strict standard for MPS format has a limited field for numerical values of 13 characters that can force rounding or truncation. Fortunately, CPLEX extends the MPS standard to allow variable length fields.

Although rounding data values as precisely as possible helps avoid introducing unnecessary ill-conditioning into the model representation, the practitioner should look for opportunities to remove such rounding completely rather than perform it more precisely. For example, by multiplying constraint c_2 in P1 by 3 before passing it to the optimizer, the problem data consists of all integer values. This eliminates all round-off error in the data input, and x_1 and x_2 are not both basic during any simplex algorithm iteration.

3. *Long sequences of transfer constraints*. The following example illustrates how seemingly benign constraints can combine to create ill-conditioning:

$$\begin{aligned} x_1 &= 2x_2, \\ x_2 &= 2x_3, \\ x_3 &= 2x_4, \\ &\vdots \\ x_{n-1} &= 2x_n, \\ x_n &= 1, \\ x_j \ge 0 &\quad \text{for } j = 1, \ldots, n. \end{aligned} \tag{29}$$

This defines a square linear system with a right-hand side consisting of the unit vector e_n and a square matrix

$$\tilde{B} = \begin{pmatrix} 1 & -2 & & & & \\ & 1 & -2 & & & \\ & & 1 & & & \\ & & & \ddots & & \\ & & & & 1 & -2 \\ & & & & & 1 \end{pmatrix}. \tag{30}$$

At first glance, this linear system shows no sign of being ill-conditioned or creating unnecessary round-off error. All coefficients have the same order of magnitude. The values are all integer and, in fact, can be represented exactly as IEEE doubles. The matrix is upper triangular, so the eigenvalues are just the diagonal elements. There is no obvious linear combination of rows or columns that results in a vector with a small norm. So, in terms of (absolute) distance to singularity, Gastinel and Kahan's result (19) does not suggest ill-conditioning. Yet, for even modest values of n, this system is highly ill-conditioned. First of all, consider the most general definition of ill-conditioning in (1). This states that the model is ill-conditioned if there exists a small change to the input that yields a much larger change to the output. With the right-hand side of the unit vector e_n, the solution to this system is the vector $\lambda = (2^{n-1}, 2^{n-2}, \ldots, 2, 1)$. If we change the right-hand side of the last constraint to $1 + \epsilon$ for some small $\epsilon > 0$, each preceding constraint multiplies the round-off error by 2. The solution value for x_1 thus increases by $\epsilon \cdot 2^{n-1}$. The value for x_{n-k} increases by $\epsilon \cdot 2^k$ for $k = 0, \ldots, n-1$. Thus, a small change in the right-hand side yields a much larger absolute change in the computed solution. The condition number definition for a square matrix in (12) also indicates potential ill-conditioning. Performing the straightforward Gaussian elimination on $\tilde{B}$,

$$\tilde{B}^{-1} = \begin{pmatrix} 1 & 2 & 4 & 8 & & 2^{n-1} \\ & 1 & 2 & 4 & & 2^{n-2} \\ & & 1 & 2 & & 2^{n-3} \\ & & & 1 & & \\ & & & & \ddots & 2 \\ & & & & & 1 \end{pmatrix}. \tag{31}$$

Thus, $\tilde{B}^{-1}$ has an ∞ matrix norm on the order of 2^n, implying a condition number of $3 \cdot 2^n$ for $\tilde{B}$. This, in turn, provides a relatively tight upper bound on the absolute change factor of 2^{n-1} in the computed solution relative to the change in input obtained by perturbing the right-hand side of the last element. Note that for the right-hand side of the unit vector e_n in the transfer constraints (29), the absolute change factor in the computed solution is on the order of 2^{n-1}, but the relative change factor after normalizing by λ remains at ϵ. However, given the absolute nature of the feasibility and optimality tolerances of most optimizers, the linear system with the matrix (30) is ill-conditioned in practice.

Linear systems such as this occasionally appear as submatrices of constraints in LPs. They typically represent transfer or conversion constraints, and they can make ratios of large to small numbers implicit rather than explicit. If $x_2, \ldots, x_{n-1}$ appear only in constraints defined by B, they are essentially transfer variables that can be substituted out, leaving $x_1 = 2^{n-1} x_n$. That reveals an implicit mix of large and small coefficients in the original constraints that can introduce significant amounts of round-off error that propagate into other calculations. This can also appear in models that involve conversions of drastically different units of measurement (e.g., a global financial model that has to consider yen and also A shares of Berkshire Hathaway with a price per share of over a hundred thousand dollars). Frequently, one can rescale the units of measurement (e.g., measure in millions of yen) and improve ratios.

The absence of any linear combination of rows or columns in $\tilde{B}$ that yields a vector with a small norm appears to contradict Gastinel and Kahan's result. However, a closer examination in §6 explains the apparent contradiction.

4. *Model data values that result in large primal or dual variable values*. Large variable bounds or constraint right-hand-side vectors can result in large primal solution values. Large objective vectors can result in large dual solution values. Extremely large or small constraint matrix values can result in both. This can create numerical challenges in two primary ways. First, such large model data values appear in the right-hand sides of the linear solves that calculate primal and dual values during the simplex method. Small relative errors in these values can still be fairly large relative to the absolute tolerances used by the optimizer. For example, absolute round-off error representing numbers on the order of 10^9 is already on the order of 10^{-7} for IEEE double precision. Combine that with just a modest condition number of 10^8, and the potential error in the calculation of the primal and dual values is already on

the order of 10, far above CPLEX's default feasibility and optimality tolerances of 10^{-6}. Additional round-off errors can accumulate when constraint left-hand-side calculations involve large intermediate values, but the right-hand side is small. For example, if a constraint $\sum_{j=1}^{n} a_j x_j = 0$ has modest coefficients a_j, but other constraints and bounds enable large values for x_j, the additions, subtractions, and multiplications to compute the left-hand side involve large numbers, but the right-hand side involves small values. The relative error in the computed left-hand side may remain modest, but it is large relative to the right-hand-side value. Financial models can encounter this; when performing calculations involving billions or more dollars, an error in the solution of 100 is small in a relative sense but large in an absolute sense. In such cases, possible remedies include using larger feasibility tolerances, changing the units of measurement, or relaxing the constraints to accept such errors.

Summarizing this section, building up a list of common sources of ill-conditioning and unnecessary round-off error provides an easy way to proactively avoid creating numerically challenging model formulations. This can save significant amounts of model development time.

5. Diagnostics for Ill-Conditioning and Numerical Instability

Although building a list of common sources of ill-conditioning can proactively remove problems in a model formulation, sometimes the source of the trouble involves more elaborate or subtle aspects of the model. Scanning for a finite number of known conditions may not be sufficient. However, by making use of the various metrics of ill-conditioning described in §§2.2 and 2.5, the practitioner can develop some diagnostic tools to help investigate these more challenging cases.

Before even considering ill-conditioning metrics, examination of the solution can identify variables and constraints involved in the ill-conditioning. Extremely large primal and dual values can identify respective variables and constraints that are involved in the ill-conditioning. Examination of the constraint matrix coefficients associated with these variables and constraints sometimes reveals the source of the problem. In other cases, examination of associated intersecting variables and constraints is required.

When the solution values are all reasonable despite large basis condition numbers, the square matrix condition number definition in (12) suggests that large coefficients either in the basis or its inverse can help identify the constraints and variables involved in the ill-conditioning. Interactive CPLEX does not provide statistics on the coefficients of these matrices directly, but its C API is well suited to extracting this information. Diagnostic programs to extract the basis and its inverse are available at http://www-01.ibm.com/support/docview.wss?uid=swg21662382.

For MIPs, as the MIP Kappa output samples in §3.1 illustrated, only a small subset of the node LPs may have sufficiently large optimal basis condition numbers to be of interest for additional investigation. Interactive CPLEX does not include functionality to export such node LPs, but CPLEX's C API is well suited for this task as well, and a program to perform this task is available at http://www-01.ibm.com/support/docview.wss?uid=swg21662382.

If examination of the basis and its inverse does not provide an easy diagnosis of the source of ill-conditioning in the LP or MIP, some of the alternate measurements of §2.5 may be useful in certain cases. For example, if the basis matrix has small eigenvalues but its matrix norm is one or more, that indicates that a small perturbation to the diagonal of the matrix results in a singular matrix. Similarly, since the basis inverse has reciprocal eigenvalues, large eigenvalues in the basis matrix imply that a small perturbation to the diagonal of its inverse results in a singular matrix. Thus, the practitioner can examine the diagonals of the basis or its inverse for sources of ill-conditioning. CPLEX does not provide any functionality to examine eigenvalues of the basis matrix directly. However, CPLEX does have a connector to MATLAB, which does have this functionality. So one could create a program in MATLAB that extracts the basis matrix and calculates the eigenvalues.

As with eigenvalues, the distance to singularity metric of (19) can help the practitioner focus on specific sources of ill-conditioning that may be less apparent from examining the product of the norms of the basis and its inverse, as in (12). One can assess distance to singularity by finding a linear combination of the rows or columns of the basis matrix with a small norm relative to the vector expressing the linear combination, as in (20)–(22). For example, letting e be a vector of all ones, for a nonsingular but ill-conditioned basis matrix B, find the linear combination with minimum absolute value:

$$\begin{aligned} \text{P3:}\quad \text{minimize}\quad & e^T(u+w) \\ \text{subject to}\quad & c_1\colon B^T\lambda - v = 0, \\ & c_2\colon v = u - w, \\ & c_3\colon e^T\lambda = 1, \\ & \lambda \text{ free; } v \text{ free; } u, w \geq 0. \end{aligned} \tag{32}$$

The constraints in c_1 specify that the vector v is a linear combination of the rows of B. Constraints in c_2, when combined with the objective, ensure that $u + w$ measures the sum of the absolute values of the elements of v. Constraint c_3 excludes the zero vector from the set of feasible solutions. Note that one could substitute $v = u - w$ into c_1 and eliminate c_2; P3 was expressed with these two groups of constraints for consistency with the distance to singularity discussion in Equations (20)–(22) of §2.5. Thus, the LP in P3 finds the normalized linear combination of the rows of B with minimum absolute value. The nonzero elements of the optimal solution $\lambda^\star$ to P3 identify the rows of B involved in the ill-conditioning, provided that the optimal objective value is small relative to $\|\lambda\|$. Note that an ill-conditioned matrix can have multiple subsets of constraints or variables that cause ill-conditioning. The optimal solution to P3 can identify a single such subset. But if multiple disjoint subsets of this type exist, removing the source associated with the initial subset does not remove all sources of ill-conditioning. After adjusting the model to remove the initial source of ill-conditioning, one may need to repeatedly solve P3 and modify the model until the optimal objective is no longer relatively small.

