NONLINEAR PROGRAMMING

Editor

J. ABADIE

Electricité de France, Paris

and

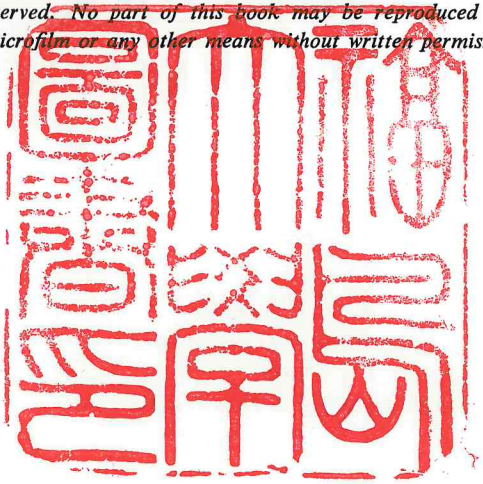
Institut de Statistique de l'Université de Paris

CONTRIBUTORS

S. VAJDA
J. ABADIE
H. W. KUHN
G. B. DANTZIG
R. W. COTTLE
A. WHINSTON
P. WOLFE
E. M. L. BEALE
P. HUARD
M. L. BALINSKI
BUI TRONG LIEU
J. J. MOREAU
J. B. ROSEN

1967

NORTH-HOLLAND PUBLISHING COMPANY — AMSTERDAM



PUBLISHERS:

NORTH-HOLLAND PUBLISHING CO. - AMSTERDAM

SOLE DISTRIBUTORS FOR U.S.A. AND CANADA:

INTERSCIENCE PUBLISHERS, a division of

JOHN WILEY & SONS, INC. - NEW YORK

PRINTED IN THE NETHERLANDS

VII

NUMERICAL METHODS

E. M. L. BEALE

C-E-I-R Ltd., London

Contents

I. THE MINIMIZATION OF A NONLINEAR FUNCTION OF SEVERAL VARIABLES WITHOUT CONSTRAINTS

I.1. Introduction

This first chapter covers some aspects of the problem of minimizing a nonlinear function of several variables in the absence of constraints. This may seem to be out of order, since mathematical programming is specifically concerned with methods of handling constraints. But as a mathematical programming problem becomes more and more nonlinear the fact that it involves constraints becomes less and less important. One is therefore building on a foundation of sand if one goes straight into a discussion of the special problems introduced with the constraints without first reviewing methods of handling the corresponding problems in the absence of constraints. This is especially true since the developments in computer capabilities in the last few years have stimulated research in methods of solving minimization problems without constraints in the same way as they have stimulated research in nonlinear programming.

Almost any numerical problem can be expressed as the minimization of a function of several variables; but this course deals specifically with iterative methods of finding local minima, so it seems logical to restrict our attention to such methods when dealing with unconstrained problems. Both here and later when dealing with programming problems, one is particularly happy with methods for finding local minima when the problem is known to be convex, since the local minimum must then be a global minimum. Many real problems are not convex, but one may still be content with a local minimum for one of 3 reasons:

(a) because the nonconvex elements in the problem are in the nature of small perturbations, so that it seems unlikely that they could introduce unwanted local minima,

(b) because one feels intuitively, perhaps from a knowledge of the real problem represented, that it will not have unwanted local minima (at least if one starts from a good initial solution), or

(c) because the only practical alternative to a local minimum is a completely arbitrary solution.

Mathematicians have concentrated on methods for finding global minima in convex problems. Many of these can also be used to find local minima in nonconvex problems. Others cannot, and are therefore much less useful in practice. The only method discussed in this part of the course that requires any convexity assumptions is decomposition.

Now the essence of an iterative method is that one has a trial solution to the problem and looks for a better one. There are 3 broad classes of such methods for minimization problems. There are quadratic methods, that use estimates of the first and second derivatives of the objective function in the neighbourhood of the current trial solution. There are linear methods, that use first but not second derivatives. And there are directional methods that use no derivatives. (The first derivatives form a vector known as the gradient vector. The second derivatives can be written as a symmetric square matrix known as the Hessian.)

In a sense quadratic methods are the most natural. Any twice differentiable function can be approximated by a quadratic function in the neighbourhood of an unconstrained minimum. To ensure rapid convergence in the closing stages of an iterative procedure, it is therefore natural to require that the procedure should converge in a finite number of steps if the objective function is truly quadratic. This can easily be accomplished with a quadratic method. On the other hand one can use the concepts of "conjugate directions" to achieve the same result with other methods. The methods are a little more complicated, and the number of steps is increased; but the work per step may be much less, since a function of p variables has p first derivatives and $\frac{1}{2}p(p+1)$ second derivatives.

This chapter is concerned with quadratic methods. They are appropriate when second derivatives can be estimated fairly easily, and in particular – for reasons discussed below – when the objective function is a sum of squares. Other methods are discussed by Dr. Wolfe elsewhere in the course.

I.2. Gauss's method for sums of squares problems

Let us now consider problems in which the objective function is a sum of squares. The problem is then to minimize

$$S = \sum_{i=1}^{n} z_i^2,$$

where the z_i are nonlinear functions of the independent variables of the problem. This type of problem arises in a statistical context when one is estimating the parameters of a nonlinear model by the method of least squares. The z_i are then the residuals, i.e. the deviations between the observed and fitted values of the observations. A similar situation arises when solving nonlinear simultaneous equations. This problem may be formulated as one of minimizing the sum of squares of the residuals between the left and right hand sides of the equations. We then know that the minimum value of S is zero, which may be useful, but otherwise the problems are the same.

The importance of this form of the expression for the objective function is that if we get linear approximations, not to S itself but to the individual components z_i, then we can use these to define a quadratic approximation to S.

For if the variables are $x_1, \cdots, x_p$, and our trial solution is given by $x_j = x_{j0}$, then, writing Δx_j for $x_j - x_{j0}$ and z_{i0} for $z_i(x_{10}, \cdots, x_{p0})$, we have

$$S \doteq S_{\text{app}} = \sum_{i=1}^{n} (z_{i0} + a_{i1}\Delta x_1 + \ \ldots \ + a_{ip}\Delta x_p)^2,$$

i.e.

$$S_{\text{app}} = b_0 + 2\sum_{j=1}^{p} b_j \Delta x_j + \sum_{j=1}^{p}\sum_{k=1}^{p} b_{jk}\Delta x_j \Delta x_k,$$

where

$$b_0 = \sum_{i=1}^{n} z_{i0}^2$$

$$b_j = \sum_{i=1}^{n} z_{i0} a_{ij} \qquad (j \neq 0)$$

$$b_{jk} = \sum_{i=1}^{n} a_{ij} a_{ik}.$$

We can now find the values of the Δx_j that minimize S_{app} by solving the "normal equations" of multiple regression in the usual way; and these define our next trial solution to the problem. This approach is due to Gauss.

It should be noted that the quadratic approximation S_{app} is not precisely the one that one would obtain by computing the first and second derivatives of S. For if one expands S as a power series in the Δx_j, one will get some quadratic terms in the expansions for the individual quantities z_i.

In fact if

$$z_i = z_{i0} + \sum_{j=1}^{p} a_{ij}\Delta x_j + \sum_{j=1}^{p}\sum_{k=1}^{p} a_{ijk}\Delta x_j \Delta x_k + \cdots,$$

we find that

$$S = c_0 + 2\sum_{j=1}^{p} c_j \Delta x_j + \sum_{j=1}^{p}\sum_{k=1}^{p} c_{jk}\Delta x_j \Delta x_k + \cdots,$$

where

$$c_0 = \sum_{i=1}^{n} z_{i0}^2$$

$$c_j = \sum_{i=1}^{n} z_{i0} a_{ij} \qquad (j \neq 0)$$

$$c_{jk} = \sum_{i=1}^{n} a_{ij} a_{ik} + 2\sum_{i=1}^{n} z_{i0} a_{ijk}.$$

The question whether one should use the c_{jk} or the b_{jk} as the quadratic terms in the approximate expression for S was discussed by Wilson and Puffer [1].

They point out that, quite apart from the labour of computing the quantities a_{ijk}, there are advantages in using the b_{jk} while optimizing – for example the matrix (b_{jk}) is certainly positive-definite, so the turning point obtained by equating the derivatives to zero is definitely a minimum of the quadratic approximation to S. This is a useful property not necessarily shared by the matrix (c_{jk}). And since $c_j = b_j$, if one has a trial solution for which S_{app} is minimized with all $\Delta x_j = 0$, then S is minimized.

Wilson and Puffer suggest that if in a statistical problem one wants to quote approximate standard errors for the estimates of the x_j, then one should derive these from the matrix (c_{jk}) rather than (b_{jk}). But this problem lies outside the scope of this course. The theoretical statistical problems involved are discussed by Beale [2].

It is, perhaps, surprising that Wilson and Puffer's work has not been rediscovered and republished by someone else in the last 30 years. But as far as I know this point has not received detailed discussion in print since that time.

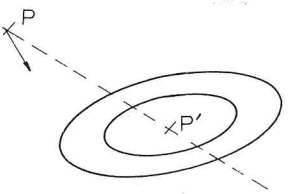
I.3 Interpolation methods

The procedure outlined above will work well for nearly linear problems, particularly if there are not many variables. But otherwise it may fail to converge. One way to overcome this problem is to regard the elementary theory as simply defining a direction from the current trial solution in which to seek the next point. One then interpolates, or extrapolates, along this line to find the point minimizing S. This procedure was suggested by Box [3], and implemented in a computer program for nonlinear estimation on the

IBM 704 computer written under his direction. It is discussed in more detail by Hartley [4].

There are various points in favour of this procedure. Theoretically it is bound to converge if the quantities z_i are differentiable (once), since S starts to decrease as one moves from the present trial solution towards the suggested new one. Furthermore there is often a fair amount of work involved in computing the derivatives (the quantities a_{ij}) at a trial solution, so it is sensible to do some exploration to improve the effectiveness of the step taken as a result of this work.

On the other hand, if one is not going all the way to the point indicated by the quadratic approximation, there is no particular reason to go in this direction. The point is illustrated in 2 dimensions in the following diagram.



The point P defines a trial solution, and the ellipses are contours of constant values of the quadratic approximation S_{app} to S. This quadratic approximation is minimized at the point P′, and the broken line indicates the set of points considered by the Box-Hartley technique as candidates for the next trial solution. But if it turns out to be impossible to move more than a small distance from P in the selected direction before the true value of S starts to increase, then it would seem more logical to move not towards P′ but in the gradient direction of the function S. In our example the gradient direction is indicated by the arrow. This direction may be up to 89.5° away from the direction PP′, and in high-dimensional problems it often is. Marquardt [5] reports that, having monitored this angle for a variety of problems, he found it usually lay between 80° and 90°. The difficulty arising from such a large deviation from the gradient direction is particularly serious for problems where the derivatives a_{ij} are estimated numerically from first differences of the function z_i. One then has to accept errors due to using nonlocal derivatives if the step-length is large, or alternatively rounding-off errors due to taking the difference between two nearly equal quantities if the step-length is small. These errors often result in failure to find a better trial solution at all if one only looks in a direction that even theoretically is inclined at a high angle to the gradient direction.

I.4 The Levenberg method

A suitable method of resolving this difficulty is given by Levenberg [6] and elaborated by Marquardt [5]. The argument is as follows.

Suppose that, instead of trying to minimize S_{app} directly, one asked for the point minimizing S_{app} conditional on it being not more than a given distance from the origin (i.e. the point where all $\Delta x_j = 0$). The logic behind this request is that one wants to remain fairly close to P because the accuracy of the quadratic approximation will tend to decrease the further away one goes. Then the procedure is of course to minimize, not S_{app}, but

$$S_{mod} = S_{app} + \lambda \sum_{j=1}^{p} (\Delta x_j)^2,$$

where the Lagrange multiplier λ must be chosen so that the resulting point is the required distance from P. Computationally this is an easy thing to do; one simply adds λ to all the diagonal elements of the normal equations before solving for the Δx_j. The only real difficulty is to choose λ. It is clear that if $\lambda = 0$ one goes the whole way to the point P', while as $\lambda \to \infty$ one goes to a point an arbitrarily short distance from P along the gradient direction. One is liable to find that an intermediate value produces a result very near to one of these extremes, and it is generally not worthwhile to try out several values of λ at one step. But this difficulty can be mitigated by allowing the program to adjust the value of λ according to the progress of the calculations.

The very best method of modifying λ is not entirely clear. One approach that has proved satisfactory in practice is as follows.

First one defines a standard value of λ, say λ^*, which is a suitably small number. Just how small this is depends on the scales on which the variables are measured. And it should be noted that the variables must be scaled to be commensurable in some sense – this scaling can be done either by a physical knowledge of the problem, or numerically, e.g. by choosing scales so that $b_{jj} = 1$ for all j, in which case a suitable standard value of λ^* may be around 0.0001.

Then one defines a scale of used values of λ equal to $(2^n - 1)\lambda^* b_0$ for $n = 0, 1, 2, \cdots$. The multiplication by b_0 makes the procedure independent of the scale on which the objective function (or dependent variable) is measured.

Then the basis of the rule for choosing λ is the idea that if the new point proves better than the old, then one moves to the new point and reduces

n by 1, while if the new point proves worse than the old, then one remains at the old point and increases n by 1. This seems to be right in principle, since one likes to use as small a value of n as circumstances permit, to give as nearly true quadratic convergence as possible. On the other hand if one is in trouble one must take a smaller step, and this is achieved by increasing n.

But one can have situations in which the solution progresses in a rather erratic manner with a low value of n and in which one can do better with a larger n. This simple rule has therefore been modified in two respects.

Firstly, if n is increased, one notes the reduction in S achieved at the last iteration, and if one achieves an even greater reduction with a higher value of n one increases n by another unit for the next iteration.

Secondly, after an iteration with $n = 0$, i.e. $\lambda = 0$, one always increases n (to 1) for the next iteration, to try the effect of a positive Levenberg parameter.

Other variants of the scheme will be appropriate in other situations. In particular it will often be desirable to use previously calculated values of the derivatives, i.e. of the a_{ij}, only recalculating the z_{i0} at every iteration. One will then recalculate the a_{ij}

(a) if one fails to make progress at some iteration, or

(b) if one has used a set of derivatives for many iterations without converging satisfactorily.

I.5 Miscellaneous practical points

A few miscellaneous practical points are perhaps worth noting. Firstly, it is entirely feasible to combine the use of the interpolation scheme with the Levenberg scheme. This is desirable in 2 situations:

(a) if there are several independent variables, so that it is a nontrivial matter to recalculate the next trial point from the normal equations, or

(b) if it is not feasible (or economical) to store the old derivatives a_{ij}, so that there is appreciably more function-calculation involved in making a complete new step rather than exploring along a line.

Secondly, it is desirable to solve the normal equations by inverting the matrix of sums of squares and products, pivoting on the diagonal elements as in step-wise multiple linear regression. One need only work with half the matrix if one keeps it symmetrical by adopting the appropriate sign convention, indicated for example by Stiefel [7] on page 65. The advantage of this

procedure is that one can (and should) refuse to pivot on a diagonal element that has become very small. This just means that one does not consider changing the trial value of this particular variable while there is little independent evidence concerning its value. This difficulty is likely to arise in practice only when the Levenberg parameter is zero.

Thirdly, it is often desirable to take the quantities z_i in groups, since the formulae for their values in terms of the independent variables may have common features that would otherwise have to be recalculated. This situation arises in particular if the quantities in a group refer to the same physical quantity at different times.

Finally, it should be borne in mind that one does not necessarily have to make a sharp choice between computing derivatives theoretically and numerically. For example one may decide that most derivatives can most easily be computed numerically in spite of the rounding-off problems involved, but some may be self-evidently zero, while others can be computed in a "wholesale fashion", using intermediate results obtained while computing other derivatives.

II. AN INTRODUCTION TO BEALE'S METHOD OF QUADRATIC PROGRAMMING

II.1 Introduction

Historically the first venture into the theory of nonlinear programming has been to the problem known as quadratic programming. This name is restricted to the specific problem of minimizing a convex quadratic objective function of variables subject to linear constraints. In accordance with the philosophy outlined in Chapter I, I would extend this definition to include the problem of finding a local minimum of any quadratic objective function of variables subject to linear constraints. But many people will object to this generalization on the grounds that most of the methods that have been proposed for quadratic programming require that the objective function be convex: indeed some require that it be strictly convex.

Many methods for quadratic programming have been published. Künzi and Krelle [8] discuss 7 of them, in addition to 3 versions of the application of gradient methods to quadratic programming. Since that time a number of variants of Wolfe's [9] method have been published.

I am not going to make any pretence of being impartial between these methods. I will content myself with explaining how my own method works and why I think it has great advantages over all other methods.

I think it is natural that quadratic programming should have received so much attention from theorists. Mathematically, it is the natural first extension beyond the realm of linear programming; and it has the great mathematical advantage of being solvable in a finite number of steps. In practice it has not been used very extensively, and I think there are 3 main reasons for this.

(a) In practical problems the nonlinearities in the constraints are often more important than the nonlinearities in the objective function.

(b) When one has a problem with a nonlinear objective function there are generally rather few variables that enter the problem nonlinearly. But most methods for quadratic programming work no more simply in this case

than with a completely general quadratic objective function, and

(c) Quadratic perturbations on a basically linear problem may well not be convex, although one may be fairly confident that, since they are perturbations, they will not introduce local minima. But most methods for quadratic programming cannot cope with such problems.

Of these difficulties, the last does not apply to any version of Beale's algorithm. The second does not apply to the second version of the algorithm discussed in the next chapter. Quadratic programming may therefore become more widely used now that this algorithm has been implemented in at least one general mathematical programming system for a large computer.

The first difficulty is more fundamental. But one should note that the natural way to solve an unconstrained minimization problem is to make the simplest meaningful local approximation. This is to make linear approximations to the derivatives of the objective function in the neighbourhood of one's trial solution, i.e. to make a quadratic approximation to the objective function itself. Applying the same philosophy to constrained minimization problems, one would naturally make linear approximations to the constraints and quadratic approximations to the objective function. I will return to the point in Chapter VII, when in particular I will show how one can throw the local nonlinearities in the constraints into the objective function. This is an essential part of the implementation of this philosophy.

