AN ALGORITHM FOR SOLVING A CONSTRAINED LEAST SQUARES PROBLEM for estimating the effects of an unknown monotonically intervening factor

by

Shalini Wadhwa

Submitted in partial fulfillment of the requirements

for the Degree of

Master of Science

in the

Mathematics

Program

[illegible] 3/27/91

Advisor Date

Sally M. Hotchkiss April 10, 1991

Dean of the Graduate School Date

YOUNGSTOWN STATE UNIVERSITY

March, 1991

ABSTRACT

AN ALGORITHM FOR SOLVING A CONSTRAINED LEAST SQUARES PROBLEM for estimating the effects of an unknown monotonically intervening factor

Y-10-4

by

Shalini Wadhwa

Master of Science in Mathematics

Youngstown State University, 1991

The conventional least squares method cannot estimate any hidden causal factor which is assumed to shift the regression hyperplane monotonically upwards (downwards). A constrained least squares problem has been formulated by *S. Thore* to model this phenomenon.

A compact and efficient algorithm is developed and presented to solve this constrained problem. Computational results are presented which illustrate the effectiveness of this algorithm. The new method is then used to estimate the price elasticity of cigarettes and the monotonic shifts of the demand curve. A causal factor is found which may reflect the awareness among the general public of the harmful effects of smoking.

ACKNOWLEDGEMENTS

I wish to express my gratitude to my thesis advisor, *Dr. Nathan P. Ritchey*, who spent endless hours reviewing my work and inspired me to successfully complete this thesis.

I would also like to thank *Dr. John J. Buoni* and *Dr. Samuel F. Barger* for their contribution and advice in improvement of this thesis.

Finally, I would like to thank *Mrs. Ritchey* for her helpful suggestions.

Table of Contents

1. Introduction

1.0 Preview 6

1.1 Reflections 7

1.2 Historical Background 8

1.3 Difficulties caused by nonlinearity 8

2. Theoretical Perspective

2.1 Convex and Concave functions 12

2.2 Maxima and Minima 14

2.3 Quadratic Programming 16

2.4 Lagrange Multipliers and Kuhn-Tucker Theory 17

2.5 Wolfe's Method 19

2.6 Convergence of Wolfe's Algorithm 24

2.7 Curve fitting and the Principle of Least Squares 25

2.8 The Constrained Least Squares Problem 28

3. Solving the Problem

3.1 Mathematical Analysis 30

3.2 Computer Implementation of the Method 40

3.3 The Demand for Cigarettes in the U. S (An Example) 42

3.4 Further Research 46

3.5 Conclusion 47

Bibliography 48

Appendix 50

Chapter 1

Introduction

1.0 Preview

This thesis will present a compact and efficient algorithm to solve a constrained least squares problem formulated by S. Thore (1988).

The first chapter covers the historical background of nonlinear programming problems, especially quadratic programming problems. Then, the difficulties caused by nonlinearity are discussed and illustrated with an example.

The second chapter introduces the theoretical aspects of the algorithm. It describes convex and concave functions since it is often necessary to determine whether the function is convex or concave in a mathematical program. Then, various types of optima are defined. A general form of the quadratic programming problem and a solution method given by the Kuhn-Tucker theory and Wolfe's method are presented. The principle of least squares is discussed. Finally, the constrained least squares problem is presented, as formulated by S.Thore.

The third chapter solves the problem presented in the previous chapter by size reduction methods like substitution and Tucker's condensed tableau method. These methods reduce the size of the problem considerably, as demonstrated by an example problem. The method is further illustrated with the "real life" problem of 'Demand of cigarettes in the U. S.', using the data which was collected by Dr. J. E. Harris. The problem is solved using the computer programs which appear in the Appendix. A causal factor is found which may reflect the awareness among the general public of the harmful effects of smoking.

A discussion of the final results of the cigarette problem is presented and illustrated with the help of a graph.

1.1 Reflections

For centuries mathematicians have been formulating and solving optimization problems and studying their theoretical properties. For example, Euclid knew how to find the point on the straight line $Ax + By = C$ that is closest to the origin. Using the compass and straightedge, he was minimizing $(x^2 + y^2)^{\frac{1}{2}}$ subject to $Ax + By = C$. In the seventeenth century Newton and Leibnitz derived the fundamental theorems of differential calculus, with which it became possible to find the maxima and minima of all continuous algebraic functions. By the nineteenth century the more specialized method of Lagrange multipliers had been developed for finding the solution of optimization problems subject to equality constraints.

However, before the advent of digital computers, a moderately large optimization problem, one involving, say, 100 constraints and 200 variables, could not be solved with the available theoretical results. In fact most of the efficient methods used today for solving simultaneous linear equations were known to Gauss. Nevertheless, he could not solve a problem consisting of 1000 such equations - and we can with our computers. Or again, if the simultaneous equations were nonlinear, the Newton-Raphson method could find a solution, provided that there weren't too many such equations. But in those days, even 10 or 15 were "too many".

Our pursuit has been much more successful because our computers give us the capability of applying our findings. This is particularly true of optimization problems subject to inequality constraints, which were terribly laborious for classical mathematicians and thus studied little by them.

Not only is a large-scale constrained optimization problem electronically feasible today, but in addition there is a great deal of interest in this area. For example modern economics, engineering, and management science have provided many large and complicated problems in the past 25 years which were difficult to formulate.

1.2 Historical Background

Operations research is an important branch of applied mathematics which is concerned with the formulation and solution of optimization problems. In operations research we use the term *mathematical program* to describe most of the optimization problems that have an objective function to minimize or maximize and constraints to satisfy.

The History of *mathematical programming* can be divided, generally speaking, into two parallel and occasionally convergent streams of development: one springing from Dantzig's simplex method (1940) and the other from Kuhn Tucker's optimality theory(1951).

In the mid 1950s a technique called separable programming was developed that allowed an approximate optimal solution to be obtained for certain types of problems. Later, this technique was generalized to all types of nonlinear programs. In this method, piecewise linear approximations were used in place of all nonlinear functions, enabling the problem to be solved by a modified simplex scheme.

Beginning in 1955, a number of papers dealing with quadratic programming began to appear. These include the works of *E. Barankin* and *R. Dorfman* (1955), *E. M. L. Beale* (1955), *M. Frank* and *P. Wolfe* (1956), *H. Markowitz* (1956), *C. Hildreth* (1957), *H. Houthakker* (1957) and *P. Wolfe* (1959). Additional papers have appeared since 1959. Most of the techniques presented for solving quadratic programming problems were a confluence of the simplex computational approach and the theory of *Kuhn and Tucker*. Basically, these methods relied on the fact that first partial derivatives of quadratic functions are linear.

It is natural that quadratic programming received so much attention from theorists. Mathematically, it is the natural first extension beyond the realm of linear programming. And, it also has the mathematical advantage of being solvable in a finite number of steps. In practice, it has been used extensively because many practical problems can be put in the form of a nonlinear quadratic program and linear constraints.