Obtaining useful diagnostic information from P3 involves some significant challenges. Its optimal solution is of interest when B is ill-conditioned. The basis matrices generated by the simplex method while optimizing this LP may also be ill-conditioned, making P3 as computationally challenging as the original model that prompted an investigation. With that in mind, the parameter settings discussed in §3.2 should be considered to remedy potential numerical difficulties during the optimization. In addition, the support of the optimal λ vector may be large, making the explanation of the ill-conditioning difficult, even for a practitioner very familiar with the model. If one is willing to solve a MIP instead of an LP, the size of the support of λ can be minimized, albeit in exchange for creating a potentially much more difficult problem to solve. Specifically, one can formulate a MIP (Rothberg [28]) to find a linear combination of constraints of B that is sufficiently small with minimal support. The formulation P4 below illustrates. The constraints in c_1 represent the positive or negative part of λ. The constraints in c_2 and c_3 introduce vectors of binary variables y and z that count the number of nonzero elements in λ. In other words, their sum, which is the objective function, measures the support of λ. Combined with the objective, c_2 and c_3 ensure that for any particular elements u_j and w_j, at most one of them can assume a positive value. Constraint c_4 then represents the positive or negative elements of $B^T\lambda$. Constraint c_5 specifies that the linear combination $B^T\lambda$ is deemed sufficiently close to singular, as measured by the parameter ϵ. Constraint c_6 normalizes the linear combination analogously to constraint c_3 in P3. Given the reciprocal relationship between distance to singularity and condition number described in §2.5, we want to choose $\epsilon = 1/\kappa$, with κ being the threshold condition number value that characterizes the model of interest as ill-conditioned:

$$\begin{aligned}
\text{P4:}\quad \text{minimize}\quad & e^T(y+z)\\
\text{subject to}\quad & c_1\colon\ \lambda = u - w,\\
& c_2\colon\ u - y \le 0,\\
& c_3\colon\ w - z \le 0,\\
& c_4\colon\ B^T\lambda = s - t,\\
& c_5\colon\ e^T s + e^T t \le \epsilon,\\
& c_6\colon\ e^T\lambda = 1,\\
& \lambda \text{ free};\ s, t, u, w \ge 0;\ y, z \in \{0,1\}.
\end{aligned}\tag{33}$$

Given that such thresholds in practice can be on the order of 10^{10} or higher, the associated epsilon values of 10^{-10} or lower are too small for most optimizer feasibility and optimality tolerance settings. So the practitioner may need to relax this requirement and use larger values on the order of 10^{-8} or higher. Thus, similar to the LP in P3, the MIP in P4 has some computational challenges, including the small values of ϵ and the ill-conditioned square matrix B in constraint c_1.

CPLEX lacks direct functionality to solve either the LP in P3 or the MIP in P4. However, CPLEX's FeasOpt feature facilitates the creation and optimization of these models. FeasOpt is an automatic feature that, when given a set of infeasible constraints and (optionally) integer restrictions on variables, finds a solution that minimizes various measure of infeasibility, including the ones in P3 and P4. The different FeasOpt metrics include a minimal sum of infeasibilities and a minimal number of violated constraints. For example, to obtain a solution for P3, one would provide it an LP with no objective function and the following constraints

$$\begin{aligned}
\text{P3\_FeasOpt:}\quad & c_1\colon\ B^T\lambda = 0,\\
& c_3\colon\ e^T\lambda = 1,\\
& \lambda \text{ free}.
\end{aligned}\tag{34}$$

Note that this LP is infeasible for any nonsingular matrix B. FeasOpt would only be allowed to relax the constraints in c_1 and would be instructed to minimize the sum of infeasibilities. To get FeasOpt to solve P4, one faces an additional challenge in that the linear combination of the rows of B with minimal support need not have a sufficiently small norm to indicate B is close to singular. Rather, one seeks the linear combination with minimal support among all those that result in a vector with small relative norm. Therefore, the LP with an empty objective passed to FeasOpt would be

$$\begin{aligned}
\text{P4\_FeasOpt:}\quad & c_1\colon\ \lambda = u - w,\\
& c_4\colon\ B^T\lambda = s - t,\\
& c_5\colon\ e^T s + e^T t \le \epsilon,\\
& c_6\colon\ e^T\lambda = 1,\\
& c_7\colon\ u_j + w_j = 0, \quad j = 1, \ldots, m,\\
& \lambda \text{ free};\ s, t, u, w \ge 0.
\end{aligned}\tag{35}$$

The constraints in c_7 make this LP infeasible since the basis B is nonsingular. This time, FeasOpt would be instructed to minimize the number of violated constraints in c_7, which means it would solve a MIP. When FeasOpt relaxes these constraints, the constraints in c_5 exclude linear combinations of B that result in a vector with a small support but an insufficiently small norm to indicate near singularity.

6. Anomalies, Misconceptions, Inconsistencies, and Contradictions

When investigating an ill-conditioned or numerically unstable LP or MIP model, the practitioner may encounter results that initially appear surprising. This section endeavors to explain or reconcile such results so that the practitioner can focus on improving the model numerics rather than investigating unexpected behavior regarding the model numerics metrics discussed in previous sections.

6.1. Unscaled Infeasibilities

Most state-of-the-art LP and MIP solvers scale the model in order to improve its numerics. After solving the scaled model within the specified feasibility and optimality tolerances, they unscale the final solution so it corresponds to the original, unscaled model. Unscaled infeasibilities occur when the scaled model satisfies the tolerances but the unscaled model does not. The solver can then perform some additional iterations to try to resolve the unscaled infeasibilities, but, in some cases, they cannot be resolved. Here is an LP with an empty objective that illustrates unscaled infeasibilities:

$$\begin{aligned} \text{P5:}\quad & c_1\colon 3y_1 + 5y_2 + 799988z \le 800000, \\ & c_2\colon y_1 + y_2 + z \le 100000, \\ & z \in \{0,1\};\ 0 \le y_1 \le 100000;\ 0 \le y_2 \le 100000. \end{aligned} \tag{36}$$

By default, CPLEX applies equilibration scaling to the constraint matrix. This involves first scaling the constraints so that the maximum element is 1.0, then scaling the columns to also have a maximum element of 1.0. For details on different scaling methods for LPs, see Elble and Sahinidis [10]. After default scaling, c_2 remains unchanged, but c_1 is rescaled to $3/799988y_1 + 5/799988y_2 + z \le 800000/799988$. For the solution $z = 1$, $y_1 = 0$, $y_2 = 2.5$, the scaled left-hand side of c_1 is $1 + 12.5/799988$, and the scaled right-hand side is $1 + 12/799988$. The scaled infeasibility for c_1 is $0.5/799988$, the difference of the scaled left-hand and right-hand sides. This is approximately $6.25 \cdot 10^{-7}$, just barely below CPLEX's default feasibility tolerance of 10^{-6}. After unscaling, however, the same solution values result in a left- and right-hand-side values for c_2 of 800000.5 and 800000, respectively, for an unscaled infeasibility of 0.5. Note that CPLEX does not exit with unscaled infeasibilities in such a simple case as this; we present it here to illustrate the concept, which occurs quite infrequently in practice.

Although one could always disable scaling to resolve this, some models on which optimizers report unscaled infeasibilities are sufficiently ill-conditioned or numerically unstable that running them without any scaling results in other performance problems. Possible remedies include using scaling based on geometric means (i.e., the square root of the product of the largest and smallest matrix row or column elements) rather than maximums. For CPLEX, this corresponds to setting the scale parameter to the nondefault value of 1 for more aggressive scaling. If the unscaled infeasibility just barely exceeds the feasibility tolerance, the practitioner can assess its significance in the context of the physical system being modeled. If insignificant, the solution with unscaled infeasibilities may be acceptable. Otherwise, one could solve the model with a smaller feasibility tolerance so that the resulting unscaled infeasibilities are within the original feasibility tolerance of interest.

Alternatively, one could try to address the cause of the unscaled infeasibilities rather than the symptom by improving the model formulation. For example, in P5 above, since z is binary, constraint c_1 states that if $z = 1$, the constraint $3y_1 + 5y_2 \le 12$ must be enforced.

If $z = 0$, constraint c_1 is nonbinding given the upper bounds of 100000 on y_1 and y_2. If these upper bounds are larger than necessary, formulating the model with tighter bounds can help. For example, if upper bounds of 1000 are valid, one could then tighten c_1 to $3y_1 + 5y_2 + 7988z \leq 8000$ without compromising its meaning. The solution $z = 1$, $y_1 = 0$, $y_2 = 2.5$ would then have a scaled infeasibility of 0.5/7988, or approximately $6.25 \cdot 10^{-5}$, which would make it infeasible for the scaled problem as well as the unscaled model. If the bounds cannot be tightened, CPLEX also supports indicator constraints that would remove the need for the large values in constraint c_1. The indicator formulation of c_1 is simply $z = 1 \rightarrow 3y_1 + 5y_2 \leq 12$. Internally, CPLEX branches explicitly on the indicator constraint when it is violated. Thus, a violation of this indicator constraint prompts CPLEX to add the already well-scaled constraint $3y_1 + 5y_2 \leq 12$, which poses no risk of creating unscaled infeasibilities. Optimizers that do not support indicator constraints may also enable this reformulation using special ordered sets, although that approach would be more complicated.