Having indicated the status of quadratic programming from the point of view of someone interested in applications of nonlinear programming, I now turn to Beale's method for this problem. This exists in 2 versions. The first was originally published in Beale [10] and amplified in Beale [11]. The second is simply a streamlined way of organizing the computations following the logic of the first method. It was introduced in Beale [11] on page 236, and is discussed in detail in the following chapter. Perhaps the most important thing about this version of the algorithm is that it is particularly convenient for the product form of inverse basis method of carrying out the simplex calculations. This aspect is covered in the chapter on the inverse matrix method. This chapter is concerned with the basic logic of Beale's method, which can be explained most easily in terms of the original version.

II.2 The simplex method

I will start by describing the simplex method in a way that applies to its use in linear programming and to its use in Beale quadratic programming.

Let me point out that this differs from Wolfe's version of the simplex method for quadratic programming. Wolfe [12] remarks that Beale's method has a better claim to this name. There are 2 main reasons for this.

(a) It was published some years earlier as an application of the simplex method to quadratic programming, and

(b) it reduces to the ordinary simplex method when the objective function is linear. On the other hand the development of Wolfe's method to a point where it can handle purely linear problems yields an algorithm in which each step of the simplex method has to be performed twice (once on the original problem and once on its transpose). This last point is important, since it suggests that Wolfe's method will take about twice as many steps as Beale's on a nearly linear problem. But this is by the way.

Suppose that we want to find a minimum, or at least a local minimum, of some objective function C of n variables x_j that must be nonnegative and satisfy the m $(< n)$ linearly independent equations, or constraints

$$\sum_{j=1}^{n} A_{ij}x_j = B_i \qquad (i = 1, \cdots, m).$$

If these constraints are not inconsistent, we can find a basic feasible solution in which m of these variables takes positive values and the others are zero. We can then use the constraints to express the basic variables, i.e. those that are positive, in terms of the nonbasic variables. We will then have

$$x_h = a_{h0} + \sum_{k=1}^{n-m} a_{hk}z_k \qquad (h = j_1, \cdots, j_m),$$

where z_k denotes $x_{j_{m+k}}$.

It is customary to write these equations in tableau form, corresponding to the coefficients of the equations when the z_k are written on the left hand sides. The above form was introduced in Beale [13]. A. W. Tucker has recently suggested writing the equations in the form

$$x_h = a_{h0} + \sum_{k=1}^{n-m} a_{hk}(-z_k),$$

so that the numerical coefficients have the same signs as in the traditional tableau. This modification obviously has merit, but I find it rather cumbersome and have not adopted it.

Let us now return to the description of the logic of the simplex method.

In the present trial solution of the problem, the basic variables x_h equal a_{h0}, and the nonbasic variables are all zero.

One can now use the equations for the basic variables to express the objective function C in terms of the nonbasic variables. And one can then consider the partial derivative of C with respect to any one of them, say z_p, assuming that all the other nonbasic variables remain fixed and equal to zero.

If now $\partial C/\partial z_p \geqq 0$, then a small increase in z_p, with the other nonbasic variables held equal to zero, will not reduce C. But if $\partial C/\partial z_p < 0$, then a small increase in z_p will reduce C. If C is a linear function, then $\partial C/\partial z_p$ is constant, and it will be profitable to go on increasing z_p until one has to stop to avoid making one of the basic variables negative.

If C is a nonlinear function with continuous derivatives, then it will be profitable to go on increasing z_p until either

(a) one has to stop to avoid making some basic variable, say x_q, negative, or

(b) $\partial C/\partial z_p$ vanishes and is about to become positive.

In case (a), which is the only possible case when C is linear, one changes the basis by making x_q nonbasic in place of z_p, and uses the equation

$$x_q = a_{q0} + \sum_{l=1}^{n-m} a_{ql} z_l$$

to substitute for z_p in terms of x_q and the other nonbasic variables throughout the constraints and also the expression for C.

In case (b) one is in trouble if C is an arbitrary function. But if C is a quadratic function, then $\partial C/\partial z_p$ is a linear function of the nonbasic variables.

The way in which the function C is expressed is the primary difference between the first, or theoretical, and the second, or practical, version of the algorithm. The theoretically simplest way to express C is as symmetric matrix (c_{kl}) for $k, l = 0, 1, \cdots, n-m$, such that

$$C = \sum_{k=0}^{n-m} \sum_{l=0}^{n-m} c_{kl} z_k z_l,$$

where $z_0 = 1$ and $z_1, \cdots, z_{n-m}$ denote the nonbasic variables.

Then

$$\tfrac{1}{2} \frac{\partial C}{\partial z_p} = c_{p0} + \sum_{k=1}^{n-m} c_{pk} z_k,$$

and if this quantity becomes positive, as z_p is increased keeping the other

nonbasic variables equal to zero, before any basic variable goes negative, then one defines a new nonbasic variable

$$u_t = c_{p0} + \sum_{k=1}^{n-m} c_{pk} z_k$$

(where the subscript t simply indicates that this is the t^{th} such variable introduced into the problem).

One then makes u_t the new nonbasic variable, using the above equation to substitute for z_p in terms of u_t and the other nonbasic variables throughout the constraints and the expression for C.

Note that if z_p is an x-variable, then there will be one more basic x-variable after this iteration than before.

The mechanics for substituting for z_p in the expression for C will be discussed later. Let us first concentrate on the theory of the procedure.

Note that u_t is not restricted to nonnegative values. It is therefore called a free variable, as opposed to the original x-variables which are called restricted variables. But there is no objection to having a free nonbasic variable. One simply has to remember that if $\partial C/\partial u_t > 0$, then C can be reduced by making u_t negative – or alternatively by replacing u_t by the variable $v_t = -u_t$ and increasing v_t in the usual way.

There is one other point about these free variables. Once a free variable has been removed from the set of nonbasic variables one can forget about it. There are only 2 reasons for keeping track of the expressions for the basic variables in terms of the others. One is to know their values in the trial solution when it is obtained. The other, and much more fundamental, reason is to prevent such variables from surreptitiously becoming negative. Neither reason applies to any basic free variable in this type of problem.

There remains the problem of explaining why one should make this particular choice of free variable. It is obviously convenient to have a set of nonbasic variables that all vanish at the present trial solution, because the values of the basic variables are then simply represented by the constant terms in the transformed equations. But any expression of the form

$$c_{p0} + c_{pp} z_p + \sum_{k \neq p} a_k z_k$$

would satisfy this condition, and it might seem much simpler to put all $a_k = 0$.

The mathematically correct way to justify this is in terms of conjugate directions. One wants to change the nonbasic variables such that if the values of other nonbasic variables are subsequently changed, keeping $u_t = 0$, the

direction of motion is conjugate to the step already taken with respect to the objective function. In other words one would like to ensure that, having made $\partial C/\partial z_p = 0$, this derivative will remain zero. If this could be achieved, then the solution to the problem could be reached in at most $n-m$ steps. Unfortunately these intentions are frustrated whenever one comes up against a new constraint. This means that one has to work on a new face of the feasible region and to start again setting up conjugate directions in this new face. In spite of these hazards, the process must terminate in a finite number of steps, as we now show.

II.3 Proof of convergence

To prove convergence in a finite number of steps, we first make the rules of procedure a little more specific. If there is any free variable that was introduced before a restricted variable was last made nonbasic, then we insist that some such free variable be removed from the set of nonbasic variables at the next iteration. I have not bothered to consider whether this condition is really necessary, since it is obviously reasonable to remove such a variable because it cannot remain nonbasic in the final solution.

We now make a further definition. We say that the objective function C is in "standard form" if the linear terms in its expression contain no free variable. When C is in standard form, its value in the present trial solution, c_{00}, is a stationary value of C subject to the restriction that all the present nonbasic restricted variables take the value zero (without any restrictions on the sign of the basic variables). So there can be only one such value for any set of nonbasic restricted variables. Now we know that C decreases at every step. So it can never return to a standard form with the same set of nonbasic restricted variables, even with a different set of free variables. There is only a finite number of possible sets of nonbasic restricted variables, so the algorithm must terminate if it always reaches a standard form in a finite number of steps when it is not already in standard form.

To prove this, we note that whenever C is not in standard form a free variable will be removed from the set of nonbasic variables. So s, the number of nonbasic free variables, cannot increase. Further, if the new nonbasic variable is free, it is easy to show that the off-diagonal elements in the new expression for C in the row and column associated with the new nonbasic variable must vanish. It follows that C does not contain a linear term in this variable, and furthermore C can never contain a linear term in this variable unless some other restricted variable becomes nonbasic, thereby decreasing s.

Therefore, if C is not in standard form, and $s = s_0$, say, then s cannot

increase and it must decrease after at most s_0 steps unless C meanwhile achieves standard form. Since C is always in standard form when $s = 0$, the required result follows.

II.4 Updating the tableau

There is one loose end in the above procedure that should be discussed theoretically before we turn to a numerical example. This concerns the updating of the tableau from one iteration to the next. There is no difficulty about the expressions for the constraints, but the objective function must also be updated. It is unprofitable to discuss this problem in detail, because it does not arise in this form in the practical version of the algorithm to be given in my next chapter. The procedure suggested in Beale [10] and Beale [11] can be expressed algebraically by saying that, starting from the expression $x'Cx$, one first substitutes for the final x the vector y of new nonbasic variables, deducing the coefficients of C^* where

$$C = x'C^*y,$$

and then substitutes for the initial x'. This would never be very convenient in a computer, since it involves operating on both the rows and columns of the matrix. I am indebted to Dr. D. G. Prinz for pointing this out, and for pointing out that the solution is to use the algebraic expressions for the combined effects of these transformations, which are given in Beale [10].

These expressions are as follows:

If we denote the pivotal column by the subscript q, and if the expression for the new basic variable z_p in terms of the new nonbasic variables is

$$z_p = e_0 + e_q z_q + \sum_{k \neq q} e_k z_k,$$

(where z_q denotes the new nonbasic variable replacing z_p), then the new coefficients (c'_{kl}) are given in terms of the old coefficients c_{kl} and the e_k as follows:

$$\begin{aligned} c'_{qq} &= c_{qq} e_q^2 \\ c'_{kq} &= c'_{qk} = c_{kq} e_q + c_{qq} e_q e_k, \\ c'_{kl} &= c_{kl} + c_{kq} e_l + c_{ql} e_k + c_{qq} e_k e_l, \end{aligned}$$

where $k, l \neq q$.

These expressions can be written in a more elegant form if we write

$$\begin{aligned} c_q^* &= \tfrac{1}{2} c_{qq} e_q, \\ c_k^* &= c_{kq} + \tfrac{1}{2} c_{qq} e_k, \qquad \text{for } k \neq q. \end{aligned}$$

For then we have

$$c'_{qq} = 2c^*_q e_q,$$
$$c'_{kq} = c'_{qk} = c^*_k e_q + c^*_q e_k,$$
$$c'_{kl} = c_{kl} + c^*_k e_l + c^*_l e_k.$$

II.5 A numerical example

We conclude this chapter with a numerical example.

The example given by Beale [11] illustrates most of the features of the method and has a simple geometrical interpretation. A similar example is not repeated here, because it seems better to produce an example that illustrates a special, though not very attractive, feature of the method. This feature is the necessity to sometimes eliminate more than one free variable from the set of nonbasic variables before restoring the problem to standard form. While this is being done one has no chance of reaching the optimum solution to the problem. On the other hand one is making progress, and in particular one may move onto a better face of the feasible region without having to complete the optimization on the present face.

To illustrate this situation we have to go into (at least) 3 dimensions. We therefore consider the following problem.

Minimize

$$C = 9 - 8x_1 - 6x_2 - 4x_3 + 2x_1^2 + 2x_2^2 + x_3^2 + 2x_1x_2 + 2x_1x_3,$$

subject to the constraints

$$x_1 \geqq 0, \qquad x_2 \geqq 0, \qquad x_3 \geqq 0,$$
$$x_1 + x_2 + 2x_3 \leqq 3.$$

We start by introducing a slack variable x_4, and write

$$x_4 = 3 - x_1 - x_2 - 2x_3.$$

We can also express C in a form that displays the coefficients (c_{kl}) and at the same time can be read as an intelligible equation as follows:

$$\begin{aligned} C = {} & (\ 9 - 4x_1 - 3x_2 - 2x_3) \\ & + (-4 + 2x_1 + x_2 + x_3)x_1 \\ & + (-3 + x_1 + 2x_2 \qquad)x_2 \\ & + (-2 + x_1 \qquad + x_3)x_3. \end{aligned}$$

We see that C can be decreased by increasing x_1. This will decrease

x_4, but x_4 remains positive until $x_1 = 3$. But $\frac{1}{2}\partial C/\partial x_1 = -4+2x_1+x_2+x_3$, and this becomes zero if $x_1 = 2$. So we introduce the variable

$$u_1 = -4+2x_1+x_2+x_3$$

as our new nonbasic variable.

We then have

$$x_1 = 2+\tfrac{1}{2}u_1-\tfrac{1}{2}x_2-\tfrac{1}{2}x_3$$
$$x_4 = 1-\tfrac{1}{2}u_1-\tfrac{1}{2}x_2-\tfrac{3}{2}x_3.$$

To deduce the new expression for C we note that

$$q = 1, \quad e_0 = 2, \quad e_1 = \tfrac{1}{2}, \quad e_2 = -\tfrac{1}{2}, \quad e_3 = -\tfrac{1}{2},$$
$$\tfrac{1}{2}c_{qq} = 1, \quad c_0^* = -2, \quad c_1^* = \tfrac{1}{2}, \quad c_2^* = \tfrac{1}{2}, \quad c_3^* = \tfrac{1}{2},$$

and $C = (\ 1 \qquad -\ x_2 \qquad)$
$+\ (\qquad \tfrac{1}{2}u_1 \qquad\qquad)u_1$
$+\ (-1 \qquad +\tfrac{3}{2}x_2-\tfrac{1}{2}x_3)x_2$
$+\ (\qquad\qquad -\tfrac{1}{2}x_2+\tfrac{1}{2}x_3)x_3.$

We now note that C can be decreased by increasing x_2. Again we are stopped by the derivative going to zero before any basic variable becomes negative. So we write

$$u_2 = -1 \quad +\tfrac{3}{2}x_2 \quad -\tfrac{1}{2}x_3.$$

Introducing u_2 as a nonbasic variable in place of x_2, we have

$$x_2 = \tfrac{2}{3} \qquad\qquad +\tfrac{2}{3}u_2 \qquad +\tfrac{1}{2}x_3$$
$$x_1 = \tfrac{5}{3} \qquad +\tfrac{1}{2}u_1 \qquad -\tfrac{1}{3}u_2 \qquad -\tfrac{2}{3}x_3$$
$$x_4 = \tfrac{2}{3} \qquad -\tfrac{1}{2}u_1 \qquad -\tfrac{1}{3}u_2 \qquad -\tfrac{5}{3}x_3.$$

$$q = 2, \quad e_0 = \tfrac{2}{3}, \quad e_1 = 0, \quad e_2 = \tfrac{2}{3}, \quad e_3 = \tfrac{1}{3},$$
$$\tfrac{1}{2}c_{qq} = \tfrac{3}{4}, \quad c_0^* = -\tfrac{1}{2}, \quad c_1^* = 0, \quad c_2^* = \tfrac{1}{2}, \quad c_3^* = -\tfrac{1}{4},$$

$C = (\tfrac{1}{3} \qquad\qquad\qquad -\tfrac{1}{3}x_3)$
$+\ (\qquad \tfrac{1}{2}u_1 \qquad\qquad\qquad)u_1$
$+\ (\qquad\qquad\qquad \tfrac{2}{3}u_2 \qquad)u_2$
$+\ (-\tfrac{1}{3} \qquad\qquad\qquad +\tfrac{1}{3}x_3)x_3.$

We now note that C can be decreased by increasing x_3. But this time we are stopped by x_4 going to zero, when $x_3 = \frac{2}{5}$.

So we write

$$\begin{aligned} x_3 &= \tfrac{2}{5} - \tfrac{3}{10}u_1 - \tfrac{1}{5}u_2 - \tfrac{3}{5}x_4 \\ x_2 &= \tfrac{4}{5} - \tfrac{1}{10}u_1 + \tfrac{3}{5}u_2 - \tfrac{1}{5}x_4 \\ x_1 &= \tfrac{7}{5} + \tfrac{7}{10}u_1 - \tfrac{1}{5}u_2 + \tfrac{2}{5}x_4 \end{aligned}$$

$$\begin{aligned} q = 3, \quad & e_0 = \tfrac{2}{5}, \quad e_1 = -\tfrac{3}{10}, \quad e_2 = -\tfrac{1}{5}, \quad e_3 = -\tfrac{3}{5}, \\ \tfrac{1}{2}c_{qq} = \tfrac{1}{6}, \quad & c_0^* = -\tfrac{4}{15}, \quad c_1^* = -\tfrac{1}{20}, \quad c_2^* = -\tfrac{1}{30}, \quad c_3^* = -\tfrac{1}{10}, \end{aligned}$$

$$\begin{aligned} C = &(\tfrac{3}{25} + \tfrac{3}{50}u_1 + \tfrac{1}{25}u_2 + \tfrac{3}{25}x_4) \\ &+(\tfrac{3}{25} + \tfrac{53}{100}u_1 + \tfrac{1}{50}u_2 + \tfrac{3}{50}x_4)u_1 \\ &+(\tfrac{1}{25} + \tfrac{1}{50}u_1 + \tfrac{34}{50}u_2 + \tfrac{1}{25}x_4)u_2 \\ &+(\tfrac{3}{25} + \tfrac{3}{50}u_1 + \tfrac{1}{25}u_2 + \tfrac{3}{25}x_4)x_4 . \end{aligned}$$

We now have to remove both u_1 and u_2 from the set of nonbasic variables. Starting with u_1, we must decrease this. $\partial C/\partial u_1$ becomes zero when $u_1 = -\frac{6}{53}$, and all basic variables are still positive.