1.3 Difficulties caused by nonlinearity

Various types of mathematical programs are distinguished according to the nature of functions to be optimized. The simplest type in which all the functions are linear, is called a *linear* *gram.* All other mathematical programs may be referred to collectively as nonlinear programs. It is generally much easier to solve a linear program than a nonlinear program. To see this, first consider the following linear program:

$$Max \quad z = x_1 + 4x_2$$

subject to

$$x_1 + x_2 \leq 8$$
$$-3x_1 + 2x_2 \leq 6$$
$$\text{and} \qquad x_1, x_2 \geq 0.$$

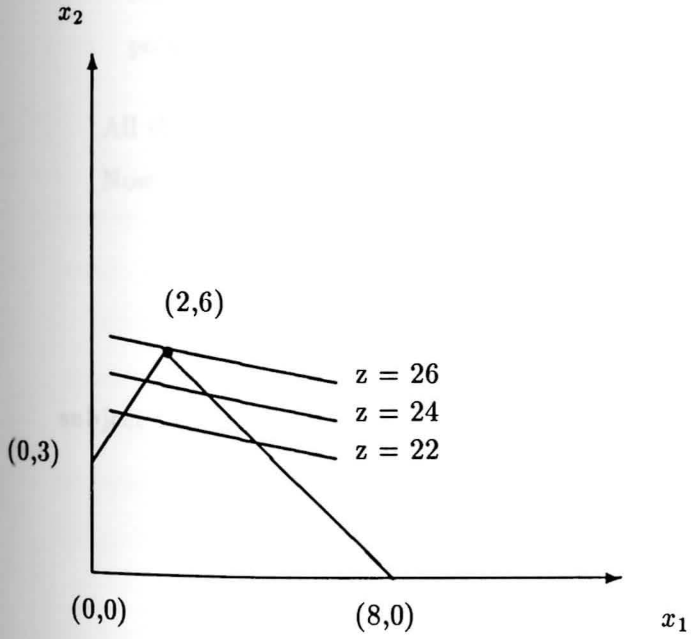


Figure 1.

The feasible region is shown in Figure 1. To solve this problem via the simplex method , we start at the origin, where z = 0, and test for optimality by computing the reduced costs. We choose the x_2 -direction, in which the rate of increase of z is greater. By pivoting, we "jump" from one corner point to the other until the optimal point is reached.

The ease with which linear programming problems or LPP's can be solved is due to the following 3 important factors:

1) Given any bounded LPP, at least one corner or extreme point of the feasible region must be optimal. Therefore, to solve any linear program it is only necessary to search over a finite number of feasible solutions, i.e., those corresponding to the extreme points.

2) The various extreme-point solutions can be obtained easily and directly by means of linear algebraic transformations.

3) When an extreme point is found such that no movement away from it in any feasible direction can improve the value of the objective function, then that extreme point must be the optimal solution to the problem.

All the above characteristics of LPP are fully exploited by the Simplex Method.

Now consider a nonlinear program:

$$Min \quad z = [(x_1 - 8)^2 + (x_2 - 4)^2]^{\frac{1}{2}}$$

subject to

$$x_1 + x_2 \leq 8$$
$$-3x_1 + 2x_2 \leq 6$$
$$and \qquad x_1, x_2 \geq 0.$$

When this problem is solved graphically (Figure 2) it is seen that the optimum does not lie at a corner point; thus this problem can not be solved by a Simplex-type algorithm.

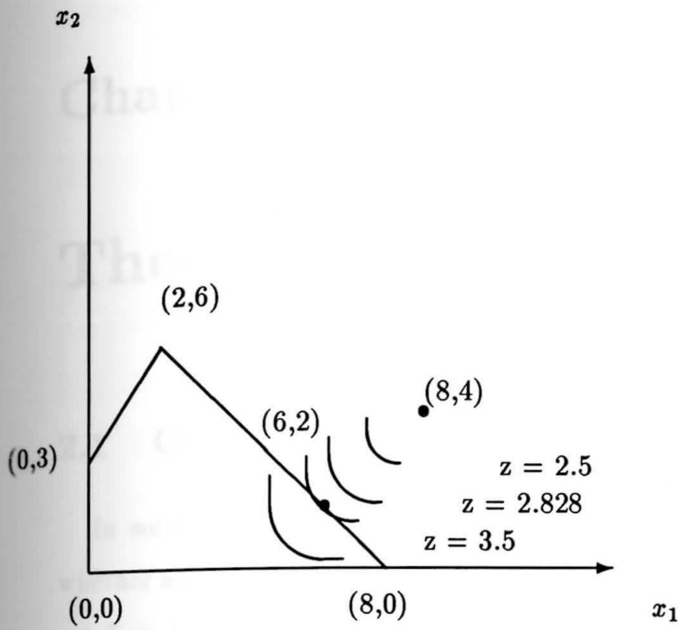


Figure 2.

Furthermore, an algorithm that would search over all the boundary points of any given feasible region cannot be used to solve every nonlinear program. Such an algorithm would be unable to locate the point in the interior of the feasible set.

The existence of local optima that might not be optimal overall is possible in nonlinear programs. In general, the methods in nonlinear programming, including quadratic programming, are capable of finding local optima only. To solve a problem that has several local optima, it is necessary to do extra work to find them all. This limitation causes no additional difficulties in the problems where no local optima other than overall optima exist.

Chapter 2

Theoretical Perspective

2.1 Convex and concave functions

In solving a mathematical programming problem, it is frequently necessary to determine whether a given function f is convex or concave. *The function is said to be convex over a convex set X in E^n if for any two points x_1 and x_2 in X and for all λ , $0 \leq \lambda \leq 1$,*

$$f[\lambda x_2 + (1-\lambda)x_1] \leq \lambda f(x_2) + (1-\lambda)f(x_1)$$

and concave if

$$f[\lambda x_2 + (1-\lambda)x_1] \geq \lambda f(x_2) + (1-\lambda)f(x_1).$$

Observe that if f(x) is concave, then -f(x) is convex, and vice versa.

If f(x) is a convex function over E^n, then the set of points S={ x where f(x) $\leq$ b }, where b is any real number, is a convex set. Similarly if f(x) is concave, then T={ x where f(x) $\geq$ b } is a convex set.

Consider the mathematical programming problem

$$Max \quad z = f(x)$$

subject to

$$g_i(x) \qquad (\leq, =, \geq) \quad b_i$$

$$x = (x_1, x_2, \ldots, x_m) \qquad i = 1, 2, \ldots, m.$$

Since the intersection of any two convex sets is a convex set, the feasible region of the above mathematical program is a convex set if the following three sufficient conditions are met:

(a) for all constraints $g_i(x) \leq b_i$, the function $g_i(x)$ is convex;

(b) for all constraints $g_i(x) \geq b_i$, the function $g_i(x)$ is concave;

(c) for all constraints $g_i(x) = b_i$, the function $g_i(x)$ is linear.