6.2. Model Sensitivity to Feasibility and Optimality Tolerances

Consider again the feasibility problem (13) from §2.3:

$$\begin{aligned} c_1\colon\ & -x_1 + 24x_2 \leq 21, \\ & -\infty < x_1 \leq 3, \\ & x_2 \geq 1.00000008. \end{aligned} \tag{37}$$

As discussed previously, for CPLEX's default feasibility tolerance of 10^{-6}, different bases legitimately conclude feasibility or infeasibility. So, for the broadest definition of ill-conditioning given by (1), this model appears to be ill-conditioned in the sense that a small change in the input (in particular, the lower bound of 1.00000008) can lead to a big change in the computed solution. However, regarding the basis matrices, this example does not illustrate an ill-conditioned model. All of the previously discussed bases are well conditioned. Furthermore, this model is not ill-conditioned at all under perfect precision; it is fundamentally infeasible. This model is sensitive because its distance between infeasibility and feasibility is small, both in finite and in perfect precision. Under perfect precision, a change of 0.00000008 to the lower bound of x_2 removes the infeasibility. Under finite precision, the distance between feasibility and infeasibility is even smaller, as the need for a finite feasibility tolerance means that the different coordinate systems associated with the different basis matrices can determine whether an associated basic solution is feasible. Thus, the selected tolerances in the implementation of the algorithm provide an additional source of variability in the optimizer result, but the core issue involves the model being on the edge of feasibility and infeasibility. In such cases, the practitioner should focus on the relationship between the data and tolerances, as described in the discussion of this example in §2.3, rather than try to address nonexistent symptoms or causes of ill-conditioning in the model.

Ordóñez and Freund [25] develop an alternate measure of the condition number of an LP based on the size of the perturbation needed to make a feasible LP infeasible relative to the input data. Their work built upon the results of Renegar [27]. This metric can be computed in practice for (P_{LP}) by solving a series of $2(m + n)$ convex optimization problems of similar size. Some challenges remain with this approach, because many LPs that solve quite easily in practice with no ill-conditioning at all have a high condition number based on this metric (although presolving the LP helps address this limitation).

6.3. Effect of Choice of Ill-Conditioning Metric on Assessment

The definition of the condition number of a square matrix in (12) used an arbitrary matrix norm, and discussions of the condition number in numerical linear algebra texts typically do not choose a specific matrix norm. However, when interpreting condition numbers in practice, the choice of norm can dramatically affect the assessment. The matrix (30) from §4 has a condition number on the order of 2^n when using the matrix 1- or ∞-norm. The matrix 2-norm is the square root of the largest eigenvalue of $\tilde{B}\tilde{B}^T$, which also is large. On the other hand, since $\tilde{B}$ is upper triangular with unit diagonals, all of its eigenvalues are 1. However, although small eigenvalues of $\tilde{B}$ would indicate a large condition number, one cannot conclude that the absence of small eigenvalues implies a modest condition number. Specifically, eigenvalues measure sensitivity to a specific perturbation, namely, of the diagonals of the matrix. In that regard, this matrix is not ill-conditioned; no small perturbation of the diagonals makes the matrix singular or makes computed solutions to associated square linear systems sensitive to perturbations in the input data. But perturbations to the right-hand side can result in big changes to the computed solution, as was previously discussed in §4. The fundamental difference between the condition numbers in this case occurred because the eigenvalues measured a specific perturbation to which the matrix (30) was not sensitive, while the condition number definition in (12) measures arbitrary perturbations to either the right-hand side or matrix. The basis matrix in (30) is sensitive to certain perturbations in the right-hand side.

Regarding the matrix (30) and the lower bound on the condition number based on distance to singularity, no such inconsistency in condition number values can arise. However, one should keep in mind that a certificate of ill-conditioning λ may involve values requiring more precision than is available on the computer being used. For example,

$$\lambda = \begin{pmatrix} 1 \\ 2 \\ 4 \\ \vdots \\ 2^{n-2} \\ 2^{n-1} \end{pmatrix} \tag{38}$$

provides a certificate of ill-conditioning in a relative sense, but not an absolute sense. Setting $v^T = \lambda^T \tilde{B}$, one sees that v is just the unit vector e_1. So $\|v\|_1/\|\lambda\|_1 = 1/2^n$, which approaches 0 as n increases. However, for even moderate values of n, λ is difficult to compute and represent in finite precision because of the wide range of coefficients it contains. Although one could rescale λ so that the largest coefficient is more manageable, the smallest coefficients could fall below machine precision, resulting in potential problems with values below machine precision instead of values too large to represent. For example, dividing the elements of λ by 2^{n-1} would result in a vector with maximum element of 1, but minimum element of $1/2^{n-1}$. On the other hand, the fundamental definition $\kappa(\tilde{B})$ indicates ill-conditioning in both an absolute and relative sense. However, computing $\kappa(\tilde{B})$ under finite precision is also challenging, as the values computed for $\tilde{B}^{-1}$ can quickly overflow the largest double (or even quadruple) precision values as n increases. This has two fundamental implications for the practitioner. First, it illustrates why simply checking the model for a list of known sources of ill-conditioning can simplify the diagnostic process. It also illustrates the benefits of trying to reproduce the ill-conditioning in a smaller instances of the model. In this case, a model instance with $n = 20$ would have high condition numbers, but the condition number computation could proceed under IEEE double precision with reasonable accuracy. The same could not be said for a model with $n = 100$, as some of the values in $\tilde{B}^{-1}$ would be on the order of $2^{100} \approx 10^{30}$. With 16 base-10 digits of accuracy, this would imply absolute round-off error on the order of 10^{14} in the calculations. For $n > 1024$, some of the values in $\tilde{B}^{-1}$ could not even be represented in IEEE double precision.

6.4. Large Condition Numbers That Yield a Weak Bound

Keep in mind that (12) implies that $\kappa(B)$ provides an upper bound on the magnification of round-off error in the solution of a square linear system of equations compared to the error in the input. It does not provide a specific value for the round-off error. This raises the question of the strength of the bound it provides. The following example illustrates. Consider a linear system $Bx = b$ with

$$B = \begin{pmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & \ddots & & \\ & & & & 1 & \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{pmatrix}; \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ n \end{pmatrix}. \tag{39}$$

One can easily compute or verify that

$$B^{-1} = \begin{pmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & \ddots & & \\ & & & & 1 & \\ -1 & -1 & -1 & \cdots & -1 & 1 \end{pmatrix}. \tag{40}$$

By examining the pivoting operations used to calculate B^{-1} from B, one can see that each pivot element is 1.0, indicating that B is not close to singular. Furthermore, for any arbitrary small perturbation vector $\delta = (\epsilon_1, \epsilon_2, \ldots, \epsilon_n)$, $B^{-1}\delta = (\epsilon_1, \epsilon_2, \ldots, \epsilon_n - \sum_{i=1}^{n-1} \epsilon_i)$. Assuming all of the ϵ_i are of the same order of magnitude, the term $\epsilon_n - \sum_{i=1}^{n-1} \epsilon_i \approx \sum_{i=1}^{n-2} \epsilon_i$ dominates the perturbation to the computed solution. Thus, the perturbation to the output is a factor of $n-2$ larger than any individual element in the perturbation vector. Keep in mind that when applying the condition number definitions in (1) or (12), one must use the same norm in the perturbation input and resulting output. Thus, the perturbation vector δ has a vector 1-norm of $\sum_{i=1}^{n} |\epsilon_i|$. From Equation (1), this indicates a magnification factor of $(n-2)/n \approx 1$, not $n-2$. Therefore, B is quite well conditioned. However, the matrix ∞-norms of B and B^{-1} are both n, so letting $\kappa_p(B)$ denote the condition number of the matrix B using the p-norm in Equation (12), $\kappa_\infty(B) = n^2$. A condition number of n^2 indicates that this system is potentially ill-conditioned for $n \geq 100000$. By contrast, using the absolute column sum associated with the matrix 1-norm, $\kappa_1(B) = 4$, which is more consistent with the actual error analysis on this linear system. The bound established by the ∞-norm is quite weak. Similarly, for B^T, the analysis involves a single dense column rather than the single dense row of B. Therefore, the 1-norm would provide the weak bound, and the ∞-norm would provide the tight bound.

The difference between the 1- and ∞-norms in this example is an extreme case. Chapter 6 of Higham [15] shows a table of the maximum differences between two matrix norms, and for a square matrix, the difference is bounded by the dimension n of the matrix. In this example, the difference between the 1- and ∞-norm is $n-2$. Although this difference is extreme, it is not a pathological case that never appears in practice. Adding a dense row to a constraint matrix that previously had well-conditioned bases with small condition numbers can occur when performing multiobjective optimization or when adding dense cuts to node relaxations of a MIP. So, in such situations, the MIP Kappa output described in §3.1 could potentially overestimate the level of ill-conditioning in the statistics it reports.

This example has several implications. First, one should use both 1- and ∞-norms in the condition number definition in (12) to obtain a tighter bound on the error in the computed solution. Most cases are less extreme than the example in this section. Second, in the presence of a high basis condition number, examination of the associated solution quality can help shed light on whether the large condition number truly signals potential problems in the solution. A large condition number, along with residuals on primal or dual feasibility that exceed the feasibility and optimality tolerances, indicates that ill-conditioning is problematic, and removing the source of the ill-conditioning will significantly improve the accuracy of the computed solution. A large condition number accompanied by good solution quality does not validate that ill-conditioning is no problem whatsoever. But it does indicate that the large condition number did not magnify round-off error in a manner that made this particular solution inaccurate. In the latter case, the practitioner should not assume that the model has no problems with accuracy in the computed solutions. Rather, the good solution quality indicates that the bound provided by the large condition number may be weak, and additional analysis should be used to determine whether that is the case. Possible additional diagnostics include error analysis of the model or some of its individual components, as well as comparison of the 1- and ∞-norms of various basis matrices.

7. Building Blocks for Automating the Diagnosis of Ill-Conditioning

This tutorial has primarily discussed methods for identification and treatment of symptoms and causes of ill-conditioning and numerical instability in LP and MIP models and algorithms. Armed with the techniques of §§3, 4, and 5, the practitioner can address associated performance and solution accuracy problems. However, the techniques discussed require some level of manual interaction with the model. A higher level of automated diagnosis would help large industrial optimization applications that need to run with minimal manual interaction. This section briefly discusses additional work that might lead to more automated detection and diagnosis.

Problems P3 and P4 in §5 illustrated LP and MIP problems one can solve to obtain information about subsets of constraints or variables that are involved in the ill-conditioning. However, at this point, few computational results are available regarding either the computation time required or the usefulness of such computed subsets, which the practitioner must interpret. P3 and P4 fundamentally involve an ill-conditioned matrix that may challenge the optimization algorithm. At a minimum, care is needed regarding optimizer parameter settings. P4 is a MIP rather than an LP, so it may pose challenges for the typical branch-and-cut algorithm used to solve most MIPs. An alternate algorithm involving decomposition or other approaches may be needed.

