# DRAFT — Some tricks for converting linearly constrained nonlinear optimization problems into unconstrained problems — DRAFT

**Steve Verrill**, Mathematical Statistician
Forest Products Laboratory, Madison, Wisconsin

## 1 Introduction

This note describes some methods for converting linearly constrained nonlinear optimization problems into unconstrained nonlinear optimization problems. It is is written by a user of nonlinear optimization routines rather than by a numerical analyst. The author's understanding is that some of these methods increase the nonlinearity of a problem, and thus they should be avoided. However, if one only has access to unconstrained routines, these methods permit one to attack constrained problems. The author would appreciate receiving pointers to constrained nonlinear optimization software written in Java. The author would also appreciate tips on better methods for converting constrained problems into unconstrained problems.

## 2 2-sided bounds on a parameter

The constraint is of the form

$$a \leq \theta \leq b \tag{1}$$

where $a, b$ are fixed bounds and $\theta$ is the parameter. In this case we replace the problem of optimizing $f(\theta, \boldsymbol{\beta})$, where $\boldsymbol{\beta}$ is the vector of the remaining parameters, with the problem of optimizing $f(g(\theta'), \boldsymbol{\beta})$ where $\theta'$ is unconstrained, and

$$g(\theta') = (a+b)/2 + (b-a)/2 \times \cos(\theta'). \tag{2}$$

## 3 1-sided bounds on a parameter

The constraint is of the form

$$a \leq \theta \tag{3}$$

where $a$ is a fixed bound and $\theta$ is the parameter. In this case we replace the problem of optimizing $f(\theta, \boldsymbol{\beta})$, where $\boldsymbol{\beta}$ is the vector of the remaining parameters, with the problem of optimizing $f(g(\theta'), \boldsymbol{\beta})$ where $\theta'$ is unconstrained, and

$$g(\theta') = a + (\theta')^2. \tag{4}$$

# 4 Linear equality constraints

The constraints are of the form ($p \leq n$)

$$\begin{aligned} a_{11}\theta_1 + \ldots + a_{1n}\theta_n &= b_1 \\ &\vdots \\ a_{p1}\theta_1 + \ldots + a_{pn}\theta_n &= b_p \end{aligned} \tag{5}$$

or

$$\begin{pmatrix} a_{11} \ldots a_{1n} \\ \vdots \\ a_{p1} \ldots a_{pn} \end{pmatrix} \begin{pmatrix} \theta_1 \\ \vdots \\ \theta_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_p \end{pmatrix}$$

or

$$\mathbf{A}\boldsymbol{\theta} = \mathbf{b}$$

where (without loss of generality) the $p$ rows of $\mathbf{A}$ are linearly independent and

$$\mathbf{A}_p = \text{the first } p \text{ columns of } \mathbf{A}$$

is invertible. In this case we optimize $f(\boldsymbol{\theta})$ by letting $\theta_{p+1}, \ldots, \theta_n$ vary in an unconstrained manner and then obtaining $\theta_1, \ldots, \theta_p$ via

$$\begin{pmatrix} \theta_1 \\ \vdots \\ \theta_p \end{pmatrix} = \mathbf{A}_p^{-1} \left[ \begin{pmatrix} b_1 \\ \vdots \\ b_p \end{pmatrix} - \mathbf{A}_{n-p} \begin{pmatrix} \theta_{p+1} \\ \vdots \\ \theta_n \end{pmatrix} \right]$$

where $\mathbf{A}_{n-p}$ is the matrix formed by the last $n-p$ columns of $\mathbf{A}$.

## 4.1 Aside — Testing for linear independence

In the material above we assume that the rows of $\mathbf{A}$ are linearly independent. This is permissible because it is assumed that we have already removed redundant constraints. The process of identifying and removing redundant constraints is straightforward, but in complicated cases it will require some programming. If the independence of the linear constraints is not immediately apparent from the structure (for example, linear independence is clear if different constraints involve non-overlapping sets of $\theta$'s), then we need to perform a test of the assumption of linear independence. We can do this by performing a singular value decomposition of $\mathbf{A}$.