It can be very difficult to show that even a simple function like f(x) = e^x is concave or convex by the above definitions. However, there is another method to classify the function based on the second derivative. If it is a function of one variable we determine whether the second derivative is negative or positive over the interval of interest. If a function has two variables, we determine the Hessian Matrix, H, associated with the function $f(x_1, x_2)$:

$$H = \begin{bmatrix} \frac{\delta^2 f}{\delta x_1^2} & \frac{\delta^2 f}{\delta x_1 \delta x_2} \\ \frac{\delta^2 f}{\delta x_2 \delta x_1} & \frac{\delta^2 f}{\delta x_2^2} \end{bmatrix}.$$

The *quadratic form*[1] q(x) = $X^T H X$, where H, the Hessian matrix, is positive semidefinite if and only if the variables can be ordered such that, if h_{ij} are the elements of H, h_{11} is positive and the determinant of H is nonnegative:

$$\begin{vmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{vmatrix} \geq 0.$$

$\boxed{2.1}$

If the above determinant is strictly positive, then the quadratic form is positive definite. Similarly, if the h_{11} is negative and the sign of the above determinant is nonpositive, the quadratic form can be described as positive semidefinite, and negative definite if the determinant is strictly negative.

[1]A quadratic form is any scalar valued function, defined for all X in E^n, that takes the form shown below.

$$q(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} h_{ij} x_i x_j$$

where each h_{ij} is a real number.

If the function f(x) has continuous second order partial derivatives, then it is concave (convex) over some region R in E^n if and only if its Hessian matrix is negative (positive) definite or semidefinite for all x.

2.2 Maxima and Minima

Before we consider solution methods for mathematical optimization problems, we must be careful to specify exactly what an optimum is. Three kinds of optima are defined below. The definitions are described in terms of maxima but analogous definitions and properties exist for minima.

One of the goals in solving a mathematical program is to identify the global or absolute maximum from among the feasible points, if one exists.

A global maximum of the function f(x) over a closed set S occurs at the point x^* in S if and only if $f(x) \leq f(x^*)$ for all points x in S. We then say that x^* maximizes f(x) in S, or that f(x) takes on a global maximum over S at the point x^* .

The definition makes it clear that global maxima may occur at more than one point in S, although the maximum feasible value of f(x) must be unique. If the closed set S is the feasible region of any mathematical programming problem having a bounded, nonempty feasible region, then at least one feasible point must be a global maximum. If the feasible region is unbounded, the maximum may either occur at some specific finite point or may not exist.

There exist in addition to global maxima, two types of local maxima: unconstrained and constrained.

Let f(x) be defined for all x in some set T. An unconstrained local (or relative) maximum of f(x) occurs at x^* in T provided that there exists some $\delta \geq 0$ such that if x is within a distance δ of x^*, then x is in T and $f(x) \leq f(x^*)$.

An unconstrained local maximum can be thought of as the top of any hill in a region containing one, two or several hills. By definition it can only occur in the interior of T, never on the boundary. Let us take a closed interval T, $0 \leq x \leq 1$ on the real line (Figure 3) . The points x=a, x=b and x=c are unconstrained local maxima of f(x), but x = 0 and x = 1 are not.

Let f(x) be defined for all x in some U. A constrained local maximum of f(x) with respect to the set U occurs at x^* in U provided there exists some $\delta \geq 0$ such that, if x is in U and lies *within a distance δ of x^*, then $f(x) \leq f(x^*)$.*

This definition, unlike the one preceding, allows x^* to be a boundary point. We see that any unconstrained maximum is also a constrained maximum, although the reverse is not true. It is also true that any (finite) point that constitutes a global maximum of f(x) over a set S must be a constrained local maximum as well.

The global maximum in Figure 3 occurs at x = 0, which is not an unconstrained global maximum but a constrained local maximum.

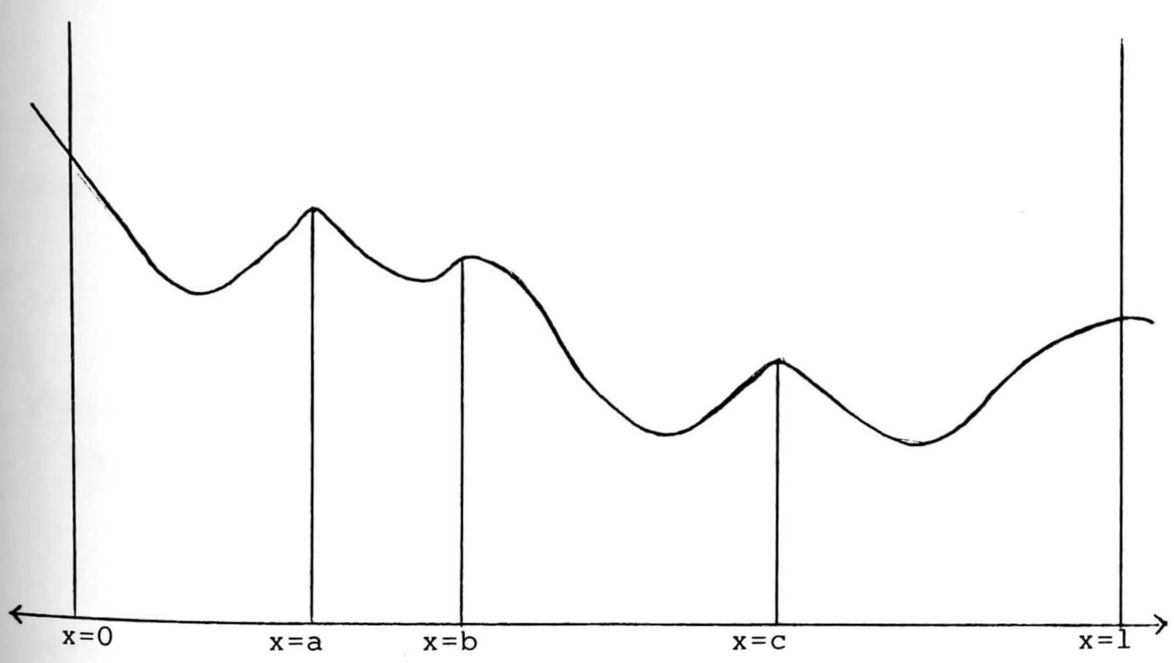


Figure 3

2.3 Quadratic Programming

A quadratic programming problem is a problem that has a quadratic objective function and the constraints that are linear.

Although linear forms are the most widely used in the modeling of mathematical optimization problems, quadratic forms come next. Quadratic programs are similiar to linear programs from an analytical and computational point of view. Because of this, they are easy to handle and many real world problems are approximated by quadratic forms.

For example: The surface area of a circle, cube or other regular figure is proportional to the square of its characteristic linear dimension. The sales revenue of a monopolistic firm that sells x_1 units of some product at a price of x_2 per piece is x_1x_2, which is a quadratic term. In statistics, the variance of a given sample of observations is a quadratic function of the values that constitute the sample. Kinetic energy carried by a rocket and/or an atomic particle is proportional to the square of its velocity and the potential energy of a rigid standing wall or a dam is a quadratic function of its height.

The general quadratic programming problem can be written as

$$Max \quad z = \sum_{j=1}^{n} c_j x_j + \sum_{j=1}^{n}\sum_{k=1}^{n} c_{jk} x_j x_k$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \qquad i = 1, ..., m$$

$$and \quad x_j \geq 0 \qquad j = 1, ..., n$$

where a_{ij}, b_i, c_j and c_{jk} are any real numbers.