Filtering might provide another approach. One could start with an ill-conditioned basis matrix and systematically remove a row and column in a manner that keeps the resulting submatrix nonsingular. When the condition number of this submatrix remains on the same order of magnitude after one such deletion, the deleted row and column can be excluded from consideration as potential sources of the problem. By contrast, if the condition number drops significantly after one such deletion, the associated row and column must be reinserted.

In contrast to the build-down approach of filtering, a build-up approach starts with the identity matrix and systematically pivots in columns from the ill-conditioned basis in question, stopping as soon as the basis matrix condition number passes a suitable threshold (e.g., 10^{10} for typical optimizer tolerances and IEEE double precision, as discussed in §2.3). As soon as the condition number passes this threshold, the rows and columns that were pivoted in should provide an explanation of the ill-conditioning in the model. If only a modest number of pivots is required, this may provide a concise explanation that the practitioner can easily interpret.

Other schemes that track the pivots during the factorization of an ill-conditioned matrix may also help identify a subset of rows and columns of the basis matrix that are ill-conditioned. Even if that subset is not minimal, it may provide useful information. Or it could be used to formulate the models P3 and P4 with a subset of the ill-conditioned basis matrix, potentially making those problems easier to solve.

8. Additional Examples

This section considers additional examples of ill-conditioned or numerically unstable models available from public test sets and illustrates how some of the remedies discussed in this tutorial can improve run time, solution accuracy, or both.

Section 3 already provided a detailed discussion of the LP model ns1687037, which we do not repeat here. The key points were that using scaling based on geometric means addressed the numerical trouble introduced by data values below CPLEX's default feasibility tolerances. In addition, treating the cause by modifying the formulation in order to remove the problematic coefficients yielded additional performance improvements. Fortunately, the problematic data values were isolated within a relatively small number of constraints that were straightforward to reformulate. Although this is true for numerous other models, it will not always be the case, and other diagnostic and remedial tactics can help the practitioner.

We now move on to the MIP model cdma from the unstable test set from MIPLIB 2010 (http://miplib.zib.de/miplib2010-unstable.php). As of this writing, this model has yet to be solved to optimality, either by a commercial optimizer such as CPLEX or by alternative, specialized approaches (e.g., adding model-specific cuts to the formulation that a commercial optimizer does not detect, decomposition algorithms, or model reformulations). Node Log #1 provides a sample of CPLEX 12.6 performance with default settings on this model. Note the extremely large objective values in both the integer solution and optimal node LP objective values. Also, note that CPLEX requires almost 15 hours to process the root node, and although the cuts and heuristics improve the best integer and best node values, the gap associated with the solution at the end of the root node is very large. Once the root node is completed, node throughput improves, and CPLEX has a best solution with a MIP gap of 192% after 20 hours. The growth in the number of active nodes (i.e., the nodes left column in the log below), the slow progress in the best node value, and the large MIP gap all indicate that one could let CPLEX run for many more days without solving the model to optimality.

CPLEX Node Log #1, default settings, MIP model cdma

```
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 4 threads.
Root relaxation solution time = 89.23 sec. (39911.83 ticks)
        Nodes                             Best        Cuts/                 Gap
    Node  Left    Objective    IInf    Integer     Best Bound    ItCnt     (%)
*     0+    0                         -1.97987e+14 -6.51749e+17   71155     ---
      0     0   -6.33335e+16  704    -1.97987e+14 -6.33335e+16   71155     ---
      0     0   -6.31118e+16  702    -1.97987e+14  Cuts: 1870   185865     ---
      0     0   -6.26076e+16  631    -1.97987e+14  Cuts: 1870   292616     ---
...
      0     0   -5.87363e+16 1206    -3.21168e+15  Cuts: 1604  4545331    ---
*     0+    0                        -7.75173e+15 -5.87363e+16 4654552 657.72
...
*     0+    0                        -1.16804e+16 -5.81032e+16 5626309 397.44
      0     0   -5.80853e+16 1585    -1.16804e+16   Cuts: 566  5632163 397.29
Heuristic still looking.
      0     2   -5.80853e+16 1583    -1.16804e+16 -5.80853e+16 5633601 397.29
```

```
Elapsed time = 52951.48 sec.
  (11682283.45 ticks, tree = 0.01 MB, solutions = 17)
      1     3   -5.80295e+16 1444 -1.16804e+16 -5.80853e+16  5643488  397.29
...
    12862 10763 -4.10127e+16 1032 -1.46845e+16 -4.29552e+16 29639775 192.52
Elapsed time = 71901.33 sec.
  (18469780.15 ticks, tree = 33.05 MB, solutions = 24)
    12866 10767 -3.86482e+16  979  -1.46845e+16 -4.29552e+16 29661467 192.52
```

The objective values on the order of 10^{16} suggest either some enormous objective coefficients or some other problem data values that result in large solution values. Examination of the problem statistics indicates the former.

CPLEX problem statistics output, MIP model cdma

```
Variables           :   7891 [Fix: 1, Box: 3655, Binary: 4235]
Objective nonzeros :   2383
Linear constraints :   9095 [Less: 8390, Greater: 645, Equal: 60]
  Nonzeros          : 168227
  RHS nonzeros      :   4145

Variables           : Min LB : 0.000000   Max UB : 1.000000e+07
Objective nonzeros : Min    : 1.000000   Max    : 1.724400e+11
Linear constraints :
  Nonzeros          : Min    : 1.000000   Max    : 5.000000e+07
  RHS nonzeros      : Min    : 1.000000   Max    : 3000000.
```

In some cases, the minimum and maximum data values listed in the problem statistics do not provide sufficient detail about the distribution of the problem data coefficients. The largest or smallest coefficient may be a single outlier that has no effect on the problem numerics. CPLEX's APIs are well suited for obtaining more detailed information, and a program to obtain a histogram of problem data coefficients is available at https://www-304.ibm.com/support/docview.wss?uid=swg21400100. Here is the histogram output for the objective coefficients.

CPLEX histogram program output, MIP model cdma

```
OBJECTIVE
Range             Count
[10^0, 10^1]:      31
[10^9, 10^10]:  1236
[10^11, 10^12]: 1116
```

Most of the objective coefficients are 10^9 or higher. Additional examination of the model reveals that the 31 variables with smaller objective coefficients of at most 10 are all binary variables with objective coefficients of 1 or 5. This indicates that these variables contribute very little to the final objective value. Yet they still could cause the branch-and-cut algorithm to do significant work trying to address their minute effect on the objective.

Examination of the node LP optimization logs and associated solution quality confirms that the large objective coefficients are problematic. For example, Iteration Log #6 shows dual simplex method output from one of the node LPs, followed by the solution quality. Many other node LPs had similar results. Note the frequent loss of dual feasibility during the dual simplex method iterations and how the residuals on dual feasibility in the solution quality exceed the optimality tolerance by six orders of magnitude.

CPLEX Iteration Log #6, node LP from MIP model cdma

```
Problem ''nodelp67_0.sav'' read.
Read time                                  = 0.10 sec. (0.74 ticks)
Iteration log ...
Iteration: 1 Scaled dual infeas            = 0.000130
Iteration: 8 Scaled dual infeas            = 0.000069
Iteration: 12 Dual objective               = -6614900586660791.000000
Iteration: 23 Scaled dual infeas           = 0.000107
Iteration: 32 Dual objective               = -6614900586660791.000000
Iteration: 46 Dual infeasibility           = 0.000038
Iteration: 52 Dual objective               = -6614900586660791.000000
Iteration: 58 Dual infeasibility           = 0.000038
Iteration: 64 Dual objective               = -6614900586660791.000000

Maximum unscaled reduced-cost infeasibility = 7.62939e-06.
Maximum scaled reduced-cost infeasibility   = 7.62939e-06.

Dual simplex - Optimal: Objective          = -6.6149005867e+15
...
Max. unscaled (scaled) bound infeas.       = 1.81311e-07 (1.81311e-07)
Max. unscaled (scaled) reduced-cost infeas. = 7.62939e-06 (7.62939e-06)
Max. unscaled (scaled) Ax - b resid.       = 9.99989e-10 (6.10345e-14)
Max. unscaled (scaled) c - B'pi resid.     = 7.3125 (7.3125)
Max. unscaled (scaled) |x|                 = 5596 (55296)
Max. unscaled (scaled) |slack|             = 6.12827e+06 (10.6908)
Max. unscaled (scaled) |pi|                = 8.67329e+16 (4.02442e+17)
Max. unscaled (scaled) |red-cost|          = 4.31401e+17 (4.31401e+17)
Condition number of scaled basis           = 5.2e+08
```

The huge residuals on dual feasibility indicate that the dual simplex algorithm makes decisions based totally on round-off error. Note that the basis condition number is reasonable, as was the case for all other node LPs examined. This suggests that the instability of the model does not primarily involve ill-conditioning. Rather, the large objective coefficients have 16 digits of accuracy, which implies potential round-off error on the order of 10^{-5} just in their finite precision representation. This already slightly exceeds CPLEX's default feasibility and optimality tolerances, and the modest condition numbers such as the one in the solution quality for the above node LP can then magnify it to levels many orders of magnitude above those tolerances. The LP and MIP algorithms then make decisions based on round-off error, which can result in significant amounts of wasted computation.

The preceding analysis suggests that performance could be improved if the large objective coefficients could be reduced without significantly compromising the meaning of the model. Fortunately, this can be accomplished. The 31 binary variables with relatively tiny objective coefficients of 1 and 5 can be removed from the objective. The relative impact on the overall objective is at most 10^{-10}. So an optimal solution to the model with those coefficients removed will remain within a small fraction of 1% of optimal for the original model. After changing the objective coefficients of these 31 binary variables to zero, one can then rescale the remaining large objective coefficients to reasonable values. This will then remove the large dual residuals, the primary source of the instability.

Node Log #2 shows some results for this modified formulation of cdma, after removing the 31 small objective coefficients and dividing the remaining large objective coefficients by 10^9. With a well-scaled objective, CPLEX's simplex algorithms no longer make decisions based on the massive amounts of round-off error in the reduced cost calculations. This, in turn, leads to a huge reduction in cumulative simplex iteration counts for the node subproblem solves. For example, the cumulative iteration count at the end of the root node dropped by more than a factor of 5, from 5633601 to 891048. The root node processing time improved by a factor of over 20. Node throughput improved after the root node as well.