So we write

$$\begin{aligned} u_3 &= \tfrac{3}{50} + \tfrac{53}{100}u_1 + \tfrac{1}{50}u_2 + \tfrac{3}{50}x_4 \\ \text{i.e. } u_1 &= -\tfrac{6}{53} + \tfrac{100}{53}u_3 - \tfrac{2}{53}u_2 - \tfrac{6}{53}x_4 \\ x_3 &= \tfrac{23}{53} - \tfrac{30}{53}u_3 - \tfrac{10}{53}u_2 - \tfrac{30}{53}x_4 \\ x_2 &= \tfrac{43}{53} - \tfrac{10}{53}u_3 + \tfrac{32}{53}u_2 - \tfrac{10}{53}x_4 \\ x_1 &= \tfrac{70}{53} + \tfrac{70}{53}u_3 - \tfrac{12}{53}u_2 + \tfrac{17}{53}x_4 \end{aligned}$$

$$\begin{aligned} q = 1, \quad & e_0 = -\tfrac{6}{53}, \quad e_1 = +\tfrac{100}{53}, \quad e_2 = -\tfrac{2}{53}, \quad e_3 = -\tfrac{6}{53}, \\ \tfrac{1}{2}c_{qq} = \tfrac{53}{100}, \quad & c_0^* = \tfrac{3}{100}, \quad c_1^* = \tfrac{1}{2}, \quad c_2^* = \tfrac{1}{100}, \quad c_3^* = \tfrac{3}{100}, \end{aligned}$$

$$\begin{aligned} C = &(\tfrac{6}{53} + \tfrac{2}{53}u_2 + \tfrac{6}{53}x_4) \\ &+(\tfrac{100}{53}u_3)u_3 \\ &+(\tfrac{2}{53} + \tfrac{36}{53}u_2 + \tfrac{2}{53}x_4)u_2 \\ &+(\tfrac{6}{53} + \tfrac{2}{53}u_2 + \tfrac{6}{53}x_4)x_4 . \end{aligned}$$

This trial solution is of some theoretical interest, since it is one that other methods, such as Wolfe's, manage to bypass. Some methods for quadratic programming have been put forward as variants of Beale's algorithm. One good test of the validity of this claim is whether the method passes through this trial solution on this problem.

Our last step is to remove the variable u_2 from the set of nonbasic variables. We again have to decrease the variable being removed. And again we can go to the point where the partial derivative vanishes.

So we put

$$\begin{array}{lcrrrr} & u_4 & = & \frac{2}{53} & & +\frac{360}{530}u_2 & +\frac{2}{53}x_4 \\ \text{i.e.} & u_2 & = & -\frac{1}{18} & & +\frac{530}{360}u_4 & -\frac{1}{18}x_4 \\ & x_3 & = & \frac{4}{9} & -\frac{300}{530}u_3 & -\frac{5}{18}u_4 & -\frac{5}{9}x_4 \\ & x_2 & = & \frac{7}{9} & -\frac{100}{530}u_3 & +\frac{8}{9}u_4 & -\frac{2}{9}x_4 \\ & x_1 & = & \frac{4}{3} & +\frac{700}{530}u_3 & -\frac{1}{3}u_4 & +\frac{1}{3}x_4 \end{array}$$

$q = 2$, $c_0 = -\frac{1}{18}$, $e_1 = 0$, $e_2 = \frac{530}{360}$, $e_3 = -\frac{1}{18}$,

$\frac{1}{2}c_{qq} = \frac{18}{530}$, $c_0^* = \frac{1}{530}$, $c_1^* = 0$, $c_2^* = \frac{1}{2}$, $c_3^* = \frac{1}{530}$,

$$\begin{array}{rrrr} C = (\frac{1}{9} & & & +\frac{1}{9}x_4) \\ +(& \frac{100}{530}u_3 & &)u_3 \\ +(& & \frac{530}{360}u_4 &)u_4 \\ +(\frac{1}{9} & & & +\frac{1}{9}x_4)x_4 . \end{array}$$

This represents the final solution, $C = \frac{1}{9}$, with $x_1 = \frac{4}{3}$, $x_2 = \frac{7}{9}$ and $x_3 = \frac{4}{9}$.

III. THE PRACTICAL VERSION OF BEALE'S METHOD OF QUADRATIC PROGRAMMING

III.1. Introduction

At the beginning of the last chapter I stressed the importance of having a method for quadratic programming that:

(a) would find a local minimum of a nonconvex function,

and (b) would be specially simple to operate if there were only a few quadratic terms.

This chapter starts with a few remarks about local minima. This is followed by a discussion of the extent to which the algorithm presented in the previous chapter meets the second of these criteria. The practical version of this algorithm, originally presented rather briefly on page 236 of Beale [11], is then introduced. It is illustrated by the same numerical example as that solved in the previous chapter. The practical version of the algorithm can be applied more easily using the inverse matrix method. This important point will be taken up in the chapter devoted to the inverse matrix method.

III.2. Local minima and virtual local minima

In principle the algorithm described in the previous chapter will find a local minimum of a nonconvex quadratic objective function. The objective function is reduced at every step. At no stage have we assumed that the diagonal elements of the matrix (c_{kl}) must be positive. If we are increasing z_p and the element c_{pp} is negative or zero, then there is no danger of $\partial C/\partial z_p$ vanishing and threatening to go positive, but this is no disadvantage. Note that if we do introduce a free nonbasic variable it must have a positive squared term in the expression for C, and this coefficient remains unaltered unless some new restricted variable becomes nonbasic, in which case this free variable will in due course be removed from the set of nonbasic variables. So the algorithm cannot terminate with negative quadratic coefficients.

Unfortunately there is a theoretical danger of termination at a point that is not a local minimum. One might have some restricted nonbasic variable, say x_p, with a reduced cost of zero, i.e. such that $\partial C/\partial x_p = 0$.

The algorithm may then terminate, but if the objective function is not convex an increase in this variable might be profitable.

This is an example of general difficulty when looking for local minima. It is convenient to define a point that is not strictly a local minimum but could easily be taken for one in a numerical minimization process as a "virtual local minimum". I am grateful to my colleagues at the NATO Advanced Study Institute, and in particular E. H. Jackson, H. I. Scoins and A. C. Williams, for help in sorting out a satisfactory definition of a virtual local minimum. Beale [11] defines it as a point that could be made into a local minimum by an arbitrarily small change in the coefficients of the objective function in a quadratic programming problem. But this definition cannot be applied to more general nonlinear programming problems, and in particular to problems involving nonlinear constraints. Jackson points out that in a minimization problem without constraints it is natural to define a virtual local minimum as a point where the gradient vector vanishes and the Hessian is positive semi-definite. This can be expressed as a point that can be turned into a local minimum by arbitrarily small changes in the linear and quadratic terms of the objective function. But other problems can arise with nonlinear constraints, and it is desirable to extend the class of perturbations permitted to include arbitrarily small changes in the constant terms of constraints.

Note that the assumption that we can make arbitrarily small changes in the constant terms of the constraints, and in the linear terms of the objective function, means in the terminology of the simplex method that we do not have to worry about either primal or dual degeneracy.

So much for generalities at this point. Returning to the subject of quadratic programming, we find that it is possible to extend the algorithm so that it can only terminate at a true local minimum. Whether this is worthwhile in practice, and whether it could cause cycling, I am not sure. But, for the record, the procedure will now be outlined.

The partial derivative $\partial C/\partial z_p$ is given by

$$2(c_{p0} + \sum_{k=1}^{n-m} c_{pk} z_k).$$

Normally one terminates the algorithm if the objective function is in standard form, and all $c_{p0} \geqq 0$. But in the modified algorithm one will not stop if some $c_{p0} = 0$ unless $c_{pk} \geqq 0$ for all k such that $c_{k0} = 0$. It is easy to see that if one does stop under these conditions the trial solution must be a local minimum. But if the conditions are not satisfied the trial

solution may be only a virtual local minimum. If $c_{p0} = 0$ and $c_{pp} < 0$ one can immediately decrease C by increasing z_p as far as possible. If $c_{p0} = 0$ and $c_{pp} = 0$, one can decrease C by first increasing z_p and then increasing some other nonbasic variable for which $c_{q0} = 0$ and $c_{pq} < 0$. If $c_{p0} = 0$ and $c_{pp} > 0$ one cannot increase z_p immediately without increasing C. But by making z_p nonbasic and introducing a new free nonbasic variable one may produce a situation in which several of the present nonbasic variables can be increased together so as to reduce C.

III.3. The compactness of Beale's algorithm

One of the features of Beale's algorithm is that the size of the tableau fluctuates. This could be a nuisance in a computer program, though it is not necessarily so, if the matrix is stored by rows on magnetic tape (or other convenient backing store). In any case it is of some interest to have an upper bound on the number of rows required.

For an arbitrary problem, one might have every single x-variable nonbasic. One then needs n rows to store the expressions for these variables, plus the e row containing the expression for the variable just leaving the set of nonbasic variables (which might conceivably be a free variable and therefore not among the n x-variables), plus the c^* row, plus the expression for C.

The situation will be better if there are only a few quadratic terms. This means among other things that the rank r of the matrix of the purely quadratic terms in the objective functions must be much less than its maximum possible value of $n-m$. Now this rank will be unaffected by any change of basis. And this proves that one cannot have more than r nonbasic free variables at any iteration. For before one could introduce an $(r+1)^{st}$ such variable, C would have to be in standard form with r nonbasic free variables. There would therefore be r nonzero diagonal elements in the quadratic part of C with nonzero off-diagonal elements in the same row or column. This implies that all the remaining elements of C referring to quadratic terms must vanish if the rank is to be not more than r. And this in turn implies that the next new nonbasic variable (if any) must be a restricted variable.

So there must be at least $n-m-r$ restricted nonbasic variables. And there cannot therefore be more than $m+r$ restricted basic variables.

This is some consolation, but in a typical linear programming problem there are many more variables than constraints. So if the objective function is stored as an $(n-m) \times (n-m)$ matrix then most of the tableau will be taken

up by these coefficients. It would be possible to use the symmetry of the matrix and work with only half of it (i.e. the coefficients c_{kl} for $k \leqq l$). The coding for this would come quite easily to anyone who had coded the stepwise multiple regression procedure to work in this way. But even saving half the matrix leaves the problem in an unwieldy form if one has, say, 10 quadratic variables and 200 linear ones.

III.4. Representing the objective function as a sum of squares

As with so many problems in mathematical programming, the important decision here is not so much the choice of basic logic for the iterative solution procedure as the choice of how to keep track of the numbers required to implement it.

Now it is clear that the most compact way to represent a quadratic function of low rank r is as a sum or difference of squares.

We can write

$$C = \lambda_0 + \tfrac{1}{2}\sum_{i=1}^{r_1} \lambda_i^2 - \tfrac{1}{2}\sum_{i=r_1+1}^{r} \lambda_i^2, \tag{3.1}$$

where the λ_i are linear functions of the variables of the problem, which can be updated from iteration to iteration in the same way as any other rows of the problem. (It is to be understood that the second summation is vacuous if C is convex, i.e. if $r_1 = r$, and the first summation is vacuous if C is concave, i.e. if $r_1 = 0$.)

It turns out that this is a reasonably convenient procedure even if r is not small, so this approach is recommended for a general quadratic programming code. Let us now define the steps of the procedure in detail.

The first stage is to express the objective function in the required form. We may refer to this as the "diagonalization" of the objective function. The best approach will depend on how the problem is specified, so we treat it as a preliminary operation, to be carried out in the matrix generator before entering the main mathematical programming routine. Many problems may start with only squared quadratic terms, so there will be nothing more to do at this stage. But obviously we should not rely on this.

One procedure is to use a standard subroutine to find the eigenvectors and eigenvalues of the matrix (c_{kl}). This is quite convenient with a moderate sized problem and a powerful computer; but it is theoretically over-elaborate, since it goes to some trouble to create an orthogonality between the λ_i that has no real relevance to the problem. If one is prepared to write a special routine for this part of the work, the following logic is recommended.

Consider the expression

$$C = \sum_{k=1}^{n-m} \sum_{l=1}^{n-m} c_{kl} x_k x_l$$

where $c_{kl} = c_{lk}$, and look for its largest coefficient in absolute value. If this is a diagonal element, say c_{pp}, and if it is positive, then define

$$\lambda_1 = \sqrt{\frac{2}{c_{pp}}} \sum_{k=1}^{n-m} c_{kp} x_k .$$

Then we see that

$$C = \tfrac{1}{2}\lambda_1^2 + \sum_{\substack{k=1 \\ k \neq p}}^{n-m} \sum_{\substack{l=1 \\ l \neq p}}^{n-m} c'_{kl} x_k x_l ,$$

where $c'_{kl} = c_{kl} - c_{kp} c_{pl} / c_{pp}$.

Similarly if c_{pp} is negative we write

$$\lambda_1 = \sqrt{\frac{-2}{c_{pp}}} \sum_{k=1}^{n-m} c_{kp} x_k$$

and $C = -\tfrac{1}{2}\lambda_1^2 + \sum c'_{kl} x_k x_l$ where c'_{kl} is defined as above.

So in this way we have removed one variable from the part of the expression for C that is not in the form of a sum or difference of squares. We can now repeat the procedure on the remainder.

It therefore only remains to define the procedure if the largest coefficient is not a diagonal one. This is something that cannot happen if C is convex, but it is important not to be bound to this condition. We might then be in trouble without some additional rule of procedure. For example all the diagonal elements might even vanish. So we adopt the following policy.

If the largest coefficient is c_{pq} with $p \neq q$ we make a preliminary change of variable as follows:

Write

$$y_p = x_p + x_q$$
$$y_q = x_p - x_q ,$$

then

$$C = \sum_{k=1}^{n-m} \sum_{l=1}^{n-m} c_{kl} x_k x_l = \sum_{k=1}^{n-m} \sum_{l=1}^{n-m} c'_{kl} y_k y_l ,$$

where $y_k = x_k$ for $k \neq p, q$

and

$$\begin{aligned} c'_{pp} &= c_{pp}+c_{qq}+2c_{pq} \\ c'_{pq} &= c_{pp}-c_{qq} \\ c'_{qq} &= c_{pp}+c_{qq}-2c_{pq} \\ c'_{kp} &= c_{kp}+c_{kq} \\ c'_{kq} &= c_{kp}-c_{kq} \\ c'_{kl} &= c_{kl}, \text{ where } k, l \neq p, q. \end{aligned}$$

And if c_{pq} is the numerically largest of the c_{kl}, then either c'_{pp} or c'_{qq} must be the numerically largest of the c'_{kl}. We are therefore back in the situation discussed earlier and can extract a new squared term in the y-variables. It is then a simple matter to substitute back the original x-variables in this expression.

Now suppose that the diagonalization is complete, and denote the expressions for the λ_i in terms of the nonbasic variables of the problem by

$$\lambda_i = d_{i0}+\sum_{k=1}^{n-m} d_{ik}z_k. \tag{3.2}$$

To carry out the logic of the algorithm in this form, one must be able to compute $\partial C/\partial z_p$. (It is easiest to work with this quantity rather than half of it.) Its value a_{0p} at the current trial solution, used to decide which nonbasic variable to increase, is given by

$$a_{0p} = d_{0p}+\sum_{i=1}^{r_1} d_{i0}d_{ip}-\sum_{i=r_1+1}^{r} d_{i0}d_{ip}, \tag{3.3}$$

from (3.1) and (3.2). Having chosen the variable z_p to be increased, we must be able to compute $\partial^2 C/\partial z_p^2$ in order to decide whether the new nonbasic variable should be a new free variable. This second derivative is obtainable as

$$\sum_{i=1}^{r_1} d_{ip}^2-\sum_{i=r_1+1}^{r} d_{ip}^2.$$

Finally, if a new free variable u_t is needed, it is defined by the equations

$$u_t = d_{0p}+\sum_{i=1}^{r_1} d_{ip}\lambda_i-\sum_{i=r_1+1}^{r} d_{ip}\lambda_i \tag{3.4}$$

$$= a_{0p}+\sum_{k=1}^{n-m} c_k z_k, \tag{3.5}$$

where

$$c_k = \sum_{i=1}^{r_1} d_{ik}d_{ip}-\sum_{i=r_1+1}^{r} d_{ik}d_{ip}. \tag{3.6}$$

Note that, to avoid difficulty with rounding-off errors, we do not use the derivatives to test whether a free variable should be removed from the set of nonbasic variables. We do this by theory, removing such variables if and only if they became nonbasic before the last restricted variable became nonbasic.

At the very end of the process, we shall need to know the value of the objective function. This is of course

$$d_{00}+\tfrac{1}{2}\sum_{i=1}^{r_1} d_{i0}^2-\tfrac{1}{2}\sum_{i=r_1+1}^{r} d_{i0}^2. \tag{3.7}$$

III.5. Numerical example

We now resolve the numerical example considered in the last chapter using the practical version of the algorithm. It must be admitted that for this type of problem, with a quadratic form of full rank, the practical version of the algorithm has no significant advantage over the original form. But nevertheless the problem can be solved quite easily this way.

We must first diagonalize the quadratic terms in the objective function. Following the steps outlined above for doing this we find that

$$\begin{aligned}
C &= 9-8x_1-6x_2-4x_3 \\
&\quad + 2x_1^2+2x_2^2+ x_3^2+2x_1x_2+2x_1x_3 \\
&= 9-8x_1-6x_2-4x_3 \\
&\quad + \tfrac{1}{2}(2x_1+ x_2+ x_3)^2 \\
&\quad + \tfrac{3}{2}x_2^2-x_2x_3+\tfrac{1}{2}x_3^2 \\
&= 9-8x_1-6x_2-4x_3 \\
&\quad + \tfrac{1}{2}(2x_1+ x_2+ x_3)^2 \\
&\quad + \tfrac{1}{2}(x_2\sqrt{3}-\tfrac{1}{3}x_3\sqrt{3})^2 \\
&\quad + \tfrac{1}{3}x_3^2.
\end{aligned}$$

So our initial tableau is as follows:

$$\begin{aligned}
x_4 &= 3-x_1 - x_2-2x_3 \\
\lambda_0 &= 9-8x_1-6x_2-4x_3 \\
\lambda_1 &= 2x_1+ x_2+ x_3 \\
\lambda_2 &= x_2\sqrt{3}-\tfrac{1}{3}x_3\sqrt{3} \\
\lambda_3 &= \tfrac{1}{3}x_3\sqrt{6}
\end{aligned}$$

and $r_1 = 3$, $r = 3$.