The above quadratic program can be written more compactly in matrix form as

$$Max \quad z = C^T X + X^T D X$$

subject to

$$AX \leq b$$

$$and \qquad X \geq 0 \qquad \boxed{2.2}$$

where A is an m x n matrix, D is an n x n matrix, b is an m-component vector, and X and C are n-component vectors. Here, D is a symmetric matrix and the elements of D are defined as

$$d_{ij} = d_{ji} = \frac{c_{ij} + c_{ji}}{2}.$$

$\boxed{2.3}$

To date, no technique that directly finds a global optimum for a quadratic programming problem has been developed except when it is known that any local optimum is also a global optimum. *If the objective function is defined and convex over the closed convex set X in E^n, then any constrained local minimum of z in X is a global minimum over X. If the objective function is concave over the closed convex set X, then any constrained local maximum in X is a global maximum over X.* The objective function above is the sum of a linear form and a quadratic form. A linear form is a concave function. If $X^T DX$ is concave then the objective function will be sum of two concave functions and will be concave. Note that $X^T DX$ will be concave, if the Hessian matrix D is negative definite or negative semidefinite, then any constrained local maximum will be a global maximum in $\boxed{2.2}$.

2.4 Lagrange Multipliers and Kuhn-Tucker Theory

The Lagrange Multiplier was named for the French Mathematician *Joseph Louis Lagrange* (1736-1813). This material, classical in origin, is relatively straightforward. It provides a set of necessary conditions which must be obtained at all local optima in a mathematical program with equality constraints and in unconstrained problems. On the other hand, Kuhn-Tucker theory provides an insight into inequality constraints in a bounded feasible region and develops the computational algorithm for dealing with them.

There is a theoretical similarity between the unconstrained problems and the problems with equality constraints. We can solve both of these problems by identifying all the points at which the partial derivatives are equal to zero. The vanishing of the partial derivatives constitutes a set of necessary conditions which must be obtained at all local optima.

H. Kuhn and *A.W. Tucker* (1951) developed the Kuhn-Tucker necessary conditions which are closely bound up with the classical notion of the gradient vector and constitute the basis for identifying the local optima of a nonlinear constrained problem subject to inequality constraints. The development is based on the Lagrangean method. These conditions are also sufficient under certain limitations.

Kuhn-Tucker conditions for the quadratic program

Consider a quadratic programming problem of the form $\boxed{2.2}$. The problem is written as

$$Max \quad z = C^T X + X^T D X$$

subject to

$$G(X) \quad = \quad \begin{pmatrix} A \\ -I \end{pmatrix} X \; - \; \begin{pmatrix} b \\ 0 \end{pmatrix} \quad \leq 0. \qquad \boxed{2.4}$$

Let

$$\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m)^T \qquad and \qquad U = (\mu_1, \mu_2, \ldots, \mu_m)^T$$

be the Lagrange multipliers (≥ 0) corresponding to the set of constraints $AX - b \leq 0$ and $X \geq 0$ respectively. Let $s^2(\geq 0)$ and $r^2(\geq 0)$ be the nonnegative slack variables of the constraints. Associated with above quadratic program is the Langrangean function of the type:

$$L(X, \lambda, \mu, s, r) = C^T X + X^T D X - \lambda(AX - b + s^2) - U(-X + r^2).$$

The necessary condition for the optimality is that λ be non-negative for maximization and non-positive for minimization problems. The ith Lagrange multiplier value associated with the global optima x^* gives the rate of change of the optimal attainable value of the objective function with respect to a change in g_i. The restrictions on λ must hold as part of the Kuhn-Tucker necessary conditions. The remaining conditions are derived as below:

$$
\begin{array}{lllll}
\frac{\delta L}{\delta X} = & \nabla z - (\lambda^T, U^T)\nabla G(X) & = 0 \longrightarrow \boxed{1} & & \\
\frac{\delta L}{\delta s_i} = & -2\lambda_i s_i & = 0 & i = 1,2,...,m \longrightarrow \boxed{2} \\
\frac{\delta L}{\delta r_i} = & -2\mu_i r_i & = 0 & i = 1,2,...,m \longrightarrow \boxed{3} \\
\frac{\delta L}{\delta \lambda} = & AX - b + s & = 0 \longrightarrow \boxed{4} & \\
\frac{\delta L}{\delta U} = & -X + r & = 0 \longrightarrow \boxed{5} &
\end{array}
$$

The above Kuhn-Tucker conditions as written as follows:

$$
\begin{array}{rclccc}
-2X^T D + \lambda^T A - U^T - C & = & 0 & & & \\
AX + s & = & b & & & \\
\mu_j x_j & = 0 = & \lambda_i s_i & j = 1,2,...,n & and & i = 1,2,...,m \\
\lambda, U, X, s, r & \geq & 0 & & & \boxed{2.5}
\end{array}
$$

Sufficiency of the Kuhn-Tucker conditions:

If objective function is a concave function and $G(X)$ is convex in $\boxed{2.4}$, the region defined by $G(X) \leq 0$ is a convex set. It follows that any local maximum of z in the feasible region must be the optimal solution to the problem. Problems such as this one, in which a concave function is to be maximized or a convex function minimized over a convex set, are known as convex programming problems. For such problems, Kuhn-Tucker conditions are sufficient to determine the global optimum.

2.5 Wolfe's Method

A method of solving quadratic programs was proposed by Phillip Wolfe in 1959. Wolfe's algorithm can be applied directly to any quadratic programming problem of the form $\boxed{2.2}$

The basic approach of the algorithm is to generate, by means of a modified simplex pivoting procedure, a sequence of feasible points that terminates at a solution point x^* where the Kuhn-

Tucker conditions are satisfied. Note that the feasible region of $\boxed{2.2}$, which is bounded entirely by hyperplanes, is a convex set. Therefore, provided that the objective function is concave, i.e., the matrix D is negative definite or semidefinite, the point x^* will be an optimal solution. This follows from the sufficiency of the Kuhn-Tucker conditions for convex programming problems. When D is indefinite or positive (semi) definite, convergence may fail entirely, or if it occurs, the solution obtained may not even be a local optimum.

Wolfe was able to modify the simplex method in a way that enabled it to solve quadratic programs by finding feasible points that satisfy the Kuhn-Tucker conditions (Wayne L. Winston,1991).

We will illustrate Wolfe's method by applying it to the following example problem.

$$Min \;\; z = -x_1 - x_2 + (\frac{1}{2})x_1^2 + x_2^2 - x_1x_2$$

subject to

$$\begin{aligned} x_1 + x_2 &\leq 3 \\ -2x_1 - 3x_2 &\leq -6 \\ and \quad x_1, x_2 &\geq 0 \end{aligned}$$

The above problem can be written in the matrix form $\boxed{2.4}$ as follows:

$$Min \;\; z \;\; = \;\; (-1,-1)\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + (x_1,x_2)\begin{pmatrix} \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & -1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

subject to

$$\begin{pmatrix} 1 & 1 \\ -2 & -3 \\ -1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 3 \\ -6 \\ 0 \\ 0 \end{pmatrix}$$

The objective function is convex,[2] so any point satisfying Kuhn-Tucker conditions will solve this QPP(Quadratic Programming Problem).