CPLEX Node Log #2, modified MIP model cdma, objective divided by 10^9

```
            Nodes                            Best         Cuts/                   Gap
       Node  Left    Objective    IInf      Integer     Best Bound      ItCnt     (%)
*      0+     0                          -364040.0000  -6.51749e+08      74       ---
       0      0    -6.33335e+07   701    -364040.0000  -6.33335e+07      74       ---
*      0+     0                         -8331483.3067  -6.33335e+07    34506    660.17
       0      0    -6.30976e+07   732   -8331483.3067   Cuts: 1870     34506    657.34
...
       0      0    -5.81038e+07  1557    -1.29488e+07    Cuts: 792     891041    348.72
Heuristic still looking.
Heuristic still looking.
Heuristic still looking.
       0      2    -5.81038e+07  1547    -1.29488e+07  -5.81038e+07   891048    348.72
Elapsed time = 2580.99 sec. (1213168.03 ticks, tree = 0.01 MB, solutions = 12)
       1      3    -5.80196e+07  1397    -1.29488e+07  -5.81038e+07   899299    348.72
...
   12305   9821    -2.80907e+07   267    -1.33001e+07  -4.62919e+07 11706518    248.06
   12474   9970    -3.90051e+07   562    -1.33001e+07  -4.62919e+07 11806093    248.06
Elapsed time = 10325.21 sec. (3250815.76 ticks, tree = 30.84 MB, solutions = 14)
...
   26002  20095    -4.24866e+07   930    -1.61393e+07  -4.55842e+07 25961444    182.44
Elapsed time = 19130.24 sec. (5058128.09 ticks, tree = 66.59 MB, solutions = 24)
   26018  20110    -4.20128e+07   892    -1.61393e+07  -4.55842e+07 25987523    182.44
...
   79351  64440    -4.07095e+07   760    -1.82582e+07  -4.45425e+07 95768661    143.96
   79431  64514    -3.25658e+07   520    -1.82582e+07  -4.45425e+07 95881388    143.96
Elapsed time = 71862.83 sec. (13884870.84 ticks, tree = 220.25 MB, solutions = 29)
```

This run and the one in Node Log #1 on the original model were done with default settings. The faster node throughput and better numerical characteristics of the modified formulation create additional opportunities for nondefault parameter settings to improve performance. In the original run in Node Log #1, over 14 hours elapsed before CPLEX finished processing the root node. Therefore, any nondefault parameter settings that primarily help performance only after the root node are of no use in the first 14 hours for that run. By contrast, the run in Node Log #2 finishes the root node in less than an hour. Performance tuning via parameter settings or tightening the model formulation is a separate topic that lies outside the scope of this tutorial, so we do not discuss additional performance improvements here. See Klotz and Newman [19] for a general discussion of approaches that might yield additional performance improvements.

This example illustrates the potential for performance degradation caused by a mix of large and small objective coefficients. It also suggests that solving multiple problems that each have a single objective can take much less time than optimizing over a blended objective. By separating an objective with a wide range of coefficients into a hierarchy of objectives, each with coefficients of similar orders of magnitude, one can then solve a series of models, each having a well-scaled objective. The first optimization run considers the group of coefficients with the largest values. The next run considers the next largest group of coefficients while also constraining the solution to be optimal (or near optimal, depending on the practitioner's goals for the overall multiobjective optimization) relative to the objective function value of the previous run. One can repeat the process for each group of separate objective coefficients. The time saved from performing each optimization on a well-scaled objective frequently exceeds the additional time required for the multiple optimizations. This approach works for either LPs or MIPs, although the methods to constrain the objective functions can differ.

A quick examination of other models in the MIPLIB 2010 unstable test set indicate that several other models could benefit from this hierarchical objective approach. These include neos-1112782, neos-1112787, neos-1225589, ns2017839, and neos-520729. Although CPLEX solves them all with good solution quality, and all but neos-520729 within a minute, a hierarchical objective approach may yield additional performance improvements on these models.

We now consider de063155 and iprob, two numerically challenging models from a set of publicly available LPs from Meszaros at http://www.sztaki.hu/~meszaros/public_ftp/lptestset/problematic/. They offer an interesting contrast in the kind of behavior that the practitioner may encounter. The model de063155 clearly has an extremely wide range of coefficients that makes accurate double precision calculations very difficult. Yet CPLEX consistently solves it, with little sign of trouble in the iteration logs (although examination of the solution quality suggests a closer look at the solution is in order before using it in the physical system being modeled). By contrast, the problem data for the model iprob look very reasonable in terms of largest and smallest values, and they do not exhibit any of the other problematic characteristics (nearly linearly dependent constraints, limited precision in rounding of data, etc.) previously discussed in this tutorial. Yet a more detailed examination of this model reveals that it is highly ill-conditioned under perfect precision, and minute changes to the problem data can make the difference between a conclusion of feasibility or infeasibility.

Let us begin the analysis of de063155. Examination of the model statistics reveals an extremely wide range of coefficients.

CPLEX problem statistics, model de063155

```
Variables          : 1488   [Nneg: 756, Fix: 205, Box: 215, Free: 228, Other: 84]
Objective nonzeros : 852
Linear constraints : 852    [Less: 360, Equal: 492]
  Nonzeros         : 4553
  RHS nonzeros     : 777

Variables          : Min LB : -10000.00      Max UB : 30.90000
Objective nonzeros : Min    : 1.279580e-05   Max    : 1000.000
Linear constraints :
  Nonzeros         : Min    : 2.106480e-07   Max    : 8.354500e+11
  RHS nonzeros     : Min    : 0.0002187500   Max    : 4.227560e+17
```

CPLEX consistently finds an optimal solution with optimal objective $9.8830944565 \cdot 10^9$ in less than 0.1 seconds, regardless of the choice of simplex method and scaling parameter setting. For brevity, we do not display any of the iteration logs. Some do show limited signs of numerical problems in terms of loss of dual feasibility or increases in the Markowitz tolerance. Still, if one did not look at the problem data, one would not have any major concerns about wasted iterations based on the iteration logs. However, examination of the solution quality raises some significant questions about the usefulness of the solution relative to the physical system being modeled. Here is the solution quality for a dual simplex method run with default settings; although the specific values vary for nondefault scale parameter settings and the primal simplex method, the implications remain the same.

Solution quality, de063155, dual simplex with default parameter settings

```
There are no bound infeasibilities.
There are no reduced-cost infeasibilities.
Max. unscaled (scaled) Ax - b resid.   = 747.949 (5.12641e-08)
Max. unscaled (scaled) c - B'pi resid.= 7.74852e-10 (8.51025e-06)
Max. unscaled (scaled) |x|            = 3.10148e+13 (3.76112e+07)
Max. unscaled (scaled) |slack|        = 3.75814e+07 (3.75814e+07)
Max. unscaled (scaled) |pi|           = 62061.1 (5.106e+09)
Max. unscaled (scaled) |red-cost|     = 6.78639e+09 (8.5923e+09)
Condition number of scaled basis      = 1.7e+08
```

The primal residuals are below CPLEX's feasibility tolerance for the scaled model CPLEX solved. For the unscaled model, the maximum absolute primal residual of 747.949 exceeds the feasibility tolerance. However, the primal solution values have norm on the order of 10^{13}, so the relative primal residuals are on the order of 10^{-11}. Such solution quality with small relative and large absolute primal or dual residuals does not necessarily indicate a problem with the result. Rather, it asks the practitioner to consider whether, for the physical system associated with this model, a constraint violation on the order of 747.949 is acceptable. To assess this, the practitioner must first identify the constraints with large residuals, then assess the significance of the violation. If the residuals are acceptable, then the solution can be used. If such residuals are excessive, then more precision is needed. Turning on CPLEX's numerical emphasis parameter on a machine with an Intel chip and the associated 80-bit extended precision registers offers only about three additional base-10 digits of accuracy. Not surprisingly, setting this parameter did not improve the primal residuals.

The analysis of model de063155 illustrates the importance of examining solution quality, distinguishing between absolute and relative error in the solution, and assessing the meaning of any residuals or bound violations in the context of the physical system associated with the model.

Let us now consider the model iprob. This model is also modest in size, and the problem statistics look quite reasonable.

CPLEX problem statistics, model iprob

```
Variables          : 3001   [Free: 3001]
Objective nonzeros : 3000
Linear constraints : 3001   [Equal: 3001]
  Nonzeros         : 9000
  RHS nonzeros     : 3000

Variables          : Min LB : all infinite   Max UB : all infinite
Objective nonzeros : Min    : 1.000000       Max    : 1.000000
Linear constraints :
  Nonzeros         : Min    : 0.01030000     Max    : 9940.000
  RHS nonzeros     : Min    : 1.000000       Max    : 1.000000
```

Although the numerical data are unremarkable here, the problem dimensions and variable characteristics are noteworthy for the ensuing discussion. All constraints are equality constraints, and all variables are free variables. This model involves solving a single square linear system of equations. As long as the associated square matrix is nonsingular, this system of equations has a unique solution, which is feasible (under perfect precision). CPLEX declares this model infeasible, which implies this square matrix is singular.

CPLEX dual simplex method result for iprob, default settings

```
Infeasibility row ''r1687'': 0 = 284.596.
Presolve time = 0.01 sec. (17.02 ticks)
Presolve - Infeasible.
```

Disabling presolve, so that the simplex or barrier algorithms run, consistently yields the same conclusion. Solution quality shows no indications of problems. However, making a relatively small change to any of the nonunitary coefficients in the constraint matrix results in a conclusion of feasibility. So, a small change to the input leads to a big change in the output. Yet, the basis condition number associated with the infeasible run is on the order of 10^7, indicating no ill-conditioning for the final basis. In addition, CPLEX has a conflict refiner feature that computes a minimal subset of constraints and variables that explain the source of infeasibility in an LP or MIP. Conflict refiner output should also define an infeasible problem, yet the constraints and variable bounds in the conflict generated for the model iprob is feasible. What is the source of these inconsistent results?