The singular value decomposition of $\mathbf{A}$ is (here we assume $p \leq n$)

$$\begin{aligned} \mathbf{A} &= \mathbf{U}_{p\times p} \begin{pmatrix} \gamma_1 & 0 & \ldots & 0 & 0 \\ 0 & \gamma_2 & 0 & \ldots & 0 \\ & & \vdots & & \\ 0 & 0 & \ldots & 0 & \gamma_p \end{pmatrix} \mathbf{V}_{p\times n}^T \\ &= \mathbf{U}\mathbf{D}\mathbf{V}^T, \end{aligned}$$

where $\mathbf{U}$ is an orthogonal matrix, the columns of $\mathbf{V}$ are orthonormal to each other, and $\gamma_1 \geq \gamma_2 \geq \ldots \geq \gamma_p \geq 0$ are the singular values of $\mathbf{A}$. Thus the rank of $\mathbf{A}$ is just the number of nonzero singular values. Because of the limitations of computer arithmetic, the null $\gamma_i$'s will not in general be exactly equal to zero. We need to determine a threshold value. If a $\gamma_i$ lies below that threshold, we take it to be equal to zero and conclude that the rows of $\mathbf{A}$ are not linearly independent. A threshold value that is suggested in the numerical analysis literature (for example, Golub and Van Loan (1996)) is

$$\|\mathbf{A}\| \times (\text{the machine precision}).$$

Our experience suggests that

$$\text{threshold} = \|\mathbf{A}\| \times (\text{the machine precision}) \times 10$$

might be a better rule of thumb. Recall that

$$\begin{aligned}
\|\mathbf{A}\| &= \sqrt{\sum_{i=1}^{p}\sum_{j=1}^{n} a_{ij}^2} = \sqrt{\text{trace}(\mathbf{A}\mathbf{A}^T)} \\
&= \sqrt{\text{trace}(\mathbf{U}\mathbf{D}\mathbf{V}^T\mathbf{V}\mathbf{D}\mathbf{U}^T)} \\
&= \sqrt{\text{trace}(\mathbf{U}\mathbf{D}^2\mathbf{U}^T)} \\
&= \sqrt{\text{trace}(\mathbf{D}^2\mathbf{U}^T\mathbf{U})} = \sqrt{\gamma_1^2 + \ldots + \gamma_p^2}.
\end{aligned}$$

For double precision arithmetic on 32 bit computers one would use

$$\sqrt{\gamma_1^2 + \ldots + \gamma_p^2} \times 10^{-15}$$

as the threshold value.

A Java translation of the Linpack singular value decomposition routine can be found at `http://www1.fpl.fs.fed.us/linear_algebra.html`.

## 4.2 Aside — Removing redundant constraints

If the $p$ rows of $\mathbf{A}$ are not linearly independent, then either the linear equality constraints cannot be simultaneously satisfied or some of the constraints are redundant and can be removed. One can determine whether the linear constraints can be simultaneously satisfied, and, at the same time, remove redundant constraints by following a simple algorithm.

Accept the first row of $\mathbf{A}$ as your first set of constraints (assuming, of course, that the $a_{1j}$'s are not all zeros). Then work through rows 2 through $p$. Suppose that you have accepted as many as possible linearly independent constraints from rows 1 through i. Add row $i+1$ to these rows and perform a singular value decomposition of the resulting matrix. If none of the singular values is zero, then add the $i+1$th row to the collection of linearly independent constraints, and continue.

If one of the singular values is zero, then the $i+1$th row is a linear combination of the previously accepted constraints. Let $\mathbf{a}_{i+1}^T$ denote the $i+1$th row of $\mathbf{A}$. We have

$$\mathbf{C}\boldsymbol{\beta} = \mathbf{a}_{i+1} \tag{6}$$

where $\mathbf{c}_1^T, \ldots, \mathbf{c}_j^T$ $(j \leq i)$ are the previously accepted constraints, and the columns of $\mathbf{C}$ are $\mathbf{c}_1, \ldots, \mathbf{c}_j$. We use a linear equation solver (see, for example, the Java versions of the Linpack routines dqrdc and dqrsl in the linear_algebra Java package available at `http://www1.fpl.fs.fed.us/linear_algebra.html`) to obtain $\boldsymbol{\beta}$.

Now the previously accepted constraints have the form

$$\mathbf{C}^T \boldsymbol{\theta} = \mathbf{r} \tag{7}$$

for some (known) vector $\mathbf{r}$. Thus, from (6) and (7), we should have

$$b_{i+1} = \mathbf{a}_{i+1}^T \boldsymbol{\theta} = \boldsymbol{\beta}^T \mathbf{C}^T \boldsymbol{\theta} = \boldsymbol{\beta}^T \mathbf{r}. \tag{8}$$

If $b_{i+1} \neq \boldsymbol{\beta}^T \mathbf{r}$ then the $i+1$th constraint is not compatible with the preceding $i$ constraints and there is no solution to the constrained nonlinear optimization problem. If we do have $b_{i+1} = \boldsymbol{\beta}^T \mathbf{r}$ then the $i+1$th constraint is redundant and can be deleted. We then continue on and consider the $i+2$nd constraint.