The Kuhn-Tucker conditions $\boxed{2.5}$ are now written as:

$$\begin{aligned}
x_1 - 1 - x_2 + \lambda_1 - 2\lambda_2 - \mu_1 &= 0\\
-x_1 - 1 + 2x_2 + \lambda_1 - 3\lambda_2 - \mu_2 &= 0\\
x_1 \quad + x_2 \qquad\qquad\qquad + s_1 &= 3\\
2x_1 \quad + 3x_2 \qquad\qquad\qquad - s_2 &= 6
\end{aligned}$$

and $$\lambda_2 s_2 = 0,\ \lambda_1 s_1 = 0,\ \mu_1 x_1 = 0,\ \mu_2 x_2 = 0$$

$$s_1,\ s_2 \geq 0, \qquad \lambda_1,\ \lambda_2 \leq 0.$$

Except for the four complementary slackness conditions for this QPP, the Kuhn-Tucker conditions are all linear.

To find a point satisfying the Kuhn-Tucker conditions, Wolfe's method applies a modified version of Phase I of the two-phase simplex method to the linear Kuhn-Tucker conditions. An artificial variable is added to each equation from the Kuhn-Tucker conditions that does not have an obvious basic variable. Then the sum of the artificial variables is minimized. To ensure that the complementary slackness conditions are satisfied, Wolfe's method modifies the simplex's choice of the entering variable as follows:

1) Never perform a pivot that would make μ_i from the ith constraint and x_i both basic variables.

2) Never perform a pivot that would make the slack (or surplus) variable for the ith constraint and λ_i both basic variables.

[2]The function is convex if its Hessian matrix is positive definite or semidefinite. From $\boxed{2.1}$ the Hessian matrix

$$\begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}$$

of the given function is positive definite as $h_{11} = 1 \geq 0$ and $|H| = 1 \geq 0$.

Now we solve the following LP:

$$Min \;\; z = a_1 + a_2 + a_3$$

subject to

$$\begin{aligned} x_1 - x_2 + \lambda_1 - 2\lambda_2 - \mu_1 + a_1 &= 1 \\ -x_1 + 2x_2 + \lambda_1 - 3\lambda_2 - \mu_2 + a_2 &= 1 \\ x_1 + x_2 + s_1 \qquad\qquad\qquad &= 3 \\ 2x_1 + 3x_2 - s_2 \qquad\qquad a_3 &= 6 \end{aligned}$$

note that all the variables are nonnegative.

The Initial Tableau for Wolfe's Method;

w	x_1	x_2	λ_1	λ_2	μ_1	μ_2	s_1	s_2	a_1	a_2	a_3	rhs
1	2	4	2	-5	-1	-1	0	-1	0	0	0	8
0	1	-1	1	-2	-1	0	0	0	1	0	0	1
0	-1	$\boxed{2}$	1	-3	0	-1	0	0	0	1	0	1
0	1	1	0	0	0	0	1	0	0	0	0	3
0	2	3	0	0	0	0	0	-1	0	0	1	6

In the above tableau, **2** is the pivot element as x_2 is the entering variable and a_2 is the leaving variable.

The First Tableau for Wolfe's Method:

w	x_1	x_2	λ_1	λ_2	μ_1	μ_2	s_1	s_2	a_1	a_2	a_3	rhs
1	4	0	0	1	-1	1	0	-1	0	-2	0	6
0	$\frac{1}{2}$	0	$\frac{3}{2}$	$-\frac{7}{2}$	-1	$-\frac{1}{2}$	0	0	1	$\frac{1}{2}$	0	$\frac{3}{2}$
0	$-\frac{1}{2}$	1	$\frac{1}{2}$	$-\frac{3}{2}$	0	$-\frac{1}{2}$	0	0	0	$\frac{1}{2}$	0	$\frac{1}{2}$
0	$\frac{3}{2}$	0	$-\frac{1}{2}$	$\frac{3}{2}$	0	$\frac{1}{2}$	1	0	0	$-\frac{1}{2}$	0	$\frac{5}{2}$
0	$\frac{7}{2}$	0	$-\frac{3}{2}$	$\frac{9}{2}$	0	$\frac{3}{2}$	0	-1	0	$-\frac{3}{2}$	1	$\frac{9}{2}$

The Second Tableau for Wolfe's Method:

w	x_1	x_2	λ_1	λ_2	μ_1	μ_2	s_1	s_2	a_1	a_2	a_3	rhs
1	0	0	$\frac{12}{7}$	$-\frac{29}{7}$	-1	$-\frac{5}{7}$	0	$\frac{1}{7}$	0	$-\frac{2}{7}$	$-\frac{8}{7}$	$\frac{6}{7}$
0	0	0	$\frac{12}{7}$	$-\frac{29}{7}$	-1	$-\frac{5}{7}$	0	$\frac{1}{7}$	1	$\frac{5}{7}$	$-\frac{1}{7}$	$\frac{6}{7}$
0	0	1	$\frac{2}{7}$	$-\frac{6}{7}$	0	$-\frac{2}{7}$	0	$-\frac{1}{7}$	0	$\frac{2}{7}$	$\frac{1}{7}$	$\frac{8}{7}$
0	0	0	$\frac{1}{7}$	$-\frac{3}{7}$	0	$-\frac{1}{7}$	1	$\frac{3}{7}$	0	$\frac{1}{7}$	$-\frac{3}{7}$	$\frac{4}{7}$
0	1	0	$-\frac{3}{7}$	$\frac{9}{7}$	0	$\frac{3}{7}$	0	$\frac{3}{7}$	0	$-\frac{3}{7}$	$\frac{2}{7}$	$\frac{9}{7}$

The current basic feasible solution is w $= \frac{6}{7}$, $a_1 = \frac{6}{7}$, $x_2 = \frac{8}{7}$, $s_1 = \frac{4}{7}$, $x_1 = \frac{9}{7}$. The simplex method recommends that λ_1 should enter the basis. However, Wolfe's modification of the simplex method for the entering variable does not allow both λ_1 and s_1 to be basic variables since, the condition $\lambda_1 s_1 = 0$ has to be satisfied. If we let λ_1 enter the basis and let s_1 leave the basis then the condition $\lambda_1 s_1 = 0$ will be satisfied but the minimum ratio rule will be violated. Thus, λ_1 cannot enter the basis. Since s_2 is the only other variable with a positive coefficient in row zero, we enter s_2 into the basis. We perform pivots using the above rules until we reach the optimum solution with z=0.

2.6 Convergence of Wolfe's algorithm

In this section we will show that when D is negative definite in case of maximization problems, Wolfe's algorithm will eventually find a feasible point satisfying the Kuhn-Tucker conditions (Simmons, 1975). Here, we will only demonstrate that such a point actually exists, though, to obtain the result we need to show that when D is negative definite, the objective function is bounded and we show that some finite feasible point must be a global maximum and a constrained local maximum, at which the Kuhn-Tucker conditions necessarily hold.