Fortunately, iprob has a very simple design that makes additional investigation straightforward. Had it been more complex, the practitioner would probably have to perform some filtering or simplification of the model to isolate the components involved in the inconsistency. The model can be expressed as follows, with x representing variables and α and b representing constraint matrix and right-hand-side values, respectively:

$$\text{iprob:} \quad \text{minimize} \sum_{j=2}^{3001} x_j \tag{41}$$

$$\text{subject to} \quad r1\colon \sum_{j=2}^{3001} |\alpha_j| x_j = b_1, \tag{42}$$

$$rk\colon \alpha_k x_1 + x_k = b_k \quad k = 2, \ldots, 3001, \tag{43}$$

$$x \text{ free.} \tag{44}$$

For this data instance, $b_k = 1$ except for $b_{3001} = 0$. However, the specific right-hand-side values are minimally involved in any inconsistent results.

Examination of the basis that prompted CPLEX's declaration of infeasibility reveals that one slack variable remained basic. When provided a starting basis consisting of all of the free variables, CPLEX rejects this basis as singular, replacing some of the free variables with artificial variables to construct a starting basis. This, along with the initial declaration of infeasibility, indicates that CPLEX concludes that the square matrix associated with this LP is singular. If it were nonsingular, a unique solution would exist, and since all the variables in the model are free, it would be feasible. However, the question remains as to whether this conclusion of singularity involved round-off error from the finite precision calculation of the LU factorization.

The basis matrix consisting of all the free variables has the form

$$B = \begin{pmatrix} 0 & |\alpha_2| & |\alpha_3| & \cdots & & |\alpha_{3001}| \\ \alpha_2 & 1 & & & & \\ \alpha_3 & & 1 & & & \\ \vdots & & & \ddots & & \\ \alpha_{3000} & & & & 1 & \\ \alpha_{3001} & & & \cdots & & 1 \end{pmatrix}. \tag{45}$$

To simplify the discussion, permute the dense row associated with constraint $r1$ and the dense column associated with variable $x1$ to the last row and column of the matrix. Let α represent the vector of coefficients $\alpha_2, \ldots, \alpha_{3001}$, and let $|\alpha|$ represent the vector of absolute values of the individual coefficients of α. After performing this permutation, we can represent the basis of all-free variables as

$$B = \begin{pmatrix} I & \alpha \\ |\alpha^T| & 0 \end{pmatrix}. \tag{46}$$

Given this simple structure, one can compute closed-form representations of the LU factorization and inverse of B. Specifically,

$$L = \begin{pmatrix} I & 0 \\ |\alpha^T| & 1 \end{pmatrix}, \quad U = \begin{pmatrix} I & \alpha \\ 0 & -|\alpha^T|\alpha \end{pmatrix}. \tag{47}$$

Letting $\delta = |\alpha^T|\alpha$, the last diagonal element of U,

$$B^{-1} = \begin{pmatrix} I - \dfrac{\alpha|\alpha^T|}{\delta} & \dfrac{\alpha}{\delta} \\ \dfrac{|\alpha^T|}{\delta} & -\dfrac{1}{\delta} \end{pmatrix}. \tag{48}$$

L is nonsingular. One can assess singularity of B by checking the diagonals of U for elements that are 0, or looking for divisions by 0 in the closed-form expression of B^{-1}. Either way, singularity of B occurs when $\delta = 0$. If $\delta > 0$, B is nonsingular, and iprob has a unique feasible (and hence optimal) solution. If $\delta = 0$, one can see from U above that B is rank deficient by 1. Therefore, the basis of the null space of B has dimension 1 (since it is the difference between the rank and number of columns of B). Given the representation of B in (46), the vector $v = (-|\alpha|, 1)$ satisfies $v^T B = 0$, providing a certificate of singularity. Since the null space of B has dimension 1, all other nonzero vectors in the null space are multiples of v. Furthermore, since $v^T b \neq 0$ for the right-hand-side vector for iprob, v provides a linear combination of the constraints of iprob that results in a left-hand side of 0 and a nonzero right-hand side. Thus, v provides a Farkas certificate of infeasibility for iprob as well as a certificate of singularity for B.

The preceding discussion was predicated on perfect precision arithmetic associated with closed-form calculations. However, it also sheds light on why the model iprob is inherently ill-conditioned, and why one can easily obtain inconsistent results when performing finite precision calculations. As δ approaches zero, B is nonsingular but is approaching singularity. In turn, the condition number of B approaches infinity, either by Gastinel and Kahan's distance-to-singularity metric in (19), or by the original matrix condition number metric $\|B\|\|B^{-1}\|$. Regarding the latter, note that as δ approaches zero, its appearance in the denominators of (48) indicate that $\|B^{-1}\|$ approaches infinity. Hence, the condition number of B approaches infinity as well.

For the model iprob, δ is indeed zero when calculated under perfect precision, indicating that this model should be declared infeasible. So CPLEX and the other optimizers that declared infeasibility came to the correct conclusion. However, iprob is highly ill-conditioned in the sense that a single, small perturbation to an element of α can make B nonsingular, resulting in a unique feasible (and optimal) solution. For example, consider the last constraint in the model:

iprob, constraint r3001

```
r3001: x3001 - 0.0889 x1 = 0
```

Changing the coefficient of $x1$ to -0.09 yields some interesting results. With default settings, CPLEX still declares this model infeasible, rejecting the basis of all-free variables as singular. Disabling scaling enables CPLEX to accept this basis, but with a high condition number, large variable values, and massive absolute primal residuals.

Solution quality for all-free-variable basis, perturbed iprob

```
There are no bound infeasibilities.
There are no reduced-cost infeasibilities.
Maximum Ax - b residual             = 11661.1
Maximum c - B'pi residual           = 9.24498e-06
Maximum |x|                         = 2.88183e+14
Maximum |pi|                        = 112058
Maximum |red-cost|                  = 0
Condition number of unscaled basis = 5.1e+20
```

Note that the primal feasibilities on $Ax - b$, although huge in an absolute sense, remain significant but are much more modest relative to the primal variable values. These results illustrate how this model lies on the boundary of feasibility and infeasibility, with the all-free-variable basis associated with small perturbations to the vector α of matrix coefficients being highly ill-conditioned. If such a perturbation is so small that CPLEX assesses this basis as singular, it will conclude that the model is infeasible. By contrast, if the small perturbation results in an assessment that this basis is nonsingular, CPLEX will declare the model feasible (and hence optimal), albeit with potentially low solution quality and a high basis condition number.

The simple structure of this model has some other interesting ramifications. With the exception of constraint $r1$, one can delete a single constraint and intersecting variable other than x1, and obtain a smaller model with the same structure. Conversely, one could add a similar constraint and associated new variable and obtain a larger model with the same structure. For example, consider deleting constraint r3001 and variable x3001. Since the model structure remains the same, so do the closed-form representations of LU and B^{-1} in (47) and (48). Once again, we can assess singularity based on the quantity δ. Since δ was 0 before removing this constraint and variable, one can see from the model description in (41)–(44) that it now changes by the square of the coefficient of x1 in r3001. So, for this smaller instance of the model, $\delta = 0.0889^2 = 0.00790321$ under perfect precision, so the all-free-variable basis is indeed nonsingular, with δ significantly above 0. However, this instance of the model still yields inconsistent results under double precision. Running CPLEX with default parameter settings results in a declaration of infeasibility. By contrast, disabling scaling enables CPLEX to find the unique feasible solution associated with the all-free-variable basis. At first glance, the declaration of infeasibility seems incorrect. The distance-to-singularity metric based on δ is much greater than before the deletion of r3001 and x3001. Furthermore, the ratio of largest to smallest element in the diagonals of U is reasonable, namely, $1/\delta$. However, examination of B^{-1} in (48) reveals that it has a large matrix norm. As long as $\delta < 1$, its appearance in the denominator of the expressions that comprise B^{-1} inflate the associated values. Recall from the problem statistics that the maximum coefficient in α was 9940, and that $\delta = 0.00790321$. Therefore, the outer product in the submatrix $I - (\alpha|\alpha^T|)/\delta$ in (48) results in coefficients on the order of 10^{10}. Combine this with the 3000 rows and columns in the basis matrix, and $\|B^{-1}\|_\infty$ can be on the order of 10^{12}. From (46), $\|B\|_\infty$ can be on the order of 10^6, resulting in a condition number of 10^{18} for the all-free-variable basis. For the feasible solution found with scaling disabled, CPLEX indeed calculates a condition number on the order of 10^{18}. With such a high condition number, the perturbations introduced by enabling or disabling scaling are enough to change CPLEX's assessment (under double precision) of the singularity (or lack thereof) of the all-free-variable basis matrix, and the associated conclusion of infeasibility or feasibility.

In addition, the structure of the model is well suited to generating such perturbations when double precision calculations are involved. Whereas the problem statistics indicate that the ratio of largest to smallest values in the α vector is reasonable, the calculation of δ involves a lengthy sum of squared elements of α. The coefficient range of these squared terms varies from 10^{-4} to 10^{8}, a more significant range that can result in more round-off error. Combine that with the lengthy sum of 3001 elements, and some nontrivial round-off errors in the calculation of δ can accumulate. Specifically, we saw that after deleting r3001 and x3001, α has 3000 elements, and $\delta = 0.0889^2 = 0.00790321$ under perfect arithmetic. By contrast, under double precision, just calculating δ as the 3000-term inner product $|\alpha^T|\alpha$ yields a result of 0.007902341358, indicating round-off error on the order of 10^{-6}. This perturbation can not only yield inconsistent results regarding feasibility or infeasibility, but it can alter feasible solution values. This long summation illustrates the benefit of avoiding large intermediate values compared to final solution values. Whereas the final value of δ is on the order of 10^{-3}, intermediate sums of the inner product used in the calculation are on the order of 10^{8}. This results in summing values of greatly different orders of magnitude, an example of large intermediate values compared to final solution values, which was among the list of numerically unstable calculations described in §2.4. In fact, Higham [15] devotes a whole chapter to discussing numerically stable implementations of summations.