### 4.3 Aside — Reordering the columns of A so that $\mathbf{A}_p$ is of full rank

Let us assume that we have removed any redundant constraints so that $\mathbf{A}$ is of full rank $p \leq n$. This still does not guarantee that the first $p$ columns of $\mathbf{A}$ are linearly independent. To ensure that they are, we proceed in a manner similar to the manner in which we proceeded while testing for the linear independence of the rows of $\mathbf{A}$. However, we now focus on the columns of $\mathbf{A}$ rather than the rows. We can again use the singular value decomposition. If a column turns out to be a linear combination of the previously accepted columns, then we, for example, move it to the position of the last column and change the ordering of the $\theta$'s in a corresponding manner. After we have found $p$ linearly independent columns we stop.

## 5 Linear two-sided bounds constraints

The constraints are of the form

$$\begin{array}{c} c_1 \leq a_{11}\theta_1 + \ldots + a_{1n}\theta_n \leq d_1 \\ \vdots \\ c_p \leq a_{p1}\theta_1 + \ldots + a_{pn}\theta_n \leq d_p \end{array} . \tag{9}$$

Define

$$\mathbf{A} \equiv \begin{pmatrix} a_{11} \ldots a_{1n} \\ \vdots \\ a_{p1} \ldots a_{pn} \end{pmatrix} .$$

*If* the $p$ rows of $\mathbf{A}$ are linearly independent, then we can proceed. (The rows of $\mathbf{A}$ are often *not* linearly independent. See section 7 below.) As described in section 4.3 we can reorder the columns of $\mathbf{A}$ so that

$$\mathbf{A}_p = \text{the first } p \text{ columns of } \mathbf{A}$$

is invertible.

Now the constraints embodied in (9) hold if and only if

$$\begin{array}{c} L_1 \leq a_{11}\theta_1 + \ldots + a_{1p}\theta_p \leq U_1 \\ \vdots \\ L_p \leq a_{p1}\theta_1 + \ldots + a_{pp}\theta_p \leq U_p \end{array} \tag{10}$$

where

$$L_i = c_i - (a_{i,p+1}\theta_{p+1} + \ldots + a_{in}\theta_n) \tag{11}$$

and

$$U_i = d_i - (a_{i,p+1}\theta_{p+1} + \ldots + a_{in}\theta_n). \tag{12}$$

Thus we can optimize $f(\theta_1, \ldots, \theta_p, \theta_{p+1}, \ldots, \theta_n)$ subject to the constraints (9) by letting $\theta'_1, \ldots, \theta'_p, \theta_{p+1}, \ldots, \theta_n$ vary in an unconstrained manner and obtaining $\theta_1, \ldots, \theta_p$ via

$$\begin{pmatrix} \theta_1 \\ \vdots \\ \theta_p \end{pmatrix} = \mathbf{A}_p^{-1} \begin{pmatrix} v_1 \\ \vdots \\ v_p \end{pmatrix}$$

where

$$v_i = (L_i + U_i)/2 + (U_i - L_i)/2 \times \cos(\theta'_i).$$

# 6 Linear one-sided bounds constraints

Assume that the constraints are of the form

$$\begin{array}{c} c_1 \leq a_{11}\theta_1 + \ldots + a_{1n}\theta_n \\ \vdots \\ c_p \leq a_{p1}\theta_1 + \ldots + a_{pn}\theta_n \end{array} . \tag{13}$$

Define

$$\mathbf{A} \equiv \begin{pmatrix} a_{11} \ldots a_{1n} \\ \vdots \\ a_{p1} \ldots a_{pn} \end{pmatrix}.$$

*If* the $p$ rows of $\mathbf{A}$ are linearly independent, we can proceed. (The rows of $\mathbf{A}$ are often *not* linearly independent. See section 7 below.) As described in section 4.3 we can reorder the columns of $\mathbf{A}$ so that

$$\mathbf{A}_p = \text{the first } p \text{ columns of } \mathbf{A}$$

is invertible.