Consider what happens to the value of the objective function $z = C^TX + X^TDX$ where D is negative definite, as X is displaced from some fixed point x_0 in any specified direction $y_0 \neq 0$. The points generated are of the form $X = x_0 + \theta y_0$ where θ is a nonnegative scalar. The objective value at any point X, expressed as a function of θ , is then

$$Z(\theta) = C^Tx_0 + \theta C^Ty_0 + x_0^TDx_0 + 2\theta x_0^TDy_0 + \theta^2 y_0^TDy_0.$$

The $y_0^TDy_0$ has a negative value. Therefore, as θ increases, the term $\theta^2 y_0^TDy_0 \leq 0$ eventually becomes large enough in magnitude to dominate the sum and cause $Z(\theta)$ to begin decreasing; regardless of the values of c, D, x_0 and y_0 , the term $2\theta x_0^TDy_0$ does not increase at the same rate because of single θ in this term. Therefore we conclude that z cannot become infinitely large, and that there exists at least one point in the feasible region that satisfies the Kuhn-Tucker conditions.

Wolfe's algorithm seeks a feasible solution by starting with an initial set of values, that satisfy some of the constraints. Then by adding nonnegative artificial variables a_j to satisfy the others and using the modified simplex pivoting procedure to minimize $z = \sum_j a_j$, with $x_j\mu_j = 0 = \lambda_j s_i$, a feasible solution is found. The variable chosen to enter the basis at each pivot will have a positive reduced cost. The objective function will therefore decrease or remain the same at each iteration. Assuming that a repeating sequence of degenerate bases is not encountered, the procedure must eventually terminate at a solution having the property that no non-basic variable with a negative reduced cost can be brought into the basis without violating the condition $x_j\mu_j = 0 = \lambda_i s_i$ in $\boxed{2.5}$.

When this point is reached, the value of every artificial variable will be zero, and the Kuhn Tucker conditions will be satisfied. This can be proven by taking the dual of the program and by exploiting some of the joint properties of optimal solutions to primal and dual linear programs.

2.7 Curve fitting and the Principle of Least Squares

One very useful application of quadratic programming arises when statistical data are to be fitted to the mathematical model by the method of least squares.

Let (x_i, y_i), i = 1,...,n be n distinct points with x the independent variable and y the dependent variable. The general problem in curve fitting is to find an analytical expression of the form $y = f(x)$, for the functional relationship of the data.

Fitting of a curve to the set of numerical data is of considerable importance theoretically as well as practically. Theoretically, it is useful in the study of correlation and regression. For example, lines of regression can be regarded as the fitting of linear curves to a bivariate distribution. In practical statistics, it enables us to represent the relationship between two variables by simple algebraic expressions, e.g., polynomials, exponential or logarithmic functions. It may be used to estimate the values of one variable which would correspond to the specified values of the other variable.

Fitting of a Straight line:

Consider the fitting of a straight line

$y = a + bx$

to a set of n points (x_i, y_i), $\quad for \quad i = 1, ..., n.$

The problem is to determine 'a' and 'b' so that the approximating line is the line of 'best fit'.

The term 'best fit' can be interpreted as the deviations of the actual values of y from their estimated values as given by the line of best fit.

Let $P_i(x_i, y_i)$ be any general point in the scatter diagram (Figure 4). We find the distance of this point from our line, y= ax + b.

The error of estimate or residual for y_i is,

$P_iH_i = P_iM - H_iM = y_i - (a + bx_i)$.

According to the principle of least squares, we have to determine a and b so as to minimize

$E = \sum_{i=1}^{n}(P_iH_i)^2 \quad = \sum_{i=1}^{n}(y_i - a - bx_i)^2.$

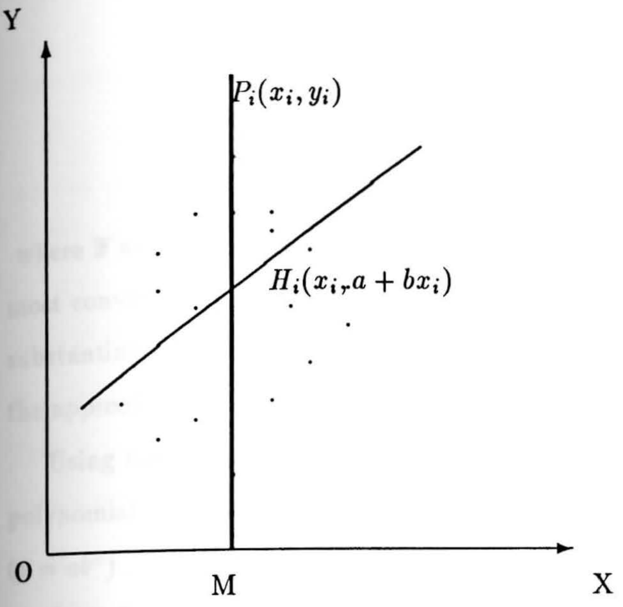


Figure 4

From the calculus the partial derivatives of E *w.r.t* a and b should vanish separately.

$$i.e \quad \frac{\delta E}{\delta a} = 0 = -2\sum_{i=1}^{n}(y_i - a - bx_i)$$

$$=> \sum_{i=1}^{n} y_i = na + b\sum_{i=1}^{n} x_i.$$

$$\frac{\delta E}{\delta b} = 0 = -2\sum_{i=1}^{n}(y_i - a - bx_i)x_i$$

$$=> \sum_{i=1}^{n} x_i y_i = a\sum_{i=1}^{n} x_i + b\sum_{i=1}^{n} x_i^2.$$

The above two equations are known as the normal equations for estimating a and b.

We can calculate $\sum_{i=1}^{n} x_i y_i, \sum_{i=1}^{n} y_i, \sum_{i=1}^{n} x_i, \sum_{i=1}^{n} x_i^2$ from the set of points (x_i, y_i) and the values of a and b can be found as follows:

$$b = \frac{n\sum_{i=1}^{n} x_i y_i - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}$$
$$a = \overline{y} - b\overline{x}$$

where $\overline{x}$ and $\overline{y}$ are the mean values of x_i *and* y_i, i=1, ..., n. The least squares method is the most convenient procedure for determining a line of best fit. This method does, however, put substantially more weight on an outlier but will not allow that point to completely dominate the approximation.

Using the least squares method, we can fit a straight line (y = ax + b) , a K^{th} degree polynomial $(a_0 + a_1x + a_2x^2 + \cdots + a_kx^k)$, power curves $(y = ax^b)$ and exponential curves $(y = ab^x)$.

One of the limitations of the method of curve fitting by the principle of least squares is the choice of mathematical curve to be fitted to the given data. If we plot the data on an arithmetic or semi-logarithmic scale, it often provides adequate basis for selecting the type of curve.

Another problem with the least squares method arises with data that contains some points that are wildly inaccurate when compared to the overall accuracy that is expected. In addition to errors in measurement and round off, there is always the danger of points being recorded incorrectly. These wild points may comprise 15 percent or so of the total (I.Barrodale, 1966) and can prevent a good approximation from being obtained and consequently these points may remain undetected. Theoretically it is seen that the L_1 approximations is better than L_2 or least squares approximations in the presence of wild points.