What does this examination of iprob and its variants imply for the practitioner? In this case, iprob is infeasible because of the singularity of the all-free-variable basis matrix when the quantity $\delta = |\alpha^T|\alpha$ is zero, and the model is feasible but increasingly ill-conditioned for this basis matrix as δ approaches zero. Although one could try to perform this calculation under higher (or even infinite) precision, the model will remain on the boundary between feasibility and infeasibility for data instances with δ close to zero. Fundamentally, the practitioner needs to assess the meaning of such small values of δ in terms of the physical system being modeled. If such small values indicate some sort of error condition or other problem in the physical system, then the practitioner should either recognize this by examining the model data before solving or by examining the solution quality after optimizing. On the other hand, if solving the model for such boundary conditions is meaningful to the physical system, other possible remedies include optimizing with quadruple (or higher) precision, or reformulating the model when δ is close to 0.

Models de063155 and iprob both illustrate examples that test the limits of double precision calculations in simplex algorithms. However, these models differ in that the limited precision was apparent in the data for de063155 but hidden in the simplex algorithm calculations for iprob.

For our final example, we consider ns2122603, another MIP model from the unstable test set from MIPLIB 2010 (http://miplib.zib.de/miplib2010-unstable.php). As with the previously discussed model cdma from this set, this model is colored red in the MIPLIB 2010 model ratings, indicating it has never been solved to optimality. However, the behavior for this model is quite different. With cdma, we saw how extremely large objective coefficients, combined with moderately high node LP basis condition numbers, resulted in slow node throughput that hindered any chances for good progress in the MIP. For ns2122603, the objective coefficients are reasonable, but the model has some matrix coefficients on the order of 10^{8}. The matrix coefficients with value 0.04166670 also suggest the presence of imprecise rounding of the data, which could create ill-conditioning in the model.

Problem statistics, ns2122603

```
Variables          : 19300 [Nneg: 11712, Binary: 7588]
Objective nonzeros:  5880
Linear constraints: 24754 [Less: 14706, Greater: 7560, Equal: 2488]
  Nonzeros         : 77044
  RHS nonzeros     : 10088

Variables          : Min LB : 0.000000   Max UB : 1.000000
Objective nonzeros: Min    : 1.000000   Max    : 450.0000
Linear constraints:
  Nonzeros         : Min    : 0.04166670 Max    : 1.000000e+08
  RHS nonzeros     : Min    : 0.02083335 Max    : 1.000000e+08
```

CPLEX's results for ns2122603 vary dramatically, even when making irrelevant changes to the way CPLEX solves the model. The disparity involves not only run time but also the claimed optimal objective value. This can occur with MIPs with highly ill-conditioned node LP optimal bases, which the MIP Kappa statistics confirm.

MIP Kappa output, ns2122603

```
Max condition number:                        6.0818e+23
Percentage (number) of stable bases:         1.89%       (108)
Percentage (number) of suspicious bases:     60.29%      (3446)
Percentage (number) of unstable bases:       35.22%      (2013)
Percentage (number) of ill-posed bases:      2.61%       (149)
Attention level:                             0.137747
```

With basis condition numbers as high as 10^{23}, the perturbations on the order of 10^{-16} arising from the problem data representation can be magnified as high as 10^{7}. Inconsistent results can easily arise because of this round-off error in the solves that dramatically exceeds the optimizer's tolerances. An ill-conditioned optimal node LP basis at the root node can prompt a conclusion of infeasibility for a model for which feasible solutions exist.

To investigate the performance problem, we used the program at http://www-01.ibm.com/support/docview.wss?uid=swg21662382 to generate node LPs with condition numbers larger than 10^{15}, then examined some of the associated solutions, looking for unusually large or small values. The solution quality output from one such node LP indicates large values for the primal variables and slacks. Reduced costs also have large values, but the primal values seem more likely to be involved since the round-off error reported in the primal residuals significantly exceeds the corresponding error in the dual residuals.

Ill-Conditioned Node LP Solution Quality, ns2122603

```
Max. unscaled (scaled) bound infeas.        = 1.72662e-12 (1.72662e-12)
Max. unscaled (scaled) reduced-cost infeas.= 3.97904e-13 (3.97904e-13)
Max. unscaled (scaled) Ax - b resid.        = 7.73721e-07 (9.67152e-08)
Max. unscaled (scaled) c - B'pi resid.      = 4.33376e-12 (4.33376e-12)
Max. unscaled (scaled) |x|                  = 4.8e+09 (6e+08)
Max. unscaled (scaled) |slack|              = 4.81015e+09 (6e+08)
Max. unscaled (scaled) |pi|                 = 66773.2 (6.71089e+07)
Max. unscaled (scaled) |red-cost|           = 1.00002e+08 (1.34218e+08)
Condition number of scaled basis            = 8.6e+17
```

Examination of the primal variables reveals that most of them have very modest values, and a small subset assume the large ones.

Ill-Conditioned Node LP Optimal Primal Variable Values, ns2122603

```
Variable Name            Solution Value
C0213                          0.007973
C0217                          0.025673
...
C10142                        36.450029
C10458                4799996160.003072
C10459                4799996160.003072
...
C11441                2399998080.001536
C11442                2399998080.001536
...
C11710                2583222987.267681
C11711                2583222987.267681
C11712                         7.851078
C11914                         0.191344
...
C19155                         0.000476
All other variables in the range 1-16642 are 0.
```

Examination of the constraints intersecting such variables identifies some with potential for imprecisely rounded coefficients. For example, looking at the solution for variable C11441 leads to the constraint, R13553, and its associated variable solution values.

Intersecting Constraints, Variable C11441, Model ns2122603

```
Variable C11441 coefficients
  Linear Constraint:  Coefficient:
  Linear Objective          0
              R13353   -0.0416667
              R14613        1
              R15873        1
              R17133        1
              R18393        1
R13353: C10181 - 0.0416667 C11441 - 100000000 C12701 + 100000000 C14025
         ≥ -100000000
Variable Name          Solution Value
C11441          2399998080.001536
The variable ‘‘C12701’’ is 0.
The variable ‘‘C14025’’ is 0.
The variable ‘‘C10181’’ is 0.
```

This underscores the question raised by the problem statistics about the coefficient value of 0.0416667 and whether the apparent rounding in the seventh decimal place plays a role in the ill-conditioning. Note that 0.0416667 appears to be the rounded value for $\frac{1}{24}$. Note also that this constraint contains much larger coefficients of 10^8. So this constraint has two of the problematic characteristics discussed previously in §4—namely, mixtures of large and small coefficients and imprecise rounding. The other variables in this constraint all have solution values of 0. Therefore, the rounding at the seventh decimal place in the input has a much more profound effect on the output. Had the value of 0.0416667 been represented exactly as $\frac{1}{24}$, the value for C11441 would have been 2400000000 instead of 2399998080.001536. So a change in the input on the order of 10^{-7} would result in a change in the output on the order 10^3. These changes were absolute rather than relative, but one must take care to assess whether such changes in the output, although modest in a relative sense, impact the physical system being modeled. Unfortunately, although many models from the MIPLIB 2010 test set contain descriptions of the formulation and references, little is known about `ns2122603`, because it was submitted to the NEOS Server for Optimization (http://www.neos-server.org) without any background information.

Based on experience with numerous other models, we decided to assume that values in `ns2122603` such as 0.0416667 were indeed imprecisely rounded values that could be represented exactly by multiplying constraints by the appropriate integer values. Since 0.0416667 was interpreted as the rounded value of $\frac{1}{24}$, the rescaled constraints containing that value such as R13353 become

Constraint R13353, Rescaled by 24

```
R13353: 24 C10181 + 2400000000 C14025 - 2400000000 C12701
          -1.0 C11441 ≥ -2400000000
```

Similarly, the model also contained coefficient values of 8.750007 and 0.02083335. The former was interpreted as $\frac{35}{4} = 8.75$, and the latter was interpreted as $\frac{1}{48}$, with the associated constraints rescaled appropriately to eliminate any rounded values. Note that any arbitrary numeric value that repeats its decimal values periodically can be translated into the corresponding rational representation using a simple algorithm described at http://en.wikipedia.org/wiki/Repeating_decimal and https://www.khanacademy.org/math/algebra/solving-linear-equations-and-inequalities/conv_rep_decimals/v/coverting-repeating-decimals-to-fractions-1.

After rescaling all the decimal values to remove the rounding, the resulting model remains plagued by inconsistencies in run times and optimal objective values, along with the same high level of MIP Kappa statistics. Therefore, additional investigation was needed to improve the conditioning of the model. The mixture of small and large matrix coefficients remained, as is illustrated by constraint R13353. Variables C14025 and C12701 in that constraint are both binary variables. The values of 2400000000 (originally 100000000 before scaling by 24 to address the rounding issue) appear to be arbitrarily large "big M" values designed to enforce a logical condition, rather than legitimate data values associated with some aspect of the physical system being modeled. If so, constraint R13353 states that C11441 ≤ 24 C10181 if C12701 is 1 and C14025 is 0. For all other values of C12701 and C14025, the resulting right-hand side of the constraint is at most −2400000000, ensuring that constraint R13353 will be satisfied. One can then remove the large numerical values associated with constraints such as this by taking advantage of CPLEX's indicator constraint functionality. To use indicator constraints, one must introduce a binary variable that assumes a value of 1 if C12701 is 1 and C14025 is 0; otherwise, it assumes a value of 0. Letting Z be the binary variable, this can be accomplished with the following three constraints. The first constraint forces Z to 1 when the condition holds; otherwise, it is nonbinding. Similar logic applies to the second and third constraints, albeit with different conditions. The second forces Z to 0 when the condition is false, except for the case in which both C12701 and C14025 are 1. Otherwise, it is nonbinding. The third constraint forces Z to 0 when both the other binaries are 1; otherwise, it is nonbinding:

$$Z \geq C12701 - C14025, \tag{49}$$

$$Z \leq C12701, \tag{50}$$

$$Z + C12701 + C14025 \leq 2. \tag{51}$$

With these constraints established to properly set the binary variable Z, one can then simply specify the indicator constraint. In CPLEX LP format, it would simply be $Z \geq 1 \rightarrow$ 24 C10181 − 1.0 C11441 ≥ 0.

CPLEX's APIs are well suited to performing this translation. Writing a short C++ program performs the translation of the constraints in question. The resulting model has better problem statistics.