Applying the formula for derivatives, we find that $\partial C/\partial x_1 = -8$, $\partial C/\partial x_2 = -6$, $\partial C/\partial x_3 = -4$.

So we increase x_1. We see that $\partial^2 C/\partial x_1^2 = 4$. So, applying the usual ratio test, we see that we must introduce a free nonbasic variable. We write

$$u_1 = -8+4x_1+2x_2+2x_3.$$

So the next tableau becomes

$$\begin{aligned}
x_1 &= 2+\tfrac{1}{4}u_1-\tfrac{1}{2}x_2 \quad -\tfrac{1}{2}x_3\\
x_4 &= 1-\tfrac{1}{4}u_1-\tfrac{1}{2}x_2 \quad -\tfrac{3}{2}x_3\\
\lambda_0 &= -7-2u_1-2x_2\\
\lambda_1 &= 4+\tfrac{1}{2}u_1\\
\lambda_2 &= \qquad x_2\sqrt{3}-\tfrac{1}{3}x_3\sqrt{3}\\
\lambda_3 &= \qquad \tfrac{1}{3}x_3\sqrt{6}.
\end{aligned}$$

Applying the formula for the derivatives of C with respect to the nonbasic restricted variables, we find that

$$\partial C/\partial x_2 = -2, \quad \partial C/\partial x_3 = 0.$$

So we increase x_2. We see that $\partial^2 C/\partial x_2^2 = 3$. So, applying the ratio test, we see that we must again introduce a free nonbasic variable. We write

$$u_2 = -2+3x_2-x_3.$$

So the next tableau becomes

$$\begin{aligned}
x_2 &= \tfrac{2}{3} + \quad \tfrac{1}{3}u_2+ \quad \tfrac{1}{3}x_3\\
x_1 &= \tfrac{5}{3} + \quad \tfrac{1}{4}u_1+\tfrac{1}{6}u_2-\tfrac{2}{3}x_3\\
x_4 &= \tfrac{2}{3} - \quad \tfrac{1}{4}u_1-\tfrac{1}{6}u_2-\tfrac{5}{3}x_3\\
\lambda_0 &= -\tfrac{25}{3}-2u_1-\tfrac{2}{3}u_2-\tfrac{2}{3}x_3\\
\lambda_1 &= 4 \quad +\tfrac{1}{2}u_1\\
\lambda_2 &= \tfrac{2}{3}\sqrt{3} \qquad +\tfrac{1}{3}u_2\sqrt{3}\\
\lambda_3 &= \qquad\qquad \tfrac{1}{3}x_3\sqrt{6}.
\end{aligned}$$

Applying the formula for the derivative of C with respect to the remaining nonbasic restricted variable, we find that

$$\partial C/\partial x_3 = -\tfrac{2}{3}.$$

So we increase x_3. We see that $\partial^2 C/\partial x_3^2 = \tfrac{2}{3}$. So, applying the ratio test, we see that we must make x_4 nonbasic. So we write

$$
\begin{aligned}
x_3 &= \tfrac{2}{5} - \tfrac{3}{20}u_1 - \tfrac{1}{10}u_2 - \tfrac{3}{5}x_4 \\
x_2 &= \tfrac{4}{5} - \tfrac{1}{20}u_1 + \tfrac{3}{10}u_2 - \tfrac{1}{5}x_4 \\
x_1 &= \tfrac{7}{5} + \tfrac{7}{20}u_1 + \tfrac{1}{10}u_2 + \tfrac{2}{5}x_4 \\
\lambda_0 &= -\tfrac{43}{5} - \tfrac{19}{10}u_1 - \tfrac{3}{5}u_2 + \tfrac{2}{5}x_4 \\
\lambda_1 &= 4 + \tfrac{1}{2}u_1 \\
\lambda_2 &= \tfrac{2}{3}\sqrt{3} + \tfrac{1}{3}u_2\sqrt{3} \\
\lambda_3 &= \tfrac{2}{15}\sqrt{6} - \tfrac{1}{20}u_1\sqrt{6} - \tfrac{1}{30}u_2\sqrt{6} - \tfrac{1}{5}x_4\sqrt{6}.
\end{aligned}
$$

The variables u_1 and u_2 must now be eliminated.
Starting with u_1, we see that

$$\frac{\partial C}{\partial u_1} = \tfrac{3}{50}, \qquad \frac{\partial^2 C}{\partial u_1^2} = \tfrac{53}{200}.$$

So we must decrease u_1, and introduce a new nonbasic free variable. We write

$$u_3 = \tfrac{3}{50} + \tfrac{53}{100}u_1 + \tfrac{1}{100}u_2 + \tfrac{3}{50}x_4.$$

So the next tableau becomes

$$
\begin{aligned}
u_1 &= -\tfrac{12}{53} + \tfrac{200}{53}u_3 - \tfrac{2}{53}u_2 - \tfrac{12}{53}x_4 \\
x_3 &= \tfrac{23}{53} - \tfrac{30}{53}u_3 - \tfrac{5}{53}u_2 - \tfrac{30}{53}x_4 \\
x_2 &= \tfrac{43}{53} - \tfrac{10}{53}u_3 + \tfrac{16}{53}u_2 - \tfrac{10}{53}x_4 \\
x_1 &= \tfrac{70}{53} + \tfrac{70}{53}u_3 - \tfrac{6}{53}u_2 + \tfrac{17}{53}x_4 \\
\lambda_0 &= -\tfrac{433}{53} - \tfrac{380}{53}u_3 - \tfrac{28}{53}u_2 + \tfrac{44}{53}x_4 \\
\lambda_1 &= \tfrac{206}{53} + \tfrac{100}{53}u_3 - \tfrac{1}{53}u_2 - \tfrac{6}{53}x_4 \\
\lambda_2 &= \tfrac{2}{3}\sqrt{3} + \tfrac{1}{3}u_2\sqrt{3} \\
\lambda_3 &= \tfrac{23}{159}\sqrt{6} - \tfrac{10}{53}u_3\sqrt{6} - \tfrac{5}{159}u_2\sqrt{6} - \tfrac{10}{53}x_4\sqrt{6}.
\end{aligned}
$$

We must now eliminate u_2.
We see that

$$\frac{\partial C}{\partial u_2} = \tfrac{2}{53}, \qquad \frac{\partial^2 C}{\partial u^2} = \tfrac{18}{53}.$$

So we must decrease u_2, and introduce a new nonbasic free variable. We write

$$u_4 = -\tfrac{2}{53} + \tfrac{18}{53}u_2 + \tfrac{2}{53}x_4.$$

So the next tableau becomes

$$\begin{array}{llll}
u_2 = -\frac{1}{9} & & +\frac{53}{18}u_4 & -\frac{1}{9}x_4 \\
x_3 = \frac{4}{9} & -\frac{30}{53}u_3 & -\frac{5}{18}u_4 & -\frac{5}{9}x_4 \\
x_2 = \frac{7}{9} & -\frac{10}{53}u_3 & +\frac{8}{9}u_4 & -\frac{2}{9}x_4 \\
x_1 = \frac{4}{3} & +\frac{70}{53}u_3 & -\frac{1}{3}u_4 & +\frac{1}{3}x_4 \\
\lambda_0 = -\frac{73}{9} & -\frac{380}{53}u_3 & -\frac{14}{9}u_4 & +\frac{8}{9}x_4 \\
\lambda_1 = \frac{35}{9} & +\frac{100}{53}u_3 & -\frac{1}{18}u_4 & -\frac{1}{9}x_4 \\
\lambda_2 = \frac{17}{27}\sqrt{3} & & +\frac{53}{54}u_4\sqrt{3} & -\frac{1}{27}x_4\sqrt{3} \\
\lambda_3 = \frac{4}{27}\sqrt{6} & -\frac{10}{53}u_3\sqrt{6} & -\frac{5}{54}u_4\sqrt{6} & -\frac{5}{27}x_4\sqrt{6}.
\end{array}$$

The problem is now back in standard form. We must therefore again consider the derivative of C with respect to the (only) nonbasic restricted variable.

We see that $\partial C/\partial x_4 = \frac{2}{9}$, which is positive. So we have the final solution. The value of the objective functions is.

$$-\tfrac{73}{9}+\tfrac{1}{2}\{(\tfrac{35}{9})^2+(\tfrac{17}{27}\sqrt{3})^2-(\tfrac{4}{27}\sqrt{6})^2\} = \tfrac{1}{9}.$$

We have achieved the same result as before.

IV. THE INVERSE MATRIX METHOD FOR LINEAR AND QUADRATIC PROGRAMMING

IV.1. Introduction

It is widely known that all, or nearly all, production computer codes for solving large linear programming problems use the product form of the inverse matrix method. But it is rather less widely known what this form really involves. It therefore seems desirable to review the product form for linear programming before discussing its application to quadratic programming.

IV.2. Outline of the inverse matrix method

In the original, or straight, simplex method one works with the tableau of coefficients in the expressions for the basic variables as linear functions of the nonbasic variables. If there are m equations and n variables this means that one works with an array of $(m+1) \times (n-m+1)$ coefficients, assuming one objective function and one "right hand side", or column of constant terms. All these coefficients have to be up-dated from one iteration to the next, although only very few of them are actually used in any single iteration. Specifically one uses

(a) all the coefficients in the objective function, in order to select the new pivotal column (normally chosen as the one with the most negative reduced cost),

(b) the right hand side and the elements in the pivotal column, in order to select the new pivotal row,

(c) the other elements in the pivotal row, in order to update the expression for the objective function.

The remaining columns are up-dated simply because they may be needed as pivotal columns in a subsequent iteration. It is therefore natural to wonder whether, instead of carrying around all this information in case it is needed, one cannot represent the problem more compactly, calculating particular elements of the tableau only when required. It turns out that this

is possible. To explain this clearly it seems necessary to use matrix notation.

The original constraints of the problem can be expressed as equations by adding suitable slack variables in all inequality constraints. And these constraints can be written as the matrix equation

$$Ax = b,$$

where A is an $(m \times n)$ matrix, x an n-vector and b an m-vector. The complete problem is defined by the constraints and the objective function, and it is desirable that our matrix equation should include the expression for the objective function. We therefore define a new variable x_0 and a new equation

$$x_0 + \sum a_{0j} x_j = b_0,$$

where $\sum a_{0j} x_j - b_0$ represents the expression to be minimized. This minimization can obviously be achieved by maximizing x_0.

So we now think of A as an $(m+1) \times (n+1)$ dimensional matrix.

Next, consider the situation at any particular iteration during the solution of this linear programming problem by the simplex method. There will be a set of basic variables, and we can imagine that the variables are renumbered so that these are the variables $x_1, \cdots, x_m$. Then our matrix equation can be written in the form

$$(B \mid A_1)x = b,$$

where B is an $(m+1) \times (m+1)$ square matrix of coefficients of the variables $x_0, x_1, \cdots, x_m$, and A_1 denotes the remaining columns of A, i.e. the coefficients of the nonbasic variables. If we now premultiply our matrix equation by B^{-1}, we have it in the form

$$(I \mid B^{-1}A_1)x = \beta,$$

where

$$\beta = B^{-1}b.$$

And this means in effect that we have expressed the variables $x_0, x_1, \cdots, x_m$ as linear functions of the variables $x_{m+1}, \cdots, x_n$. So the coefficients of the matrix $B^{-1}A_1$ are in fact the coefficients in the current tableau, and the coefficients of the column vector β are the current right hand sides, or values of the objective function and the basic variables.

Furthermore, the coefficients of the matrix B^{-1} can be updated from one iteration to the next in just the same way as one updates a tableau in the straight simplex method. This important fact follows most easily from the fact that the columns of B^{-1} can be regarded as columns in the tableau

– they are the columns of coefficients of the original slack or artificial variables associated with each row of the matrix.

In the inverse matrix method, one therefore works with the original matrix A, the current right hand side β, and some expression for B^{-1}, the inverse of the current basis. For the time being, we may imagine this as an ordinary matrix. We then have the explicit inverse form of the simplex method. Later we shall consider the alternative, product, form.

IV.3. The steps of the inverse matrix method

Each simplex iteration can be subdivided into 5 steps when using the inverse matrix method, as follows:

Step 1: Form a pricing vector, i.e. a row vector π that can be multiplied into any column of the matrix A to determine the reduced cost for the corresponding variable.

The point here is that we wish to pick out the first row of the tableau, i.e. of the matrix $B^{-1}A$, which can be done by premultiplying by a row vector c whose first element is unity and whose remaining m elements are zero. We therefore form the vector product $\pi = cB^{-1}$, so that we can subsequently form the product πA to determine the row vector of reduced costs. If B^{-1} is stored explicitly, this operation simply involves picking out its top row.

Step 2: Price out the columns of the matrix A to find a variable to remove from the set of nonbasic variables. This is normally done by forming every element of the matrix πA and choosing the most negative, though alternative methods have been proposed, since this step involves most of the computation in the inverse matrix method.

Step 3: Form the updated column α of coefficients of the variable to be removed from the set of nonbasic variables, by premultiplying the appropriate column of A by B^{-1}.

Step 4: Perform a ratio test between the elements of α and the elements of β to determine the pivot, and the variable to be made nonbasic.

Step 5: Update the set of basic variables, the vector β and the inverse B^{-1}. The first of these operations is simply bookkeeping. The remainder is the same as the straight simplex method on a smaller matrix consisting of a pivotal column α (whose updated value is not needed), the columns of B^{-1}, and a right hand side β.

IV.4. The product form of inverse

Using the product form, the inverse of the basis is not recorded explicitly. Instead, it is represented as a product of elementary matrices. Each of these

represents the effect of a single pivotal operation. Such an operation, in which the pivot is in the p^{th} row, is to premultiply B^{-1} by a matrix that is a unit matrix except for the p^{th} column, which is computed from the updated pivotal column in the tableau. This pivotal column is computed in Step 4 of the inverse matrix method. If its components are denoted by α_i, then the p^{th} element of the p^{th} column of the elementary matrix is $1/\alpha_p$, and the i^{th} element, for $i \neq p$, is $-\alpha_i/\alpha_p$. These elementary matrices can be stored in the computer very compactly. One simply records the column number p, and the nonzero elements in it; the remaining, unit, columns being understood. In fact these unit columns are so taken for granted that the elementary matrices are often referred to as vectors, specifically "η-vectors".

The steps of the inverse matrix method can be carried out when B^{-1} is represented as a product of η-vectors. Step 1 is then called the "backward transformation", since the row vector c is postmultiplied by each η-vector successively, in the opposite order to that in which they were generated. It will be noted that each η-vector affects only the p^{th} element of the row vector being built up. Step 2 is carried out as with the explicit inverse. Step 3 is called the "forward transformation", since the pivotal column of A is premultiplied by each η-vector successively in the order in which they were generated. Step 4 is carried out as with the explicit inverse, as is Step 5, except that the process of updating B^{-1} is very simple – one just adds another η-vector to the end of the list.

After a while, the list of η-vectors becomes undesirably long, and to shorten it one goes through a process known as reinversion. It should perhaps be made clear that this reinversion does not mean recording some explicit inverse which is then simply updated by adding η-vectors. One always starts from a unit inverse representing an all slack-or-artificial basis, and adds η-vectors representing pivotal operations to replace unwanted slacks or artificials by genuine variables. Knowing which variables are to be introduced, and which slacks or artificials are to be removed, one can perform these pivotal operations without thinking at all about the signs of the values of the basic variables or of the reduced costs at intermediate stages of the inversion. In fact one has considerable freedom in the choice of pivotal columns and rows during reinversion. One must choose a column corresponding to a variable that is due to be in the basis but has not yet been used, and one must choose a row corresponding to a slack or artificial that is due to be removed from the basis but has not yet been used, and which will give a nonzero value to the pivot (α_p). But that is all. If all the elements of B were nonzero, then it would probably be best to choose pivotal columns

and rows to give the largest possible pivot in absolute value, so as to maximize numerical accuracy. But in practice matrices arising in large linear programming problems contain a high proportion of zero elements, and the number of nonzero elements in the set of η-vectors depends very significantly on the order in which the pivotal columns and rows are selected. An important computational aspect of linear programming is the choice of pivots during inversion, since this affects the speed of inversion itself and also the speed of subsequent backward and forward transformations.

If B were a lower triangular matrix, and one pivoted on the diagonal elements in order, starting with the first and finishing with the last, then the η-vectors would contain no more off-diagonal elements than the original matrix B. It is of course possible to invert the matrix using different pivots, or the same pivots in a different order: but this will in general produce more nonzero elements in the resulting η-vectors. In principle therefore what a modern inversion routine does is to premute the rows and columns of B so that they are as nearly in lower triangular form as possible, and then pick the pivots in the original matrix B corresponding to pivoting on the diagonal elements of the permuted matrix in order.

In practice one often starts a linear programming calculation by inverting to a prescribed initial basis. This initial inversion is performed in exactly the same way as a re-inversion.

IV.5. The advantages of the product form

The advantages of the explicit inverse method over the straight simplex method, and of the product form over the explicit inverse, are often summarized as follows:

In the straight simplex method one has to update the complete tableau, involving $(m+1) \times (n-m+1)$ numbers, at each iteration. In the explicit inverse method one only has to update the inverse, which involves $(m+1) \times (m+1)$ numbers. This is a considerable saving if, as is usual, n is much larger than m. The advantage of the product form comes from the fact that even this updating is avoided.

From a practical point of view this explanation describes the situation reasonably accurately. But from a theoretical point of view it is so unsatisfactory as to cause a number of workers in the field to think that the inverse matrix method is an elaborate hoax and that the straight simplex method is really the best.

The theoretical weakness of the argument lies in the fact that if the matrix A were full (i.e. contained no zero coefficients) then the pricing operation,

Step 2 of the inverse matrix method, would already involve as much arithmetic as updating the tableau, and the perhaps small amount of additional work in Step 1 and 5 would simply make the inverse matrix method even less competitive.