There is another situation where least squares is not the best method (J. Peter Bloomfield and William. L. Steiger, 1983). Let Z = $(x, y) \in R^{k+1}$ be a random vector whose components obey the linear model

$Y = a_1x_1 + \ldots + a_kx_k + U,$

$Y = < ax > + U.$

Suppose the errors U follow the double exponential law f(t) = $\frac{h}{2}e^{-h|t|}$. It has been shown that L_1 approximations are superior to L_2 in such a case.

2.8 The Constrained Least Squares Problems

Formulation of Our Problem

A quadratic programming problem which will be solved using the information given in the last few sections will now be formulated.

To begin with, assume that we have available as data, T different set of joint observations (x_t, y_t) for the time periods t = 1, ..., T. If the variables x_t and y_t in a bivariate distribution are related, we will find that the points in a scatter diagram will cluster around some curve called the " curve of regression ". If the curve is a straight line, this is called a line of regression and there is said to be linear regression between the variables. To estimate a single equation linear regression, the regression equation is

$$y_t = a + bx_t + u_t, \qquad t = 1, ..., T.$$

The regression model estimates an equation using the least squares technique. That is, we minimize the squares of the residuals,

$$\sum_{t=1}^{T} {u_t}^2 = \sum_{t=1}^{T} (y_t - a - bx_t)^2.$$

There are situations that one may suspect the presence of hidden causal factors which we may not be able to observe directly in the data available to us, as, for example, when underlying economic structure is changing systematically over time due to changes of taste, custom, technology etc. These variables are often difficult to measure directly. We will make an assumption that the demand curve, cost curve, etc., shifts monotonically over time. This assumption can be translated into constraints of the type $a_T \leq a_{T-1} \leq \cdots \leq a_1$, since a_i's determine the magnitude of the shifts which we assume to be monotonic.

S. Thore, 1988, formulated this problem as follows:

$$Min \quad \sum_{t=1}^{T} (y_t - a_t - bx_t)^2$$

subject to

$$a_T \leq a_{T-1} \leq \cdots \leq a_1$$

and $$b, a_i \quad unrestricted, \qquad i = 1, ..., T. \qquad \boxed{2.6}$$

Chapter 3

Solving the Problem

3.1 Mathematical Analysis

We can rewrite the constrained least squares problems as in (2.6) as follows

$$Min \quad z = \sum_{t=1}^{T} [y_t - (a_t + bx_t)]^2$$

subject to

$$\begin{aligned} a_1 &\geq a_2 \\ a_2 &\geq a_3 \\ &\vdots \\ a_{T-1} &\geq a_T \end{aligned}$$

and $\quad b, a_1, ..., a_T \quad \textit{unrestricted.} \qquad \boxed{3.1}$

We write the above quadratic program in matrix form as:

$$Min \quad z = \sum_{t=1}^{T}[y_t^2 - C_tX_t + X_t^T D_t X_t]$$

subject to

$$\begin{pmatrix} A \\ -I \end{pmatrix} X \leq 0$$

$$X \geq 0$$

Where $\quad C_t = (2y_t, 2y_tx_t,)$

$$X_t = \begin{pmatrix} a_t \\ b \end{pmatrix} \qquad X = \begin{pmatrix} a_1 \\ \vdots \\ a_T \\ b \end{pmatrix}.$$

$$and \qquad D_t = \begin{pmatrix} 1 & x_t \\ x_t & x_t^2 \end{pmatrix} \qquad for \quad t = 1, 2, ..., T.$$

D_t is the symmetrix hessian matrix and the determinant $|D_t| = 0$. Therefore, *the quadratic form $X_t^T D_t X_t$ is positive semidefinite*[1]. The objective function is the sum of convex functions and hence convex. Here we minimize the convex function over the convex set so the Kuhn Tucker conditions are sufficient to find the optimal solution. We get an optimal solution which satisfies the Kuhn-Tucker conditions and is a global minimum, not necessarily unique.

We rewrite the problem $\boxed{3.1}$ as

$$Min \quad z = \sum_{t=1}^{T}(y_t - a_t - bx_t)^2$$

[1]Any positive semidefinite quadratic form is a convex function over all of E^n.

subject to

$$\begin{aligned} -a_1 + a_2 &\le 0 \\ -a_2 + a_3 &\le 0 \\ &\vdots \\ -a_{T-1} + a_T &\le 0 \end{aligned}$$

and $\qquad b, a_1, ..., a_T$ unrestricted. $\qquad \boxed{3.2}$

We start the solution process by writing the Lagrangian function of $\boxed{3.2}$

$$L(a_1, a_2, \ldots, a_t, b, \lambda, e_t) = \sum_{t=1}^{T} (y_t - a_t - bx_t)^2 + \sum_{t=1}^{T-1} \lambda_t(-a_t + a_{t+1} + e_t^2)$$

where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_{T-1})$ are the Lagrangean Multipliers and e_t, $t = 1, 2, ..., T-1$, are the nonnegative slack variables.

We now write the *Kuhn-Tucker* conditions as follows,

$$\begin{aligned}
\frac{\delta L}{\delta a_1} &= -2(y_1 - a_1 - bx_1) - \lambda_1 &= 0 \\
\frac{\delta L}{\delta a_2} &= -2(y_2 - a_2 - bx_2) + \lambda_1 - \lambda_2 &= 0 \\
&\cdot \\
&\cdot \\
&\cdot \\
\frac{\delta L}{\delta a_T} &= -2(y_T - a_T - bx_T) \ \ldots \ + \lambda_{T-1} &= 0 \\
\frac{\delta L}{\delta b} &= -2\textstyle\sum_{t=1}^{T}(y_t - a_t - bx_t) &= 0 \\
\frac{\delta L}{\delta \lambda_1} &= -a_1 + a_2 + e_1 &= 0 \\
\frac{\delta L}{\delta \lambda_2} &= -a_2 + a_3 + e_2 &= 0 \\
&\vdots \\
\frac{\delta L}{\delta \lambda_{T-1}} &= -a_{T-1} + a_T + e_{T-1} &= 0
\end{aligned}$$

$$\lambda_1 e_1 = 0, \quad \lambda_2 e_2 = 0, \quad \ldots \quad, \lambda_{T-1} e_{T-1} = 0$$

$$and \qquad \lambda_i \geq 0, \qquad i = 1, \ldots, T-1.$$

$\boxed{3.3}$

We form the matrix from the above equations :

$$\left[\begin{array}{ccccccccccccc|c}
b & a_1 & a_2 & \cdots & a_T & \lambda_1 & \lambda_2 & \cdots & \lambda_{T-1} & e_1 & e_2 & \cdots & e_{T-1} & r.h.s \\
2x_1 & 2 & 0 & \cdots & 0 & -1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 2y_1 \\
2x_2 & 0 & 2 & \cdots & 0 & 1 & -1 & \cdots & 0 & 0 & 0 & \cdots & 0 & 2y_2 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
2x_T & 0 & 0 & \cdots & 2 & 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 & 2y_T \\
2\sum_{t=1}^T x_t & 2 & 2 & \cdots & 2 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 2\sum_{t=1}^T y_t \\
0 & -1 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 0 \\
0 & 0 & -1 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 1 & 0
\end{array}\right]$$

Matrix A.