Problem statistics, ns2122603 after addressing rounding, replacing big Ms with indicator constraints

```
Variables             : 23080  [Nneg: 11712, Binary: 11368]
Objective nonzeros    :  5880
Linear constraints    : 28534  [Less: 18486, Greater: 7560, Equal: 2488]
  Nonzeros            : 84604
  RHS nonzeros        : 10088
Indicator constraints :  2520  [Greater: 2520]
  Nonzeros            :  3780
  RHS nonzeros        :  1260

Variables             : Min LB : 0.000000   Max UB : 1.000000
Objective nonzeros    : Min    : 1.000000   Max    : 450.0000
Linear constraints    :
  Nonzeros            : Min    : 1.000000   Max    : 110555.0
  RHS nonzeros        : Min    : 1.000000   Max    : 1.295001e+07
Indicator constraints:
  Nonzeros            : Min    : 1.000000   Max    : 24.00000
  RHS nonzeros        : Min    : 1.000000   Max    : 1.000000
```

With this change, the MIP model remains challenging. But CPLEX no longer declares different objective values optimal, and the associated variability in run time no longer occurs. By removing the most significant source of ill-conditioning, one can now focus on improving MIP performance. As with the previous examination of the model cdma, MIP performance improvements lie outside the scope of this tutorial.

Summarizing the examples, addressing the sources of ill-conditioning or numerical instability significantly improved performance on the models ns1687037, cdma, and ns2122603. The latter two models were unsolved MIPs from the MIPLIB 2010 test set, and additional work remains to tighten the formulation. But, without first addressing the numerical challenges associated with those models as was done in this tutorial, the practitioner has virtually no chance of solving these models to optimality in a reliable manner. Regarding the LPs de063155 and iprob, run time did not pose any challenge, but reliability of the final solution did. Investigating these models using the approaches described in this tutorial helped determine the source of the numerical problems.

9. Summary and Conclusions

Summarizing, here is a list of key observations involving ill-conditioning and finite precision computing.

1. *Ill-conditioning is not specific to a finite precision computing environment*. A model is ill-conditioned if a small change to its input can lead to a much larger change in computed output. Models can be ill-conditioned under perfect precision computing, but finite precision computing introduces additional perturbations to the input that can result in larger perturbations to the output of an ill-conditioned model.

2. *Algorithms cannot be ill-conditioned, but they can be numerically unstable*. Formally, a procedure is numerically stable if its backward error analysis results in small, bounded

errors on all data instances. One can also view an algorithm as unstable if it includes computations that introduce unnecessary round-off error that could be avoided with an alternate implementation of the computation. Examples of such unstable computations include ill-conditioned transformations of the model. In the context of the simplex method, this includes the absence of stability tests to avoid dividing large numbers by much smaller ones in the basis factorization and ratio test.

3. *Finite floating point calculations on numbers of significantly different orders of magnitude typically create more round-off error than calculations on numbers of similar magnitudes*. This is particularly true when division is involved, and numerically stable implementations of algorithms (and, specifically, the *LU* factorizations of the basis matrices in simplex algorithms) take steps to avoid divisions involving denominators orders of magnitude smaller than the numerator. This also implies that the practitioner can improve the stability of a model by avoiding large ratios in the problem data whenever possible.

4. *Most LP and MIP solvers use absolute, rather than relative, tolerances*. Rather than adjust the tolerances based on the problem data or magnitudes of computed values, most (but not all) state-of-the-art LP and MIP solvers scale the model in order to obtain modest orders of magnitude for the problem data values. Depending on the accuracy of the data and the round-off error in representing it, nondefault absolute tolerance settings may be needed.

Based on these observations, here is a list of recommendations that can help practitioners diagnose and resolve ill-conditioning and numerical instability in their LP and MIP models.

1. *Make sure the optimizer does not make algorithmic decisions based on round-off error*. Optimizer tolerances should always distinguish meaningful model data values from values arising from round-off error. In addition, basis condition numbers can magnify the data round-off error above the tolerances. Examine the solution quality, particularly the residuals on primal and dual feasibility, to assess if the round-off error exceeds the feasibility or optimality tolerance.

2. *Avoid single precision calculations of the problem data whenever possible*. Memory saved by calculating problem data in single, rather than double, precision is typically inconsequential for the formulation and subsequent optimization of the model. Calculating model data in single precision increases round-off error in the calculations and can introduce ill-conditioning in basis matrices for the simplex method. If single precision data calculations are unavoidable, consider using larger tolerances than the defaults, which typically are configured for double precision calculations.

3. *Build a list of known sources of ill-conditioning*. Section 4 provides a list of common sources of ill-conditioning in LP and MIP models. This list is by no means comprehensive, and the practitioner's own experiences can help expand the list. Looking for common sources of ill-conditioning may be easier than trying to diagnose the source, particularly on a large model.

4. *Try to reproduce the issue on a smaller instance of the model*. Sometimes the diagnosis of a problem (involving ill-conditioning or other problems) is challenging primarily because of the sheer size of the model. Although the symptom may not be as problematic on a small instance of the model, it may still exist at a lower level and may be much easier to diagnose.

In conclusion, different ill-conditioning metrics may vary in how well they help the diagnosis of LP and MIP models with particular characteristics. By using the metric that provides the most concise information, understanding the sources of perturbations in the problem data representation and optimizer calculations in a finite precision implementation, and building a list of common sources of ill-conditioning, the practitioner can effectively diagnose most optimizer performance problems involving numerically challenging models. Doing so can yield more accurate solutions and faster solve times. This tutorial provided numerous examples of diagnosis and treatment of such models. The author hopes that it has provided readers additional, helpful insight into investigating their own models if they discover indications of ill-conditioning or other numerical challenges.

Acknowledgments

The author wishes to acknowledge all of the CPLEX practitioners over the years, many of whom have provided the wide variety of models that helped him develop much of the material in this paper. The author also wishes to thank the past and present CPLEX development, support, and sales and marketing teams who have contributed to the evolution of the product. Among that group, the author thanks in particular Dr. Tobias Achterberg, Dr. Irv Lustig, and Dr. Roland Wunderling for numerous fruitful discussions on ill-conditioning and numerical stability. The author also wishes to thank Dr. Michael Saunders, Dr. Walter Murray, Dr. Philip Gill, and Dr. Margaret Wright, all members of the Systems Optimization Lab at Stanford in the 1980s, who first exposed the author to the importance of numerically stable computations in optimization. And finally, the author thanks Dr. Lustig, Dr. Dan Steffy of Oakland University, and two anonymous referees for numerous useful comments and suggestions that have improved the content of this paper.

References

[1] D. L. Applegate, W. Cook, S. Dash, and D. G. Espinoza. Exact solutions to linear programming problems. *Operation Research Letters* 35(6):693–699, 2007.

[2] M. Bazaraa, J. Jarvis, and H. Sherali. *Linear Programming and Network Flows*. John Wiley & Sons, Hoboken, NJ, 2005.

[3] V. Chvátal. *Linear Programming*. W. H. Freeman, New York, 1983.

[4] W. Cook, T. Koch, D. E. Steffy, and K. Wolter. A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Mathematical Programming Computation* (3):305–344, 2013.

[5] Cybernet Systems Co. Maple. Accessed August 11, 2014, http://www.maplesoft.com/.

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

[7] G. Dantzig and M. Thapa. *Linear Programming 1: Introduction*. Springer, New York, 1997.

[8] I. I. Dikin. Iterative solution of problems of linear and quadratic programming. *Soviet Mathematics Doklady* 8:674–675, 1967.

[9] I. Duff, A. Erisman, and J. Reid. *Direct Methods for Sparse Matrices*. Clarendon Press, Oxford, UK, 1986.

[10] J. Elble and N. Sahinidis. Scaling linear optimization problems prior to application of the simplex method. *Computational Optimization and Applications* 52(2):345–371, 2012.

[11] A. Fiacco and G. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. John Wiley & Sons, New York, 1968.

[12] FICO. *Xpress-MP Optimization Suite*. Minneapolis, MN, 2013.

[13] P. Gill, W. Murray, M. Saunders, J. Tomlin, and M. Wright. On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method. *Mathematical Programming* 36(2):183–209, 1986.

[14] Gurobi Optimization. *Gurobi Optimizer*. Houston, TX, 2013.

[15] N. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, 1996.

[16] IBM. *ILOG CPLEX*. Incline Village, NV, 2013.

[17] W. Kahan. Numerical linear algebra. *Canadian Mathematical Bulletin* 9:757–801, 1966.

[18] E. Klotz and A. Newman. Practical guidelines for solving difficult linear programs. *Surveys in Operations Research and Management Science* 18(1–2):1–17, 2013.

[19] E. Klotz and A. Newman. Practical guidelines for solving difficult mixed integer linear programs. *Surveys in Operations Research and Management Science* 18(1–2):18–32, 2013.

[20] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. Bixby, E. Danna, et al. MIPLIB 2010. *Mathematical Programming Computation* 3(2):103–163, 2011.

[21] MathWorks. Mathworks. Accessed August 11, 2014, http://www.mathworks.com/help/matlab/.

[22] MOSEK ApS. *MOSEK Optimization Software*. Copenhagen, 2013.

[23] B. Murtagh and M. Saunders. *Minos 5.5. User's Guide*, Systems Optimization Lab, Stanford University, Stanford, CA, 1998.

[24] L. Nader. *Naked Science: Anthropological Inquiry into Boundaries, Power, and Knowledge*. Routledge, New York, 1996.

[25] F. Ordóñez and R. Freund. Computational experience and the explanatory value of condition measures for linear optimization. *SIAM Journal on Optimization* 14(2):307–333, 2003.

[26] R. Rardin. Interior point methods for linear programming. *Optimization in Operations Research*, Chap. 6. Prentice Hall, Upper Saddle River, NJ, 1998.

[27] J. Renegar. Some perturbation theory for linear programming. *Mathematical Programing* 65:73–91, 1994.

[28] E. Rothberg. Private communication. Unpublished, 2007.

[29] H. Sherali and P. Driscoll. Evolution and state-of-the-art in integer programming. *Journal of Computational and Applied Mathematics* 124(1):319–340, 2000.

[30] A. Turing. Rounding-off errors in matrix processes. *Quarterly Journal of Mechanics and Applied Mathematics* 1(3):287–308, 1948.

[31] W. Winston, *Operations Research: Applications and Algorithms*. Brooks/Cole, Thompson Learning, Belmont, CA, 2004.

[32] Wolfram. Mathematica. Accessed August 11, 2014, http://www.wolfram.com/mathematica/.