Obviously, in order to get to the heart of the matter we must take note of the fact that a very large proportion of the elements of the matrix A vanish. But we must still be careful. If a proportion ρ of the elements of A are non-zero, then pricing in the inverse matrix method will involve about $\rho \times (m+1) \times (n-m+1)$ multiplications, since the π-vector will usually be full. (And this assumes that one does not price the basic vectors.) On the other hand if a proportion ρ of the elements of the tableau are non-zero, then the updating involves only $\rho^2 \times (m+1) \times (n-m+1)$ multiplications. So the argument of sparseness can be used to provide further support for the straight simplex method.

So what is the true position? Part of the real advantage of the inverse matrix method, and in particular the product form, lies in its greater flexibility. One does not have to complete the pricing operation for every iteration. For example one can use multiple-column-selection techniques in which one uses the pricing operation to select a number of the most promising columns, updates them all in Step 3, and then does a few steps of the straight simplex method on this very small subproblem.

But even without this flexibility the inverse matrix method would still be advantageous on typical large problems because the two ρ's in the above formulae are not the same. The original matrix A will be much more sparse than typical tableaux during the calculation. On some computers it is worthwhile taking further advantage of the special structure of the matrix A by noting that many of its elements are ± 1. These unit elements can be stored more compactly than general elements. And when working with such unit elements during pricing one needs only add or subtract rather than multiply.

Again, the advantage of the product form over the explicit inverse lies not so much in the fact that it simplifies the task of updating the inverse as in the fact that it generally provides a more compact expression for the inverse if one has a fast and efficient inversion routine on the lines described at the end of the last subsection.

There is another practical advantage of the inverse matrix method, which applies even more strongly in the product form. Having updated a tableau, or even an explicit inverse, one generally has to write it out on to a magnetic tape or some other backing store if one is solving a large problem. It is true that most computers can transfer information in this way at the same

time as getting on with other calculations, but these transfers are apt to impede the process of reading further information into the working store for more processing. This bottleneck may be a passing phase in computer technology, but if so it is taking a long time to pass. Vast improvements have been made in the past 10 years in methods of moving information from one part of the machine to another, but these are having a hard time keeping up with the vast improvements in arithmetic speed.

Published references to these problems of computational efficiency are somewhat meagre. Some careful theoretical analyses are given in chapters 4 and 5 of Zoutendijk [14], numerical results are reported by Wolfe and Cutler [15] and by Smith and Orchard-Hays [16].

IV.6. The inverse matrix method for quadratic programming

Let us now return to the subject of quadratic programming. It has been said that one disadvantage of Beale's method is that it does not lend itself to the inverse matrix method. In fact the opposite is more nearly true – only with Beale's method is the matrix an appropriate shape (with many more columns than rows) for the inverse matrix method to be attractive.

The application of the inverse matrix method to Beale's quadratic programming has some interesting features, which we now explore.

The program should store the information in the usual way, by columns. It is apparently awkward that one has to add rows, and delete rows, throughout the calculation. But in fact this causes no great difficulty. One will start with $m+r+1$ rows, representing m constraints, 1 linear part of the objective function, and r rows representing the λ_i. In addition one needs some spare rows, or "u-rows" in which to record the equations for the free variables. Theoretically $r+2$ u-rows will be enough, but in practice it is desirable to allow rather more.

The free variables are numbered serially from 1 upwards. If there are U u-rows, then free variable i will be defined in u-row j, where j is the remainder when i is divided by U. Two "markers" I and J are required, such that all free variables with a serial number less than or equal to I have been removed from the nonbasic set, and all free variables with a serial number less than or equal to J must be removed from the nonbasic set. Initially $I = J = 0$, but J is increased to the number of the latest free variable whenever a restricted variable is made nonbasic. If $I < J$, then free variable $I+1$ is made basic at the next iteration and I is increased by one. Once a free variable has been made basic it may be removed from the problem if one is

using the explicit form of inverse. But in the product form it must be retained until the next reinversion.

As we have seen, an iteration in the inverse matrix method consists of 5 steps, but in quadratic programming the first 2 are omitted if $I < J$.

1. Form a pricing vector, i.e. a row-vector that can be multiplied into any column of the matrix to determine the reduced cost, i.e. the value of $\partial C/\partial z_k$, for that variable. In quadratic programming this operation is unaffected, except that (once the problem has become feasible) the original row vector c has elements 1, $d_{10}, \cdots d_{r_1,0}, -d_{r_1+1,0}, \cdots, -d_{r0}$ in the columns corresponding to rows $\lambda_0, \lambda_1, \cdots, \lambda_r$ respectively.

2. Price out the columns of the matrix to find a variable z_p to remove from the set of nonbasic variables. In quadratic programming this operation is applied in the usual way to the restricted variables if $I = J$. Otherwise one just picks the free variable $I+1$ and increases I by one.

3. Update the column α of coefficients of z_p in the tableau – by premultiplying the original coefficients of z_p by the inverse of the current basis. This operation is the same in quadratic and linear programming.

4. Perform a ratio test to choose the next variable to become nonbasic. This works rather differently in quadratic programming, but the changes are fairly obvious.

(a) If a free variable is being made basic, and has a positive reduced cost a_{0p}, then the signs of all the elements of α must be reversed.

(b) The rows associated with the quantities λ_i, and any basic free variable, must be omitted from the test.

(c) Having found θ, the amount by which z_p can be increased, compute

$$\Delta = a_{0p} + \theta\Big(\sum_{i=1}^{r_1} d_{pi}^2 - \sum_{i=r_1+1}^{r} d_{ip}^2\Big),$$

where a_{0p} is defined by (3.3). If Δ is negative, remove the indicated restricted variable from the basis in the usual way (and increase J to the number of the latest free variable). But if Δ is nonnegative a new free variable must be defined and made nonbasic. This can be done most compactly using eq. (3.4). On the other hand one must use eq. (3.5) and (3.6) if one is not prepared to invert immediately. But it is not necessary to choose between these approaches – we can use both eq. (3.4) and (3.5). The coefficients for the

expression for u_t in terms of both the λ_i and the z_k can be found using a pricing vector, formed in the usual way except that the row vector to be post-multiplied by the inverse of the basis has elements $d_{1p}, \cdots, d_{r_1 p}$, $-d_{r_1+1,p}, \cdots, -d_{rp}$ in the columns corresponding to the rows defining $\lambda_1, \cdots, \lambda_r$ and zeroes elsewhere. Both sets of coefficients can be entered on the A matrix. Until the next re-inversion the coefficients of the λ_1 can be ignored since the λ_i remain basic throughout. Immediately before re-inversion the coefficients of the z_k can be removed from the A matrix.

5. Update the set of basic variables, the vector of constant terms, and the inverse. This operation is the same in quadratic and linear programming. The necessary changes in the markers I and J have already been described.

V. SEPARABLE PROGRAMMING

V.1. Introduction

My remaining chapters are concerned with methods of solving mathematical programming problems that are linear programming problems except for a relatively small number of nonlinear constraints. The question whether the objective function is allowed to be nonlinear is not important, because, as Wolfe first pointed out many years ago, one can always make the objective function linear by introducing another nonlinear constraint. For if C is a nonlinear function to be minimized, then one redefines the problem so that one has to minimize z, where z is a new variable satisfying the constraint

$$C - z \leqq 0.$$

This point is made, for example, in Wolfe [12]. I confess that when I first heard this I thought it was a mathematical observation of no practical importance. But now I think that it is of some significance, since it emphasizes the fact that satisfactory methods of dealing with nonlinear constraints are all one really needs to solve nonlinear programming problems.

As Wolfe points out elsewhere in this volume, powerful special methods have been developed for maximizing general nonlinear objective functions of variables subject to linear constraints. Nevertheless I do not believe that it is worthwhile to develop efficient computer codes for solving such problems on a production basis. It is important to keep one's armoury of production programs from growing unnecessarily. Each such program needs maintenance effort in much the same way as physical equipment. This involves having people who know how to operate the program. But they must also know enough about its structure to fix bugs that may appear when one tries to use it in new circumstances, and to alter it to meet special requirements or to take advantage of new techniques (in either hardware of software) that may have become available. Such maintenance effort is always in short supply, and it should be concentrated as much as possible without

unduly restricting the class of problems one can solve efficiently. And nonlinear programming problems with linear constraints can be solved reasonably efficiently using codes for more general problems.

A useful method that can in principle be applied to all nonlinear constraints, and can in practice be applied to a wide variety of real problems, has been devised by Miller. It is called separable programming. It is still not as well known as it deserves to be, primarily because it was not formally published until 1963, see Miller [17], and perhaps partly because it is so simple that it may appear trivial to the theorist.

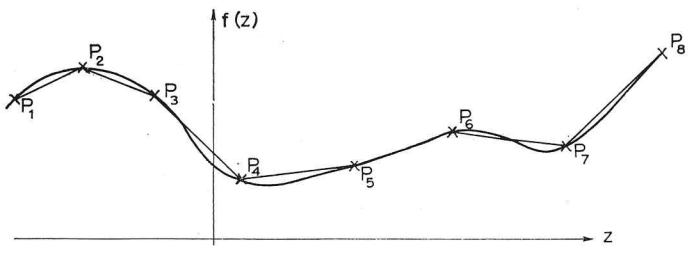
This chapter describes the theory of the method, and discusses some practical points concerned with its application, with particular reference to the handling of product terms in constraints. The material for this chapter is largely taken from Miller's paper, and from a paper by Beale, Coen and Flowerdew [18]. Some minor technical points, and also parametric programming and interpolation procedures, will be deferred until the next chapter.

V.2. The theory of separable programming

Miller's technique is known as separable programming because it assumes that all the nonlinear constraints in the problem can be separated out into sums and differences of nonlinear functions of single arguments. At first this assumption seems to severely restrict the usefulness of the method. But we shall see later that this is not really so.

As Miller points out, the technique is related to earlier work. Charnes and Lemke [19] first pointed out that convex separable nonlinearities in the objective function can be handled by the simplex method. Dantzig [20] reviews methods for minimizing separable convex objective functions. Miller's special contribution was to extend this approach to nonconvex problems. After this it was a simple matter to apply the method to nonlinear constraints as well as to nonlinear objective functions.

Suppose that we have some variable z, and that we want to deal with some function $f(z)$. Suppose that the graph of $f(z)$ looks something like this:



We now replace this function by a piecewise linear approximation based on a finite number of points on the graph. In the diagram I have taken 8 points $P_1, \cdots, P_8$.

Now let the coordinates of these 8 points be (a_i, b_i) and introduce 8 new nonnegative variables $\lambda_1, \cdots, \lambda_8$ and the equations

$$\lambda_1 + \cdots + \lambda_8 = 1 \tag{5.1}$$

$$a_1\lambda_1 + \cdots + a_8\lambda_8 = z \tag{5.2}$$

$$b_1\lambda_1 + \cdots + b_8\lambda_8 = f(z). \tag{5.2}$$

We call these new variables a single group of "special variables", for reasons that will soon be clear. The eqs. (5.1) and (5.2) are known respectively as the convexity and reference rows for this group of special variables.

Note that the quantities z and $f(z)$ are not necessarily nonnegative. This causes no inconvenience, since we will not normally want to introduce them explicitly in the mathematical programming model in any case.

Let us now consider some typical solutions of eqs. (5.1), (5.2) and (5.3).

If we put $\lambda_1 = 1$, $\lambda_2 = \cdots = \lambda_8 = 0$, then $z = a_1$ and $f(z) = b_1$, and we have the point P_1.

If we put $\lambda_1 = \frac{1}{2}$, $\lambda_2 = \frac{1}{2}$, $\lambda_3 = \cdots = 0$, then $z = \frac{1}{2}(a_1 + a_2)$ and $f(z) = \frac{1}{2}(b_1 + b_2)$, and we have a point half way between P_1 and P_2.

More generally, if we allow any 2 neighbouring special variables to take nonzero values, keeping the other special variables of the group equal to zero, then we will map out the piecewise linear approximation to $f(z)$ that we have agreed to use. On the other hand if we put say $\lambda_1 = \frac{1}{2}$ and $\lambda_3 = \frac{1}{2}$, $\lambda_2 = \lambda_4 = \lambda_5 = \cdots = 0$, we have a point midway between P_1 and P_3 which is not a valid one.

Now in some problems we know beforehand, perhaps from convexity considerations, that such inadmissible combinations of special variables cannot occur in an optimal solution even if we take no special steps to exclude them. In these circumstances we do not need to use separable programming.

Separable programming is a method of reaching a local optimum, which may possibly not be a global optimum, solution to a non-convex problem by taking special steps to exclude inadmissible combinations of special variables. Miller [17] gives an example of a phenomenon that he calls "special degeneracy", which can cause the procedure to terminate in a virtual local optimum as defined earlier. But he indicates that this hazard is not a serious one in practice.

The required special steps are very easy if we are using the simplex method. All we have to do is to restrict the set of variable to be considered as candidates for entering the basis. If two special variables of a group are in the current basis, then we do not consider allowing a third. And if one is already in the basis we consider only its neighbours.

The proof that one must reach at least virtual local optimum is straightforward. Obviously the algorithm must terminate, since it is a version of the simplex method for linear programming with the possibility of earlier termination. And when it does terminate we know that we have a true optimum to a linear programming problem obtained from our separable problem by suppressing all special variables other than those in the basis, and their neighbours when they are the only representatives of their group in the basis. This is a local optimum if all the basic special variables are at positive levels: if any are at zero levels it is still a virtual local optimum by the definition of permitted perturbations in Chapter III.

The implementation of these steps is itself easy in a mathematical programming code in which the variables have names. For example the C-E-I-R code LP/90/94 allows variable names to be any 6 characters, provided that the first is either blank or 1, 2, 3, 4, 5, 6, 7, 8 or 9. If the program is in the separable mode, then all variables with a 9 as their first character are treated as special ones. The next 3 characters define the group of special variables, and the last 2 are decimal digits defining the sequence of the special variables.

Incidentally, as Miller [17] points out, one can very easily deal with several different nonlinear functions of the same variable in the same problem. One simply writes down one equation of the type (5.3) for each function. And in practice one does not usually have to include these equations explicitly in the model, since the left hand side can be substituted for the right wherever it occurs.

Separable programming has also been implemented in codes making special provisions for bounded variables. One can then apply separable programming without introducing constraints of the type (5.1) by introducing bounded special variables representing the different increments in the independent variable z. A special variable is then allowed to increase above its lower bound (of zero) only if the previous variable of the group is at its upper bound. And it is allowed to decrease below its upper bound only if the following variable of the group is at its lower bound.

And that is all there is to the theory of separable programming. A number of technical points have to be considered when one comes to apply it.

We discuss some of these in connexion with an important class of applications, namely to product terms.

V.3. Product terms

A number of mathematical programming problems are linear except for the presence of a few product terms.

One may have a price of some commodity that is a linear function of other variables of the problem. The amount spent on this commodity is then the product of the price times the amount bought.

In oil production problems, the productivity of a well may be an approximately linear function of variables relating to production from this reservoir in previous years. The production available in the current year is then given by the product of the well productivity times the number of wells drilled.

The problem considered by Beale, Coen and Flowerdew [18] is actually concerned with iron-making, but it is in principle a rather general one.

Raw material is being fed into a number of production units. Various raw materials are available at varying costs, and they all have different specifications concerning their chemical compositions, etc. This sort of situation often leads to a standard application of linear programming to determine the cheapest combination of raw materials to meet certain specifications on the overall properties of the material supplies to the production units.

But nonlinearities arise if some raw material can be fed into a preprocessing unit, the output of which is then fed into more than one main production unit. In the iron-making application this preprocessor is a sinter plant. If the preprocessor only has to feed one type of main unit, or if it can be operated in different ways to feed the different types of main unit, then linear methods may still be applicable. But if the preprocessor has to be run in a fixed way to feed several types of main unit, then the problem is apt to be nonlinear.

A good way to handle such problems is often to define variables representing the proportion of the output from the preprocessor that is fed to particular main units. The amount of some chemical element in the preprocessed material supplied to a main unit is then the product of this new variable times the amount of this element in the output from the preprocessor. This amount is itself linearly related to the inputs to the preprocessor.

So it is important to be able to handle product terms. The expression $u_1 u_2$ is not a nonlinear function of a single variable, so it might appear that

it was not amenable to separable programming. If that were true, then separable programming would be of very limited value, but fortunately it is not true.

We note that

$$u_1 u_2 = \tfrac{1}{4}(u_1+u_2)^2 - \tfrac{1}{4}(u_1-u_2)^2, \tag{5.4}$$

so we can always express a product of 2 linear variables as a difference between 2 nonlinear functions of linear variables. This is a special case of the fact (exploited in Chapter III) that any quadratic function can be represented as a sum or difference of squares. Any such function is therefore amenable to separable programming. And we could then handle the product of such a quadratic function with another variable in the same way; so in theory there is no limit to the class of functions that can be represented in this way. In practice of course a very involved representation would be cumbersome computationally.

So we can deal with a simple product term by introducing 2 groups of special variables. This involves 4 extra equations – 2 convexity rows to represent the conditions that the sum of the special variables in each group must equal 1, and 2 reference rows to represent the value of the arguments of the nonlinear functions in terms of the special variables. These correspond to eqs. (5.1) and (5.2), in general with some linear combination of the basic variables of the problem substituted for z in (5.2). In practice we will generally not write down an equation corresponding to (5.3) explicitly, since we can substitute the left hand side of this equation for $f(z)$ wherever it occurs.

I am grateful to Eli Hellerman for pointing out that repeated use of (5.4) is not the only way to handle more complicated product terms. One may have to consider an expression of the form

$$x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n},$$

where the x_i are essentially positive and have a lower bound that is strictly greater than zero. It will then generally be more economical to write this expression as $\exp w$, where

$$w = a_1 \ln x_1 + a_2 \ln x_2 + \cdots + a_n \ln x_n.$$

This treatment involves $n+1$ groups of special variables and $2(n+1)$ extra equations – one of which defines the special variables representing $\exp w$ in terms of sums of special variables from the other groups. So this logarithmic treatment is inferior to the use of (5.4) for simple product

terms, but it will generally be best for more complicated products when the variables concerned are essentially positive. And it may be advantageous for simple product terms if one has several such terms involving the same set of variables.