Now the constraints embodied in (13) hold if and only if

$$\begin{array}{c} L_1 \leq a_{11}\theta_1 + \ldots + a_{1p}\theta_p \\ \vdots \\ L_p \leq a_{p1}\theta_1 + \ldots + a_{pp}\theta_p \end{array} \tag{14}$$

where

$$L_i = c_i - (a_{i,p+1}\theta_{p+1} + \ldots + a_{in}\theta_n) \tag{15}$$

Thus we can optimize $f(\theta_1, \ldots, \theta_p, \theta_{p+1}, \ldots, \theta_n)$ subject to the constraints (13) by letting $\theta'_1, \ldots, \theta'_p, \theta_{p+1}, \ldots, \theta_n$ vary in an unconstrained manner and obtaining $\theta_1, \ldots, \theta_p$ via

$$\begin{pmatrix} \theta_1 \\ \vdots \\ \theta_p \end{pmatrix} = A_p^{-1} \begin{pmatrix} v_1 \\ \vdots \\ v_p \end{pmatrix}$$

where

$$v_i = L_i + (\theta'_i)^2.$$

# 7 What do you do if the rows of A are not linearly independent?

As we saw in section 4, if the constraints are equality constraints, then we can remove redundant constraints and obtain an $\mathbf{A}$ whose rows are linearly independent. However, if the constraints are inequalities, this is not so easily done. For example, suppose that the $f$ that we are trying to optimize is a function of the variables $x$ and $y$, and suppose that we want to optimize $f$ on the triangle with vertices (0,0), (1,0), and (0,1). Appropriate constraints in this case would be $1 \times x + 0 \times y \geq 0$, $0 \times x + 1 \times y \geq 0$, and $1 \times x + 1 \times y \leq 1$. In this case in the notation of the preceding sections, $\mathbf{a}_3^T = \mathbf{a}_1^T + \mathbf{a}_2^T$. That is, the $\mathbf{a}$ for the third constraint is a linear combination of the $\mathbf{a}$'s corresponding to the first two constraints. However, the third constraint is *not* redundant. Thus we cannot simply delete it as we did in the case of equality constraints. However, in special cases there are still tricks that we can perform.

For example, we can turn the problem posed above into an unconstrained one by a relatively simple reparameterization. First we let a point move along the line segment from (0,0) to (1,0). That is

$$\mathbf{p}_1(\theta_1) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \theta_1 \times \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

where $0 \leq \theta_1 \leq 1$. Then we let a point move along a line segment from (0,1) to $\mathbf{p}_1(\theta_1)$.

$$\mathbf{p}_2(\theta_1, \theta_2) = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \theta_2 \times \left( \mathbf{p}_1(\theta_1) - \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right). \tag{16}$$

where $0 \leq \theta_2 \leq 1$. It is clear (at least after a little thought) that as $\theta_1$ and $\theta_2$ vary over their permitted ranges that $\mathbf{p}_2$ moves over the required region. So we have transformed the three constraints on $x$ and $y$ into two-sided bounds constraints on $\theta_1$ and $\theta_2$. In section 2 we saw how to replace $\theta_1$ and $\theta_2$ with unconstrained $\theta'_1$ and $\theta'_2$ in this case.

This approach should work for any two-dimensional convex figure. That is, one parameter moves a point around the perimeter of the figure and a second parameter moves a point from some fixed vertex toward the first point. This parameterization should yield full coverage of

the interior and boundary of the convex figure. And as noted above these parameters would be subject to 2-sided bounds constraints and could thus be replaced by 2 unconstrained parameters as was done in section 2.

# 8 An example of a nonlinearly constrained optimization problem that could be replaced by an unconstrained optimization problem

This is a problem with which we are all familiar, but it is useful to note that it is an example of the simplification that we are seeking. Suppose that we wanted to optimize the nonlinear function $f$ on the unit sphere (including the interior of the sphere). That is, we want to optimize $f(x, y, z)$ subject to the nonlinear constraint $x^2 + y^2 + z^2 \leq 1$. We simply reparameterize by the usual polar coordinates: $0 \leq \theta < 2\pi$, $0 \leq \gamma \leq \pi$, $0 \leq r \leq 1$, and then, as we saw in section 2, we can replace these parameters in turn with three unconstrained parameters.

# 9 Summary

We have looked at some tricks that permit one to replace constrained nonlinear optimization problems with unconstrained nonlinear optimization problems. If high quality constrained optimization software is available it should probably be used rather than a combination of these tricks and high quality unconstrained optimization software. However, the author is not currently aware of high quality public domain constrained nonlinear optimization software that is written in Java. Thus these tricks in combination with the Java translation of UNCMIN might prove useful. If a reader is aware of constrained nonlinear optimization software written in Java, the author would appreciate hearing about it. Also, if a numerical analyst or user of optimization software knows of other recommended tricks for converting constrained nonlinear optimization problems into unconstrained ones, the author would appreciate hearing about those. Finally, as of 3/13/02 this is an unreviewed (although checked by the author) document, and if a reader detects errors, the author would appreciate hearing about them.

# Reference

Golub, Gene H. and Van Loan, Charles F. (1996). *Matrix Computations, Third Edition.* The Johns Hopkins University Press, Baltimore and London.