V.4. Defining the ranges of the variables

The next point to notice is that separable programming only deals with nonlinear functions over definite ranges of their arguments. It is clear from eq. (5.2) that z cannot be less than a_1 or greater than a_8. Occasionally the fact that one automatically fixes lower and upper bounds on the independent variable defining a nonlinear function is a useful bonus to the formulator of the problem. More often it is an extra chore to have to fix realistic bounds for these variables, or else run one of 3 risks:

(a) to have an unnecessarily inaccurate approximation (due to using a widely spaced grid), or

(b) to have an unnecessarily large problem (due to having more points than are necessary to define the nonlinear functions involved), or

(c) to finish up with a solution in which some independent variable for a nonlinear function is up against one of its limits, indicating that a better solution might possibly be available beyond this limit. (Of course if the solution falls within all the chosen limits, then it would necessarily remain a solution if the limits were relaxed, even if one did not know a priori that these limits were justified.)

In the problems I have encountered so far, there has been no particular difficulty in fixing appropriate limits. In this connexion it is often helpful to work with the *proportion* of some input or output that is composed or used in a certain way. This proportion must obviously lie between 0 and 1. An alternative formulation might be possible in terms of the *ratio* of the amounts used in 2 different ways; but such a ratio might vary between 0 and ∞.

Another illustration of the convenience of proportions is that if one has a variable z defined by

$$z = \sum_{i=1}^{n} c_i x_i ,$$

where the x_i are proportions, and are therefore nonnegative and sum to 1, then z must lie between the smallest and the largest of the c_i.

When dealing with product terms, it seems best to define the quantities u_1 and u_2 to which one applies the identity (5.4) so that each covers the range $0 \leqq u_1, u_2 \leqq 1$. If one has a product term $v_1 v_2$ with arbitrary, but specified, ranges for v_1 and v_2, one can always write

$$v_1 = a_1 + b_1 u_1$$
$$v_2 = a_2 + b_2 u_2,$$

where $0 \leqq u_1 \leqq 1$, and $0 \leqq u_2 \leqq 1$. Then

$$v_1 v_2 = a_1 a_2 + a_1 b_2 u_2 + a_2 b_1 u_1 + b_1 b_2 u_1 u_2,$$

i.e.

$$v_1 v_2 = -a_1 a_2 + a_2 v_1 + a_1 v_2 + f_1(z_1) + f_2(z_2),$$

where

$$f_1(z_1) = b_1 b_2 z_1^2$$
$$f_2(z_2) = -b_1 b_2 z_1^2$$
$$z_1 = 0.5(-a_1 b_1^{-1} - a_2 b_2^{-1} + b_1^{-1} v_1 + b_2^{-1} v_2)$$
$$z_2 = 0.5(-a_1 b_1^{-1} + a_2 b_2^{-1} + b_1^{-1} v_1 - b_2^{-1} v_2),$$

so that $0 \leqq z_1 \leqq 1$, $-\frac{1}{2} \leqq z_2 \leqq \frac{1}{2}$.

V.5. Defining the grid

Just one further decision has to be taken: how fine a grid of points do we need to represent our nonlinear function?

It is a straightforward matter to show that if one interpolates linearly for the function cz^2 between $z = a$ and $z = a+h$, then the maximum error occurs when $z = a + \frac{1}{2}h$ and equals $\frac{1}{4}ch^2$. Since this quantity is independent of a, it implies that one should use an equal spacing for the independent variable when fitting a quadratic function, though of course there is no obligation to use equal spacing in general. A convenient rule of thumb to remember that if one is approximating the function z^2 between 0 and 1, then the maximum error is 0.01 (or 1% of the maximum value of z^2) if one uses 6 points at a spacing of 0.2 units. If one uses 11 points, at a spacing of 0.1 units, then the error is 0.0025, or $\frac{1}{4}$% of the maximum value of z^2. When a product of 2 variables lying between 0 and 1 is represented as the difference between 2 squares, then the maximum error is again 1% of the maximum value of the product if one uses 6 equally variables in each group – or $\frac{1}{4}$% if one uses 11 variables.

The question whether 6 points is enough is hard to answer definitely. I suspect that it usually is. An argument for going to 11 points is that,

even though the error in the 6 point approximation is small, the error in the approximated value of the product oscillates rather rapidly as one moves about the feasible region. And there is some danger that this oscillating error, if appreciable, could cause the optimizing process to stop at a spurious local optimum. But a counter-argument to this is that separable programming problems can take an appreciable time on the computer, even if they are inherently small problems, if one has to make large changes in one or more nonlinear variables represented by a fine grid. This point is discussed in the next chapter, in connexion with a procedure for automatically interpolating in the grid where this seems called for. The availability of such an interpolation procedure obviously has a considerable effect on the choice of initial spacing for the grid.

The maximum error in a single quadratic function can be halved by allowing grid points off the true curve being approximated. And similar results can obviously be obtained with other functions. But this does not reduce the oscillations in the error, and it does not help at all with product terms. Furthermore it prevents the use of the interpolation procedure mentioned above.

V.6. The relationship between separable programming and mixed integer programming

Although not strictly relevant to this course, it is of interest to note the connexion between separable programming, which is essentially a method for finding local optima, and the mixed integer programming formulation of the problem of finding a global optimum to the same problem. The use of mixed integer programming for this purpose has been discussed by Markowitz and Manne [21] and by Dantzig [22]. There are a number of similar ways of formulating the problem. Perhaps the simplest is one given by Dantzig [22]. The special rules defining which variables are allowed to be basic are replaced by the following constraints:

$$\begin{aligned}
\lambda_1 &\leqq \delta_1 \\
\lambda_2 &\leqq \delta_1 + \delta_2 \\
&\cdots\cdots\cdots\cdots\cdots\cdots \\
\lambda_{n-1} &\leqq \delta_{n-2} + \delta_{n-1} \\
\lambda_n &\leqq \qquad\quad \delta_{n-1}
\end{aligned}$$

where the δ_i have to be integers and

$$\sum \delta_i = 1.$$

VI. PARAMETRIC SEPARABLE PROGRAMMING AND INTERPOLATION PROCEDURES

VI.1. Introduction

This chapter starts with a discussion of 2 technical points concerning ordinary separable programming. A discussion of parametric separable programming follows. The chapter ends with a discussion of interpolation schemes in separable programming.

VI.2. Two possible traps

There are 2 possible traps concerning the use of separable programming when applied to a sophisticated linear programming code. They can both be avoided, but it is of some interest to see how they arise.

The first concerns the possibility of switching special variables. If for example λ_2 and λ_3 are in the current basis, then one might wish to introduce λ_4 and remove λ_2. This would be all right in theory, but it is not normally allowed in practice because one cannot be sure that λ_2 will be removed from the basis if λ_4 is selected as the new basic variable. So the program has to go through an intermediate solution in which λ_3 is the only variable of the group in the basis. This should not cause any fundamental difficulty. Unless there is a local optimum solution to the problem between the points P_2 and P_3, one will eventually reach the solution between P_3 and P_4 if this is better.

But a difficulty does arise if the problem is still infeasible and one has special variables in the basis at negative levels. One may have an infeasible solution on the line P_2P_3 produced which could be corrected by moving onto the line P_3P_4, or possibly P_4P_5, but which cannot be reached via the point P_3 since the sum of the infeasibilities is greater at P_3 than at the current solution. This difficulty would not arise with an elementary code that always used artificial variables rather than genuine variables at negative levels. But there are advantages in the use of these negative variables, notably in the fact that one may be able to remove several such infeasibilities in a single

iteration, whereas one can never remove more than one artificial variable in a single iteration.

Miller [17] suggests that one should overcome this difficulty by ignoring negative special variables when deciding which special variables are eligible to enter the basis. This procedure will generally work, though it is not completely sound unless one insists on pivoting a negative special variable out of the basis rather than let it become positive.

An alternative approach is as follows: if one has found an apparent local minimum of the sum of the infeasibilities with some special variable λ_n negative, then make a special step, introducing λ_p into the basis in place of λ_n, where $p = n+2$ if λ_{n+1} is in the current basis, and $p = n-2$ otherwise (i.e. if λ_{n-1} is in the current basis).

In the absence of either of these devices, one should avoid negative special variables in the initial basis (though small negative values hardly ever matter). This can be ensured by putting not more than one special variable of a group in the initial basis.

The other possible trap concerns another feature, introduced by D. M. Smith and widely used in modern linear programming codes, namely double (or multiple) column selection, mentioned in Chapter IV. This is a procedure for speeding up linear programming calculations by finding the best two (or more) candidate variables to enter the basis, updating both vectors, choosing the better of the two and then introducing the other if it still prices out profitably. The natural way to code separable programming is to test the eligibility of the variables when they become candidates, and not to take any special precautions to verify that the two candidates chosen are eligible to enter the basis together. I think that this natural way is the correct way, since any such special precautions would be a waste of valuable time in nearly all circumstances. There are really only two possible dangers in this respect. One is that if no special variables of a group are in the current basis one might select any two variables of the group as candidates. In particular one might select the two extreme ones at opposite ends of the range. This would be quite bad, and should be avoided by putting one special variable of each group into the initial basis. It is really no hardship at all to have to do this. It can always usefully replace the artificial variable associated with the convexity row for the group.

The other danger is that both neighbours of some basic special variable may enter the basis. This means that we were previously at a local pessimum solution to the problem. One will then almost certainly wish to move further off in one direction or the other, and the special variables will sort themselves

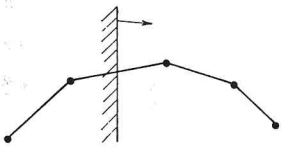
out. One can argue that there may even be some virtue in starting to move both ways at once, on the grounds that it makes it easier to recover from a false step when the rest of the problem is better sorted out.

The conclusion from all this is that, unless special precautions have been taken in the computer code, one should normally start with one and only one special variable from each group in the initial basis instead of the artificial on the convexity row.

VI.3. Parametric separable programming

One can do parametric linear programming, either on the right hand side or on the objective funtion, when in the separable mode. And there may be a special incentive for parametric programming on the right hand side; since if one changes to a new right hand side and attempts to optimize starting from the optimum basis for an old right hand side, then one may have difficulties of the kind just discussed with negative special variables.

The special points to note are as follows: In any parametric separable programming, when a special variable leaves the basis one must always check whether it would be profitable to introduce the other neighbour of the special variable in the same group that remains in the basis. This is a special variable that was not previously eligible to enter the basis. It will nearly always be unprofitable to introduce it. But when it is profitable it indicates that the family of local optima which one has been tracing out by parametric programming has come to an end, and one must do some regular optimizing steps without further changes in the parameter to find a new optimum. The significance of this event can be understood with the aid of a simple example, illustrated in the following diagram:



Suppose one is trying to find the smallest value of some nonlinear function $f(z)$ for $z \geqq a$, where a is a parameter. Suppose that one starts with $a = 0$, and suppose that $\mathrm{d}f/\mathrm{d}z > 0$ for $a = 0$. Then one has a local optimum with $z = a$. As a increases, the solution will remain $z = a$ until one reaches a point where $\mathrm{d}f/\mathrm{d}z < 0$. One will then no longer have a local optimum with $z = a$ and will increase z above its minimum value until a new local optimum is found.

A consequence of this is that one can explore for alternative local optima of a separable programming problem by "doing a Duke of York". One does not actually march 10000 men up to the top of the hill and down again. But one changes a right hand side (or an objective function) parametrically from one (vector) value to another and back again. One is by no means certain of getting back to the original starting solution. One might reach a better solution or a worse one.

One additional rule has to be considered for parametric programming on the right hand side. Since this is essentially a version of the dual simplex method, one chooses the variable to be removed from the basis first. If this is a special variable there is then no need to treat it as in the current basis when considering which special variables are eligible to enter the basis. And we have in fact come across problems in which it was essential not to do this (and to pivot between special variables differing by 2 in their indices).

VI.4. Interpolation

Miller [17] points out that separable programming problems can run quite slowly – sometimes taking as much as 4 times as long as one would expect for a normal linear problem of the same size. This slowness results primarily from the need to creep along the nonlinear functions rather than going straight to a more appropriate point on the graph. Miller also suggests that this slowness can be overcome by using some grid refinement scheme, starting from a coarse grid and refining it in the neighbourhood of the solution.

A grid refinement scheme is not very attractive in practice if it involves going back to the original problem formulation. This would mean removing all the mathematical programming results and the program itself from the computer, going back to the matrix generator to generate the additional points, and then restarting mathematical programming again. Even if the mathematical programming routine and the matrix generator were run on the same monitor, the communication problem would be quite significant. Also some tricky programming would be required, either in the matrix generator scheme, or in applying the procedure to any particular application, or both.

Fortunately this is not necessary. A purely numerical interpolation scheme can be incorporated in the mathematical programming routine without the necessity to refer back to any external information. Mrs. P. Griffiths of C-E-I-R Ltd. has recently coded a Lagrangian interpolation scheme that interpolates between points P_i and P_{i+1} by fitting a cubic to the points

P_{i-1}, P_i, P_{i+1} and P_{i+2} – or by fitting a quadratic to 3 of these points if one of the end points does not exist. This procedure is completely accurate for the quadratic functions that are often encountered in separable programming applications, and can be made accurate enough for all practical purposes for many other problems by a suitable choice of initial grid.

A somewhat more precise description of the procedure may be of interest. We hope that the best way of operating it will emerge with practical experience. Initially we intend to use it in a semi-manual way as follows.

At any time we can call for the agendum INTERPOLATE, which has a numerical parameter n associated with it. The program then looks at all the row names, and if it finds any whose first character is 9 and where last two characters are ** it assumes that this is a "reference row" for a group of special variables indexed by the 2nd, 3rd and 4th characters of the row name. This is assumed to correspond to the equation of the type (5.2), for this group of variables.

If there are no such variables in the current basis, then the program does nothing about this group.

If there is just one such variable, say the i^{th}, then the program interpolates n equally spaced new variables between the $(i-1)^{\text{th}}$ and the i^{th} variable of the group, and a further n new variables between the i^{th} and the $(i+1)^{\text{th}}$.

If there are 2 adjacent special variables of the group in the current basis, the program interpolates n equally spaced new variables between them.

If for any reason there are nonadjacent special variables in the basis, or special variables at negative levels, then the program does no interpolation in this group. Instead it uses the current values of these special variables, and their entries in the reference row, to select 2 suitable adjacent variables of the group to be in the basis; any others will be replaced by artificial variables.

This last facility simplifies the operation of a technique advocated by N. Williams that is valuable on some problems: first solve the problem using ordinary linear programming, and then enter the separable mode to clear up any illegitimate combinations of special variables.

When the interpolations have been carried out, the special variables are renumbered so that the previous basis represents the best estimate of the corresponding basis of the interpolated problem. The program then automatically reinverts the new problem to this basis and re-optimizes.

The entries in the new vectors in the reference row for the group concerned are equally spaced between those in the vectors between which one is inter-

polating. And the entries in all other rows are computed as cubic functions of the entries in the reference row as follows.

Suppose the entries in the reference row for the points P_{i-1}, P_i, P_{i+1} and P_{i+2} are r_{-1}, r_0, r_1 and r_2, and that the corresponding entries in some other row are s_{-1}, s_0, s_1 and s_2. We define the quantities $\mu_{-1} = (r_{-1}-r_0)/(r_1-r_0)$ and $\mu_2 = (r_2-r_0)/(r_1-r_0)$. Then the equation for the entry in the row in which we are interpolating at a fraction f of the distance from P_0 to P_1 is $a_0+a_1f+a_2f^2+a_3f^3$, where

$$
\begin{aligned}
a_0 &= s_0,\\
a_1 &= \mu_2 t_{-1}+(\mu_{-1}+\mu_2+\mu_{-1}\mu_2)t_0+\mu_{-1}\mu_2 t_1+\mu_{-1}t_2,\\
a_2 &= -(1+\mu_2)t_{-1}-(1+\mu_{-1}+\mu_2)t_0-(\mu_{-1}+\mu_2)t_1-(1+\mu_{-1})t_2,\\
a_3 &= t_{-1}+t_0+t_1+t_2,
\end{aligned}
$$

and

$$
\begin{aligned}
t_{-1} &= s_{-1}/\mu_{-1}(\mu_{-1}-1)(\mu_{-1}-\mu_2),\\
t_0 &= -s_0/\mu_{-1}\mu_2,\\
t_1 &= s_1/(1-\mu_{-1})(1-\mu_2),\\
t_2 &= s_2/(\mu_2-\mu_{-1})(\mu_2-1)\mu_2.
\end{aligned}
$$

If the point P_{i+2} is undefined, the corresponding equation is $a_0+a_1f+a_2f^2$, where

$$
\begin{aligned}
a_0 &= s_0,\\
a_1 &= -t_{-1}-(1+\mu_{-1})t_0+\mu_{-1}t_1,\\
a_2 &= t_{-1}+t_0+t_1,
\end{aligned}
$$

and

$$
\begin{aligned}
t_{-1} &= s_{-1}/\mu_{-1}(\mu_{-1}-1),\\
t_0 &= -s_0/\mu_{-1},\\
t_1 &= s_1/(1-\mu_{-1}).
\end{aligned}
$$

It is perhaps worth noting that there is no difficulty if one has a problem in which some nonlinear functions are continuous, interpolable, functions while others are genuinely piecewise linear functions. All one need do is avoid naming the reference row for the piecewise linear function in the conventional way.

This interpolation scheme effectively turns separable programming into a fully-fledged nonlinear programming scheme. It is also in the spirit of modern mathematical programming in that – like decomposition – it

exploits the very powerful idea that variables should be generated as and when required, rather than written down explicitly in advance.

But another, earlier, scheme used in the Standard Oil of California code is worth mentioning. Here one starts using only every 8th point in the grid specified – then, having optimized on this basis, one uses every 4th point of the original grid, etc. This scheme works quite smoothly. But it requires more work at the matrix generation stage, and it has the disadvantage that the grid refinement is done over both relevant and irrelevant parts of the grid – with consequent limitations on the extent of the refinement possible in the most relevant part.

VI.5. Other uses for interpolation

The interpolation scheme can obviously be applied to convex problems for which one does not need the separable mode. For example it can be applied to convex quadratic problems if one does not have a quadratic programming code. It is curious that this approach to quadratic programming, originally proposed by Charnes and Lemke [19] before any genuine quadratic programming procedures had been published, should again become attractive.

Another interesting application of interpolation is to stochastic linear programming. It is well known that linear programming with random elements is a difficult subject. But one area that seems less unpromising than others is to problems involving uncertain righthand sides for some equations, and a (real or hypothetical) penalty associated with any discrepancy between the unknown true value of this right hand side and the value assumed in the linear programming solution. If the true value has a known probability distribution, then this leads to a problem with a convex cost component depending on the assumed value of this right hand side. Beale [23] discussed this problem. He suggested that it might be important as a means of incorporating the assessment of realistic safety margins within a linear programming model. He also suggested solving the problem by quadratic programming. It now appears that this last suggestion may have been more ingenious than important. A direct solution, using separable programming in the original sense proposed by Charnes and Lemke, is likely to be more efficient and more flexible if combined with the interpolation procedure just described.

VII. METHODS OF APPROXIMATION PROGRAMMING

VII.1. Introduction

At one time I thought that separable programming was an adequate tool for all nonlinear programming problems, provided that they were sufficiently nearly linear to justify searching for a local optimum solution. But I now realize that this is not so, as indeed Miller [17] states.

Separable programming can obviously deal with nonlinear functions of single variables, which quite frequently arise directly. It can also deal with "multilinear programming problems", i.e. with models built up from linear factors that combine together to make nonlinear terms. For example some partial derivative might be proportional to some variable and inversely proportional to the square of another. It can also deal with some other nonlinear programming problems, notably with quadratic constraints.

But in some problems one has to deal with complicated functions of several variables, that may not even be expressed analytically at all, although they can be computed for any particular values of their arguments.

The possibility of extending the ideas of separable programming to functions of several variables is discussed by Miller [17]. He shows that it can be done. But he points out that the number of points needed to define a nonlinear function of several variables adequately is likely to be large. In contrast to the one-dimensional problem, it will therefore be necessary to generate these points only as required. This involves working with a mixed matrix-generating and linear-programming system; and a preferable scheme on these lines seems to be that due to Griffith and Stewart [24]. This scheme is known as MAP, or the Method of Approximation Programming. This chapter describes this technique, and an extension of it.

VII.2. Linear approximation programming

The problem solved by the method of approximation programming can be stated as follows: Minimize

$$C = \sum_{j=1}^{n} a_{0j}x_j + g_0(y_1, y_2, \ldots, y_r) \tag{7.1}$$

subject to the constraints

$$\sum_{j=1}^{n} a_{ij}x_j + g_i(y_1, \cdots, y_r) = b_i \qquad (i = 1, \cdots, m), \tag{7.2}$$

$$L_k \leqq y_k \leqq U_k \qquad (k = 1, \cdots, r), \tag{7.3}$$

$$x_j \geqq 0 \qquad (j = 1, \cdots, n). \tag{7.4}$$

The method is to take trial values y_k^0 for the nonlinear variables y_k, and to linearize all nonlinear functions in the neighbourhood of y_k^0. If we write v_i for $g_i(y_1^0, \cdots, y_r^0)$, and w_{ik} for $\partial g_i/\partial y_k$ evaluated when all $y_k = y_k^0$, then the linearized problem becomes:
Minimize

$$C = \sum_{j=1}^{n} a_{0j}x_j + v_0 + \sum_{k=1}^{r} w_{0k}\Delta y_k \tag{7.5}$$

subject to the constraints

$$\sum_{j=1}^{n} a_{ij}x_j + v_i + \sum_{k=1}^{r} w_{ik}\Delta y_k = b_i \qquad (i = 1, \cdots, m) \tag{7.6}$$

$$L_k - y_k^0 \leqq \Delta y_k \leqq U_k - y_k^0 \qquad (k = 1, \cdots, r) \tag{7.7}$$

$$x_j \geqq 0 \qquad (j = 1, \cdots, n). \tag{7.8}$$

But, since these linear approximations may become invalid for large values of the Δy_k, the conditions (7.7) are replaced by

$$L_k' \leqq \Delta y_k \leqq U_k', \tag{7.9}$$

where

$$L_k' = \max(L_k - y_k^0, -m_k) \text{ and } U_k' = \min(U_k - y_k^0, m_k) \tag{7.10}$$

for some suitably small bound m_k.

The next trial solution is then given by

$$y_k = y_k^0 + \Delta y_k \qquad (k = 1, \cdots, r). \tag{7.11}$$

The unrestricted variable y_k can be handled in various ways. Perhaps the simplest is to replace it by the nonnegative variable

$$z_k = \Delta y_k - L_k'. \tag{7.12}$$

The problem then becomes:
Minimize

$$C = v_0 + \sum_{k=1}^{r} w_{0k} L'_k + \sum_{j=1}^{n} a_{0j} x_j + \sum_{r=1}^{k} w_{0k} z_k, \tag{7.13}$$

subject to the constraints

$$\sum_{j=1}^{n} a_{ij} x_j + \sum_{k=1}^{r} w_{ik} z_k = b_i - v_i - \sum_{k=1}^{r} w_{ik} L'_k, \tag{7.14}$$

$$z_k \leqq U'_k + L'_k \tag{7.15}$$

$$x_j \geqq 0, \qquad z_k \geqq 0. \tag{7.16}$$

The bounding values m_k can be of course be adjusted in the light of the progress of the calculations. If a variable is consistently at its lower bound, or consistently at its upper bound, in successive iterations, then it is probably desirable to increase the corresponding m_k. Conversely, if it oscillates, a lower value of m_k is desirable.

To apply this procedure, one must of course abandon the mathematical programming code at the end of each "major iteration", i.e. when the linear approximate program has been solved, and revert to the matrix generator program to deduce revised values for the v_i, the w_{ik}, and the U'_k and L'_k. The linear programming calculations are then restarted by reinverting to the optimum basis obtained on the previous major iteration. Griffith and Stewart point out that it is not difficult to tie a FORTRAN matrix generator and an LP system together into a completely automatic system. And the necessity to use a computer program to generate the matrix is certainly no disadvantage. Even for a purely linear programming problem it is generally more convenient to write a special purpose FORTRAN program to generate the matrix rather than punch up the data manually. The problem is in its most expanded form at the point when it is set out explicitly as equations for the mathematical programming routine, so this is nearly always a very poor moment to enter the computer for the first time.

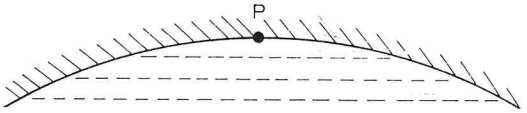
It is of interest to note that the procedure of Baumol and Wolfe [25] is a special case of linear approximation programming. They were concerned with a particular problem in which the constraints are linear and the objective function to be minimized is concave. There is then no need to replace the contraints (7.7) by the more restrictive constraints (7.9), since the true value of the objective function is never greater than the linear approximation.

VII.3. Quadratic approximation programming

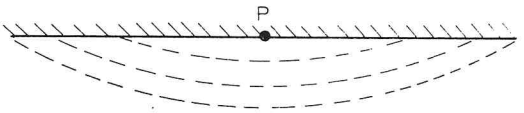
In spite of its computational successes, MAP has not been enthusiastically received by mathematical programming theorists. This may be due to a feeling that it is unnatural to make a purely linear approximation to an optimization problem whose solution may not be at a vertex of the feasible region. For example, if one is minimizing x^2-2x subject to the constraints $x \geqq 0$, $x \leqq 2$, then the solution is $x = 1$. But if one tackles this problem by a succession of purely linear approximations one will always find the trial value of x at either the upper or the lower bound arbitrarily imposed on it by the current approximation. In fact it will be at the upper bound if the previous trial value was less than 1, and at the lower bound otherwise. By carefully reducing the range of permitted values of x when the trial value starts to oscillate, one can eventually reach a solution as near to 1 as one pleases. But the process can hardly be described as an elegant one.

The point, as mentioned in Chapter II, is that, unless one knows that constraints dominate the problem, the natural approximation to an objective function is a quadratic one. On the other hand it would be fatuous to make a quadratic approximation to the objective function if one ignored the effects of the nonlinearities in the constraints. This would mean for example that one could solve the problem just considered without any trouble, but would be no better off than when using linear approximations throughout if the problem were to minimize z subject to the nonlinear constraint $z \geqq x^2-2x$.

The solution to this difficulty can be visualized geometrically as a distortion of the feasible region that straightens out the constraints at the expense of the contours of constant values of the objective function. If these contours are drawn as broken lines, we must transform a problem looking like



into one looking like



Rosen [26] showed that this could be done algebraically using the shadow prices on the constraints to throw their nonlinearities into the objective

function. Rosen's approach to the problem is described in detail in the next chapter.

The idea of making a local quadratic programming approximation on these lines was developed by Wilson [27], and implemented in a general FORTRAN nonlinear programming code named SOLVER. The theory of Wilson's work is based on making local linear approximations to the Kuhn-Tucker conditions, and solving them using Dantzig's version of Wolfe's quadratic programming algorithm. In its original form this method may fail if the problem is not convex; but this difficulty can be overcome by using a Levenberg parameter as in unconstrained minimization. The only serious limitation of SOLVER is therefore in problem size – the total number of variables plus equations cannot exceed about 100. To cope in this way with a problem containing a large number of linear variables it seems necessary to use Beale's quadratic programming algorithm.

The theory developed below is essentially equivalent to Wilson's, but it is expressed in terms of the quadratic programming approximation, and not in terms of the Kuhn-Tucker conditions.

Consider the problem defined by (7.1) to (7.4), and consider an arbitrary set of values of the nonlinear variables $(y_1, \cdots, y_r)$, say $y_k = y_k^0$. Suppose that all the functions $g_i(y_1, \cdots, y_r)$ are twice differentiable at the point $(y_1^0, \cdots, y_r^0)$. As before, we denote the values of g_i and $\partial g_i/\partial y_k$ at this point by v_i and w_{ik} respectively. We also denote the value of $\frac{1}{2}\partial^2 g_i/\partial y_k \partial y_l$ by w_{ikl}.

Now denote the shadow prices on the equations (7.2) by π_i^0. These represent the values of $\partial C/\partial b_i$, treating the nonlinear variables y_r as constants, and assuming that the operative constraints remain those that are operative at the optimal solution to the problem conditional on all nonlinear variables taking their present values. We also define $\pi_0^0 = 1$.

We now prove a theorem about the relationship of the original problem with the following approximate problem:
Minimize

$$C_{\text{app}} = v_0 + \sum_{j=1}^{n} a_{0j}x_j + \sum_{k=1}^{r} w_{0k}\Delta y_k + \sum_{k=1}^{r}\sum_{l=1}^{r}\left(\sum_{i=0}^{m}\pi_i^0 w_{ik}\right)\Delta y_k \Delta y_l \qquad (7.17)$$

subject to the constraints (7.6) to (7.8).

The theorem is that

(a) throughout the region in the space of the nonlinear variables where the same constraints are operative as at $(y_1^0, \cdots, y_r^0)$, $C_{\text{app}} = C + o(\Delta y)^2$. Consequently the error in the equation of the hypersurface at which the

partial derivative of C with respect to any nonbasic variable vanishes is $o(\Delta y)$ within this region,

(b) the error in the equation of the hypersurface at which any new constraint becomes operative is $o(\Delta y)$.

The proof of part (b) of the theorem is immediate, since (7.6) is the same as (7.2) to this order of accuracy.

The proof of part (a) rests on the fact that there exist shadow prices π_i (with $\pi_0 = 1$) such that an optimal solution to the original problem remains optimal if the nonlinear constraints are withdrawn and the objective function is replaced by

$$C = -\sum_{i=1}^{m} \pi_i b_i + \sum_{j=1}^{n} \sum_{i=0}^{m} \pi_i a_{ij} x_j + \sum_{i=0}^{m} \pi_i g_i(y_1, \cdots, y_r). \tag{7.18}$$

Replacing the g_i by their power series approximations, and using (7.6), this expression becomes

$$C = v_0 + \sum_{j=1}^{n} a_{0j} x_j + \sum_{k=1}^{r} w_{0r} \Delta y_r + \sum_{k=1}^{r} \sum_{l=1}^{r} \left(\sum_{i=0}^{m} \pi_i w_{ik} \right) \Delta y_k \Delta y_l + o(\Delta y)^2. \tag{7.19}$$

Furthermore, $\pi_i = \pi_i^0 + O(\Delta y)$, since, by hypothesis, no new constraint becomes operative. So the theorem is proved.

If one has access to an effective quadratic programming code, the following procedure is therefore attractive.

Take a set of trial values y_k^0 for the nonlinear variables, and a set of trial shadow prices – which one hopes will not differ substantially from the π_i^0 of the theorem. Then minimize C_{app} subject to the constraints (7.6), (7.8) and (7.9).

The next trial values for the nonlinear variables are given by $y_k^0 + \Delta y_k$, as with linear approximation programming. The next trial values for the π_i^0 are the final shadow prices on the constraints (7.6).

It is not obvious whether it is better to retain the constraints (7.9), or whether to revert to the basic constraints (7.7) (some of which may be vacuous) and add a term

$$\lambda \sum_{k=1}^{r} m_k^2 \Delta y_k^2$$

to the objective function. This term would of course play the same part as the Levenberg parameter in unconstrained minimization problems. On the whole I think the Levenberg approach is preferable. In either case

one would have to be careful about the choice of values of the scaling factors m_k; though the problem is not quite as acute as with linear approximation programming, since convergence does not depend on any scaling factors tending to zero.

The analogy with unconstrained minimization procedures would be strengthened if, in each major iteration, one first solved the mathematical programming problem with all $\Delta y_k = 0$. If the solution gave a larger value of C than the corresponding solution at the previous major iteration, then progress is not being made. It might then be desirable to return to the previous iteration and either increase λ or decrease the m_k. But in practice it will probably be most convenient to control these parameters from the behaviour of the final approximate values of the objective function at each major iteration.

This quadratic approximation programming procedure, if properly handled, should converge in fewer major iterations than the original linear approximation programming procedure. And this could in itself be a useful advantage when significant amounts of time are taken in changing from mathematical programming to matrix generation and back again. But will this saving compensate for the extra work in each major iteration? This question cannot be answered authoritatively without practical experience of the procedure, but the following comments seem relevant.

There may be a large number of second derivatives w_{ikl} involved in the problem. It might therefore be very time-consuming to have to recompute these in each major iteration. But this should not be necessary. It does not really matter if these terms are slightly inaccurate, since they are in the nature of correction terms anyway. It might even be acceptable to rely entirely on information about the way the w_{ik} vary with the trial solution to correct errors in estimating the w_{ikl}.

If there are any nonbasic free variables in the optimum tableau for one major iteration, it will probably not be feasible to replace the equations defining them in the matrix for the next iteration. Some suitable Δy_k can be used instead as nonbasic free variables in the initial tableau. They should be marked as "condemned" free variables, and removed from the set of nonbasic variables as soon as possible. The presence of such condemned variables may prevent the mathematical programming phase of any major iteration taking a negligible time; but in the corresponding situation in linear approximation programming the nonlinear variables are fluctuating arbitrarily between their upper and lower bounds, a process which may also take a little time.

From time to time, perhaps every second major iteration, one could update the approximate problem without reference to the original problem, by simply substituting improved shadow prices in the expression for C_{app}, and using the w_{ikl} to update the linear approximations to the nonlinear constraints in the neighbourhood of the current trial solution. Since this procedure does not refer to the original problem, it could be programmed as an agendum of the mathematical programming system, with some saving in running time.

Although it is fairly straightforward, this addition is not recommended for a first version of quadratic approximation programming. But a minor refinement that is recommended, and is perhaps applied in linear approximation programming, is to cut off the quadratic programming algorithm if it has not terminated after some prescribed number of minor iterations. There is no point in spending a lot of time putting the final touches on the quadratic programming solution to a problem that is just about to be revised.

VII.4. Semi-automatic approximation programming

Even without special facilities, Methods of Approximation Programming can be applied quite effectively in a semi-automatic way, in which one comes off the computer after each major iteration, i.e. after solving each approximate program. To make this procedure reasonably efficient one must store the results of the mathematical programming problem in a form that can easily be read by another program – perhaps a FORTRAN program – that generates the matrix for the next major iteration. This may be accomplished using a special output magnetic tape, sometimes known as an UNRAVEL tape. Such a facility is in any case required to enable one to make suitable output analysis programs for the results of mathematical programming calculations.

There are compensations for the loss of elapsed time in this semi-automatic use of approximation programming. By examining the solutions at the end of each major iteration, one has more control over the calculations. Specifically, it is easier to detect errors in the formulation of the problem, or in the assembly of the mathematical programming approximation, and one can make better decisions about when to stop and what values of m_k to use than are feasible in a fully automatic general program.

One may be able to reduce the inconvenience of switching over from FORTRAN to mathematical programming by solving several problems in parallel. This inconvenience will still remain, but, particularly with quadratic approximation programming, one may get an adequate solution to the problem in a very few major iterations – say not more than 6.

VIII. DECOMPOSITION AND PARTITIONING METHODS FOR NONLINEAR PROGRAMMING

VIII.1. Introduction

Decomposition and partitioning methods are becoming increasingly important in linear programming. The most important of these seem to be:

(a) Dantzig and Wolfe's decomposition principle, for linear programming problems consisting of separate subproblems except for a relatively small number of linking equations. See Dantzig and Wolfe [28], Dantzig and Wolfe [29] and Dantzig [20].

(b) Primal schemes for linear programming problems consisting of separate subproblems except for a relatively small number of linking variables. See Rosen [26] and Beale [30].

(c) The dual to (a), which is a dual method for the linking variables problem developed independently by Benders. See Benders [31] and Benders [32].

(d) The dual to (b), which is a dual method for the linking equations problem, developed by Rosen [33] under the name Primal Partition Programming.

This chapter contains some remarks about the application of each of these approaches to nonlinear programming. Since we are still on the thresholds of the development of such methods, these remarks will be somewhat tentative. But the subject is an extremely important one, so it seems very worth while to try to summarize the present situation.

VIII.2. Nonlinear programming by decomposition

The only version of nonlinear programming using either decomposition or partitioning methods of which I have any practical experience is one that is theoretically unsound, though it often works quite well in practice. This is the use of separable programming combined with the Dantzig-Wolfe decompositions procedure. There is no objection to introducing special variables into the master problem. But one can run into trouble using special variables in a subproblem. This is because, although one can prevent any

individual proposal from a subproblem using illegitimate combinations of special variables, there is no way of preventing the master problem from combining proposals in a way that implicitly combines special variables illegitimately. With the C-E-I-R decomposition code, it is easy to see after the event whether or not this has caused trouble. When the master problem has been solved one finds the value of the variables in the subproblems by solving the subproblems with right hand sides assigned to the common rows (including the objective function) in accordance with the values given by the master problem. The solution is then justified if and only if the value of the objective function (to be minimized) in this problem is not positive.

Since the idea of being able to take positive linear combinations of proposals is fundamental to the decomposition method, and since this is essentially a property of convex regions, and since separable programming is essentially a device for handling non-convex problems, it might seem that these 2 ideas had little to offer each other. But I believe that in practice they will be found to combine together to solve many important problems. The point is of course that all the linear combinations of proposals that could arise in an optimum solution may be legitimate even if the feasible region is not convex. A trivial example arises if the feasible region consists of the boundary of a convex region. This is not convex. But if one knew that the optimum solution must remain on the boundary of the feasible region even if one were to allow interior points, then a decomposition approach would certainly work. Let us now consider a less trivial example. This may seem rather specialized, but I suspect that it is more general than it looks.

Suppose we have a problem containing the constraint

$$x_1 x_2 \geqq c,$$

where $x_1 \geqq 0$, $x_2 \geqq 0$. This constraint defines a convex region. And there is no need to use separable programming to represent it, since we can write the constraint in the form

$$x_1 \geqq y_2,$$

where $y_2 = c/x_2$. It does not matter if we allow nonadjacent vectors in the piecewise linear representation of y_2 as a function of x_2, since this will only increase the assumed value of y_2.

But now consider the constraint

$$x_1 x_2 \geqq x_3.$$

This constraint is naturally handled using the standard separable programming procedure for products. Furthermore it is not convex. For suppose

$$x_1 x_2 \geqq x_3$$
$$(x_1+y_1)\cdot(x_2+y_2) \geqq x_3+y_3,$$

it does not follow that

$$(x_1+\theta y_1)\cdot(x_2+\theta y_2) \geqq x_3+\theta y_3$$

for $0 < \theta < 1$.

The left hand side of this tentative inequality can be written as

$$(1-\theta)x_1 x_2+\theta(x_1+y_1)\cdot(x_2+y_2)+\theta(\theta-1)y_1 y_2.$$

This is greater than the right hand side if $\theta(\theta-1)y_1 y_2 > 0$, but not necessarily otherwise. For this last inequality to hold, we must have y_1 and y_2 of opposite signs. There is no reason why this must be so, but it often is. For example, if the product represents the number of wells multiplied by well productivity, one may know that in practice the more wells one has the lower will be the well productivity.

There is another, less specific, defense of separable programming in decomposition subproblems. In a sense this says only that real problems are effectively convex, but I think it is nevertheless comforting. Suppose that the optimum solution to the master problem uses an illegitimate combination of proposals from some subproblem. This means that it would be profitable to operate the plant represented by this subproblem in a non-uniform way. For example, an instruction to use Proposal 1 with weight 0.75 and Proposal 2 with weight 0.25 could be interpreted as operating the plant in Condition 1 for 18 hours per day and in Condition 2 for 6 hours per day. If this can be done without significant changeover penalties, then the problem is automatically convex. If it cannot, then such a solution is infeasible; but one may feel instinctively that it could not be optimum even if it were feasible. This last point can be put another way. If one had 2 plants, each of half the capacity of the actual one, performing the operation represented by this subproblem, then one may feel that it could not be optimum to operate these plants in different ways.

So much for separable programming in decomposition. What about the prospects for applying other standard nonlinear programming techniques? There is no reason in principle why these should not be used, as long as one ensures that any subproblem is truly feasible whenever a proposal is offered to the master. For example if one used the Method of Approximation

Programming described in the previous chapter, in either the linear or the quadratic form, one should only offer proposals while all $\Delta y_k = 0$. The C-E-I-R work on decomposition strongly suggests that one should put up several proposals from each subproblem in each major iteration, and also that one should cut off any subproblem that fails to become optimum in some fixed number of iterations. Using the Method of Approximation Programming one would presumably operate in the same way, putting up intermediate proposals with all $\Delta y_k = 0$ (as long as the subproblem is feasible). To avoid being stuck with fixed values of the nonlinear variables, one would have to continue iterating after the cut-off time, without any restrictions on the Δy_k but without putting up any further proposals.

The Method of Approximation Programming, as well as some others, might need more tape units than are available in order to work smoothly. The C-E-I-R Decomposition Code uses 11 tape units at one time, although this could be reduced to 10 if necessary. And some extra units are likely to be needed for the matrix generating parts of MAP.

Wolfe [12] points out that, at least in principle, decomposition is a natural method for solving convex programming problems, using some special purpose algorithm to solve the subproblems and generate the required proposals. In appropriate circumstances this should be an effective procedure. It is not yet clear how widely applicable it will prove to be.

VIII.3. Nonlinear partition programming

Rosen [26] and Rosen and Ornea [34] have discussed the application of partition programming to problems in which the constraints may be non-linear in the linking variables. They report satisfactory computational experience using a program written for the IBM 7090 computer.

The problem can be written as follows:
Minimize

$$\sum_{l=1}^{s} \sum_{j=1}^{n_l} a_{0jl} x_{jl} + g_0(y_1, \cdots, y_r), \tag{8.1}$$

subject to the constraints

$$\sum_{j=1}^{n} a_{ijl} x_{jl} + g_{il}(y_1, \cdots, y_r) = b_{il} \qquad (l = 1, \ldots, s; \quad i = 1, \cdots, m_l) \tag{8.2}$$

$$x_{jl} \geqq 0 \qquad (l = 1, \cdots, s; \quad j = 1, \cdots, n_l). \tag{8.3}$$

This description of the problem differs in detail from Rosen's in order to emphasize the connexion between this work and that described in the last

chapter. If s, the number of subproblems, equals 1 the problem is essentially the same as that treated in the last chapter. Rosen's method can be applied in these circumstances, but it is particularly powerful if $s > 1$.

The procedure is as follows.

Start with an arbitrary value of the vector $\boldsymbol{y}$, say $\boldsymbol{y}^0 = (y_1^0, \cdots y_r^0)$. The problem then reduces to s linear programming problems that can be solved separately for the x_{jl}. As a by-product of this operation we will have a set of basic unlinked variables x_{jl}, and a set of nonbasic unlinked variables x_{jl} that take the value zero. We will also have a set of reduced costs (or D/Js) on all these variables that can be represented by the vector d^l. Finally, we shall have a set of shadow prices π_{il} on the rows of (8.2).

Rosen calls this part of the work Problem 1. Having solved Problem 1 for all subproblems, we now consider the problem of selecting a better value for $\boldsymbol{y}$ keeping all the nonbasic unlinked variables equal to zero. Rosen calls this Problem 2. In principle it has nonlinear constraints, but we approximate it by one with a nonlinear objective function and linear constraints. Specifically, the objective function is

$$g_0(y_1, \cdots, y_r) + \sum_{l=1}^{s} \sum_{i=1}^{m_l} \pi_{il} g_{il}(y_1, \cdots, y_r),$$

and the constraints are obtained from the local linear approximation to (8.2). At this stage we keep all nonbasic unlinked variables nonbasic, so the constraints reduce to $\sum m_l$ linear inequalities on the variables $y_1, \cdots, y_r$.

The solution to the approximate Problem 2 will be a new vector $\boldsymbol{y}^1$. But in view of the approximations made it may not be a feasible value of $\boldsymbol{y}$. We, therefore, complete the "major cycle" with a one-dimensional search for a value of $\boldsymbol{y}$ of the form $\boldsymbol{y}^0 + \theta(\boldsymbol{y}^1 - \boldsymbol{y}^0)$ that minimizes the objective function subject to the requirement that the problem must remain feasible with all the nonbasic unlinked variables equal to zero. The whole process can now be repeated, using the resulting value of $\boldsymbol{y}$ as the new $\boldsymbol{y}^0$.

Theorem 2 in Rosen [26] is important. It shows that one can recognize whether a set of values for the unlinked and linking variables that represent a local optimum in Problems 1 and 2 separately also represents a local optimum for the whole problem. This will not necessarily be so, because the optimum values of the linking variables will generally cause degeneracy in one or more subproblems. The difficulty can perhaps be seen more easily in Beale's version of the method – see Beale [30]. Problem 1 and 2 are both solved, but the problem as a whole is not, whenever a nonbasic variable has to be made pseudobasic in order to make it basic subsequently. The

point is that, in cases of degeneracy, the shadow prices on the constraints may not be the same in Problems 1 and 2. And the Problem 2 shadow prices, multiplied by the entries in the current Problem 1 tableaux, must be added to the vector of reduced costs for the nonbasic variables in Problem 1 in order to determine the true costs on these variables for the problem as a whole. If any of these variables has a negative true reduced cost, the overall solution can be improved by making one or more of them basic in Problem 1, in place of others that are basic but take the value zero. When we revert to Problem 2, we will no longer insist on such variables remaining equal to zero, and further progress can be made.

Rosen expresses the condition that the true reduced cost must remain nonnegative in the form

$$\boldsymbol{d} - Q\boldsymbol{v} \geqq 0,$$

where the matrix Q denotes the current tableau for some Problem 1, and the vector $\boldsymbol{v}$ denotes the set of shadow prices in Problem 2 on the inequality constraints of this subproblem.

This is really all there is to the theory of convex partition programming. Rosen's program in fact uses the gradient projection method, described by Rosen [35], for both the linear Problem 1 and the nonlinear Problem 2; but this is not an essential feature of the approach. Nor are the details of the way the nonlinear Problem 2 is approximated by one involving only linear constraints. No particular theoretical properties are claimed for the method in this form. Rosen has admitted in a private communication that his Theorem 3, that purported to prove that the procedure must terminate after a finite number of major cycles, is false. However, the procedure will usually terminate in a finite number of major cycles for the reasons indicated by Rosen [26]. These are that, in principle, we are optimizing the system in each major cycle for some set of bases in each subproblem. Either this is a true (local) optimum to the problem or else we can improve on it by choosing another set of bases. Since the objective function improves with each major cycle, we can never return to a set of bases previously considered, and there is only a finite number of possible bases for each subproblem.

The use of other nonlinear programming methods, such as separable programming, within the subproblems in partition programming is in principle straightforward and can cause no difficulty. Nonconvexity is not a problem with this approach. This is basically because, as Beale [30] emphasizes, partition programming can be treated as a method of applying

the precise logic of the simplex method to the problem as a whole without losing the benefit of the special structure.

VIII.4 Dual methods

Dual methods tend to be more difficult to work with than primal methods. Nevertheless, as Kelley [36] illustrates, they may have advantages for convex programming problems. This is essentially because such problems can be thought of as consisting of a finite number of variables subject to an infinite number of linear inequality constraints. One approximates to the solution by making a strategic finite selection from this infinite number, adding to it until an adequate approximation to the solution has been obtained.

Benders's partition programming scheme was conceived in the same spirit to deal with mixed problems of various sorts. See Benders [31] and Benders [32]. Among other problems, he was interested in that described in the previous section. His methods solve this problem on the following lines. Consider an arbitrary vector $\boldsymbol{y}$ for the linking variables. If we optimize each subproblem for this value of y we will have expressed the contribution of this subproblem to the objective function as some function of y plus a nonnegative linear function of the nonbasic variables of the subproblem. We therefore have a constraint involving only the linking variables to the effect that $\sum_j a_{0jl}x_{jl}$ is greater than or equal to the function of y mentioned above. At each major cycle we can choose as our trial value of y that which would minimize the objective function if the only constraints on the linking variables were those that we have deduced in previous major cycles.

It is clear that this approach is the dual to the Dantzig-Wolfe decomposition scheme. It is theoretically limited to convex subproblems in the same way as the Dantzig-Wolfe scheme. In fact it does not seem to be a very natural tool for continuous nonlinear programming; but it has considerable promise for integer programming, see Harris [37], so it may ultimately prove useful when seeking global optima to more general nonconvex problems.

The dual to partition programming, developed by Rosen [33] under the name of Primal Partition programming, is a dual method for solving the linking equations problem. Like the original partition programming scheme it can in principle be applied to nonlinear problems in which the subproblems are not convex. The application of the dual simplex method to a subproblem containing separable programming special variables is a little complicated, but by no means impossibly so. The simplest approach is probably to put just one special variable in each group equal to 1, keeping the others equal

to zero, until the problem is solved with respect to the other variables. One can then sort out the special variables using the primal simplex method.

I hope that this rather sketchy outline gives some idea of the scope for decomposition and partitioning methods in nonlinear programming. It does not seem feasible to try to do more at this stage in the development of the subject.

REFERENCES

1. Wilson, E. B. and R. R. Puffer, Least squares and laws of population growth, Proc. Amer. Acad. Arts. Sci. **68**, 285–382 (1933).
2. Beale, E. M. L., Confidence regions in nonlinear estimation, J. Roy. Statist. Soc. (B) **22**, 41–88 (1960).
3. Box, G. E. P., Use of statistical methods in the elucidation of basic mechanisms, Bull. Inst. Intern. Statist. **36**, 215–225 (1958).
4. Hartley, H. O., The modified Gauss-Newton method for the fitting of non-linear regression functions by least squares, Technometrics **2**, 269–280 (1961).
5. Marquardt, D. W., An algorithm for least squares estimation of non-linear parameters, J. Soc. Indust. Appl. Math. **11**, 431–441 (1963).
6. Levenberg, K., A method for the solution of certain nonlinear problems in least squares, Quart. Appl. Math. **2**, 164–168 (1944).
7. Stiefel, F., An introduction to numerical mathematics (Academic Press, New York and London, 1963).
8. Künzi, H. P. and W. Krelle, Nichtlineare Programmierung (Springer-Verlag, 1962).
9. Wolfe, P., The simplex method for quadratic programming, Econometrica **27**, 382–398 (1959).
10. Beale, E. M. L., On minimizing a convex function subject to linear inequalities, J. Roy. Statist. Soc. (B) **17**, 173–184 (1955).
11. Beale, E. M. L., On quadratic programming, Naval Res. Logistics Quarterly **6**, 227–243 (1959).
12. Wolfe, P., Recent advances in mathematical programming, eds. R. L. Graves and P. Wolfe (Mc Graw Hill, 1963) p. 67–86.
13. Beale, E. M. L., An alternative method for linear programming, Proc. Camb. Phil. Soc. **50**, 513–523 (1954).
14. Zoutendijk, G., Methods of feasible directions (Elsevier, 1960).
15. Wolfe, P. and L. Cutler, Recent advances in mathematical programming, eds. R. L. Graves and P. Wolfe (Mc Graw Hill, 1963) p. 177–200.
16. Smith, D. M. and W. Orchard-Hays, Recent Advances in Mathematical Programming, eds. R. L. Graves and P. Wolfe (Mc Graw Hill, 1963) p. 211–218.
17. Miller, C. E., Recent advances in mathematical programming, eds. R. L. Graves and P. Wolfe (Mc Graw Hill, 1963) p. 89–100.
18. Beale, E. M. L., P. J. Coen and A. D. J. Flowerdew, Separable programming applied to an ore-purchasing problem, Appl. Stat. **14**, 89–101 (1965).
19. Charnes, A., and C. E. Lemke, Minimization of non-linear separable functionals, Naval Res. Logistics Quarterly **1**, 301–312 (1954).
20. Dantzig G. B., Linear programming and extensions (Princeton University Press, 1963).
21. Markowitz, H. M. and A. S. Manne, On the solution of discrete programming problems, Econometrica **25**, 84–100 (1957).
22. Dantzig, G. B., On the significance of solving linear programming problems with some integer variables, Econometrica **28**, 30–44 (1960).

23. Beale, E. M. L., On the use of quadratic programming in stochastic linear programming, Paper presented at the TIMS conference in Brussels, August 1961, RAND paper P-2404.
24. Griffith, R. E. and R. A. Stewart, A nonlinear programming technique for the optimization of continuous processing systems, Management Science **7**, 379–392 (1961).
25. Baumol, W. J. and P. Wolfe, A warehouse-location problem, Operations Research **6**, 252–263 (1958).
26. Rosen, J. B., Recent advances in mathematical programming, eds. R. L. Graves and P. Wolfe (Mc Graw Hill, 1963) p. 159–176.
27. Wilson, R. B., A simplicial algorithm for concave programming; Doctoral dissertation, Graduate School of Business Administration, Harvard University, Boston (1963)
28. Dantzig, G. B. and P. Wolfe, Decomposition principle for linear programs, Operations Reserach **8**, 101–111 (1960).
29. Dantzig, G. B. and P. Wolfe, The decomposition algorithm for linear programs, Econometrica **29**, 767–778 (1961).
30. Beale, E. M. L., Recent advances in mathematical programming, eds. R. L. Graves and P. Wolfe (Mc Graw Hill, 1963) p. 133–148.
31. Benders, J. F., Partitioning in mathematical programming, Thesis University of Utrecht (1960).
32. Benders, J. F., Partitioning procedures for solving mixed-variables programming problems, Numerische Mathematik **4**, 238–252 (1962).
33. Rosen, J. B., Primal partition programming for block diagonal matrices, Numerische Mathematik **6**, 250–260 (1964).
34. Rosen, J. B. and J. C. Ornea, Solution of nonlinear programming problems by partitioning, Management Science **10**, 160–173 (1963).
35. Rosen, J. B., The gradient projection method for nonlinear programming, Part I: Linear constraints, J. Soc. Industr. Appl. Math. **8**, 181–217 (1960).
36. Kelley, J. E., Jr., The cutting-plane method for solving convex programs, J. Soc. Industr. Appl. Math. **8**, 703–712 (1960).
37. Harris, P. M. J., An algorithm for solving mixed integer linear programmes, Operational Research Quarterly **15**, 117–132 (1964).