The above matrix is of size **(2T) x (3T-1)** and **(2T) x (5T-1)** with artificial variables added to get the starting basis. This matrix is very large and sparse. Although this problem can be solved using some QPP method, we will reduce the size of the above problem so that it is easier and efficient to the solve the same problem. To do this we go through the following steps:

From $\boxed{3.3}$ we now know that

$$\begin{aligned}
a_2 &= \qquad a_1 - e_1 \\
a_3 &= \quad a_2 - e_2 = a_1 - e_1 - e_2 \\
&\vdots \\
a_T &= \; a_{T-1} - e_{T-1} = a_1 - \textstyle\sum_{t=1}^{T-1} e_t.
\end{aligned}$$

Since quadratic programming problem solving methods like Wolfe's method deal only with the positive variables, the unrestricted variables a_1 and b are substituted as follows

$a_1 = a_1^{'} - a_1^{''}$, where $a^{'} \geq 0 \quad a^{''} \geq 0$,

$b = b^{'} - b^{''}$, where $b^{'} \geq 0 \quad b^{''} \geq 0$.

Also, substituting the new values of $a_t, \quad t = 1, 2, \ldots, T$. We rewrite $\boxed{3.3}$ as

$$\begin{aligned}
2a_1^{'} - 2a_1^{''} + 2b^{'}x_1 - 2b^{''}x_1 - \lambda_1 &= 2y_1 \\
2a_1^{'} - 2a_1^{''} + 2b^{'}x_1 - 2b^{''}x_1 + \lambda_1 - \lambda_2 - 2e_1 &= 2y_2 \\
&\vdots \\
2a_1^{'} - 2a_1^{''} + 2b^{'}x_T - 2b^{''}x_T + \lambda_{T-1} - 2\sum_{t=1}^{T-1} e_t &= 2y_T \\
2(a_1^{'} + a_1^{''})\sum_{t=1}^{T} x_t - 2(b^{'} + b^{''})\sum_{t=1}^{T} x_t^2 - 2\sum_{t=1}^{T-1}(e_t \sum_{t'=t+1}^{T} x_{t'}) &= 2\sum_{t=1}^{T} x_t y_t \\
\lambda_1 e_1 = 0, \quad \lambda_2 e_2 = 0 \quad \ldots \quad \lambda_{T-1} e_{T-1} = 0
\end{aligned}$$

and all variables positive. $\boxed{3.4}$

We write $\boxed{3.4}$ in the matrix form as below

$$\left|\begin{array}{ccccccccccc|c}
a_1^{'} & a_1^{''} & b^{'} & b^{''} & \lambda_1 & \lambda_2 & \cdots & \lambda_{T-1} & e_1 & \cdots & e_{T-1} & r \\
. & . & . & . & 0 & 0 & \cdots & 0 & . & \cdots & -(2+2x_1) & \\
2 & -2 & 2x_1 & -2x_1 & -1 & 0 & \cdots & 0 & 0 & \cdots & 0 & 2 \\
2 & -2 & 2x_2 & -2x_2 & 1 & -1 & \cdots & 0 & -2 & \cdots & 0 & 2 \\
2 & -2 & 2x_3 & -2x_3 & 0 & 1 & -1 & .. & 0 & \cdots & 0 & 2 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \\
2 & -2 & 2x_T & -2x_T & 0 & 0 & \cdots & 1 & -2 & \cdots & -2 & 2 \\
2\sum_{t=1}^{T} x_t & -2\sum_{t=1}^{T} x_t & 2\sum_{t=1}^{T} x_t^2 & -2\sum_{t=1}^{T} x_t^2 & 0 & 0 & \cdots & 0 & -2\sum_{t=2}^{T} x_t & \cdots & -2x_1 & 2\sum_{t=1}^{T}
\end{array}\right|$$

Matrix B.

The size of the above matrix is **(T+1) x (2T+2)**. Since there is no ready-made basis, we add artificial variables to the set of equations $\boxed{3.4}$. The size of the matrix after adding the artificial variables will be **(T+1) x (3T+3)**. There is a method known as Tucker's condensed tableau method which will further reduce the size of the matrix. We will discuss this method of reduction process with the help of an example below.

Wolfe's method of solving a QPP is one of the best known. It has the great advantage that it reduces the task of solving a QPP to perform simplex type pivots. Also, all the variables are now non-negative, which is the requirement of the Wolfe's method. We solve a small problem with the use of size-reduction methods and Wolfe's algorithm :

Example for the above approach for the given set of points:

Consider the set of 4 points (x_i, y_i), given below:

(1,6)

(2,19)

(3,12)

(4,15).

We get the following equations from $\boxed{3.4}$:

$$\begin{aligned}
2a_1^{'} - 2a_1^{''} + 2b^{'} - 2b^{''} - \lambda_1 &= 12 \\
2a_1^{'} - 2a_1^{''} + 4b^{'} - 4b^{''} + \lambda_1 - \lambda_2 - 2e_1 &= 38 \\
2a_1^{'} - 2a_1^{''} + 6b^{'} - 6b^{''} + \lambda_2 - \lambda_3 - 2e_1 &= 24 \\
2a_1^{'} - 2a_1^{''} + 8b^{'} - 8b^{''} + \lambda_3 - 2\sum_{t=1}^{3} e_t &= 30 \\
10a_1^{'} - 10a_1^{''} + 30b^{'} - 30b^{''} - 9e_1 - 7e_2 - 4e_3 &= 140 \\
\lambda_1 e_1 = 0, \quad \lambda_2 e_2 = 0 \quad \lambda_3 e_3 = 0 &
\end{aligned}$$

$$\textit{and} \qquad e_1,\ e_2,\ e_3,\ \lambda_1,\ \lambda_2,\ \lambda_3,\ a_1^{'},\ a_1^{''},\ b^{'},\ b^{''} \geq 0.$$

Since there is no obvious basis in the above set of equations we add artificial variables a_1, ..., a_5 to the above equations respectively.

The tableau is given below:

$$\begin{array}{c|cccccccccccccc|c}
var\ in & a' & a'' & b' & b'' & \lambda_1 & \lambda_2 & \lambda_3 & e_1 & e_2 & e_3 & a_1 & a_2 & \cdots & a_5 & r.h.s \\
basis & 18 & -18 & 50 & -50 & 0 & 0 & 0 & -15 & -11 & -6 & 0 & 0 & \cdots & 0 & 244 \\
\hline
a_1 & 2 & -2 & 2 & -2 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots & 0 & 12 \\
a_2 & 2 & -2 & 4 & -4 & 1 & -1 & 0 & -2 & 0 & 0 & 0 & 1 & \cdots & 0 & 38 \\
a_3 & 2 & -2 & 6 & -6 & 0 & 1 & -1 & -2 & -2 & 0 & 0 & 0 & \cdots & 0 & 24 \\
a_4 & 2 & -2 & 8 & -8 & 0 & 0 & 1 & -2 & -2 & -2 & 0 & 0 & \cdots & 0 & 30 \\
a_5 & 10 & -10 & 30 & -30 & 0 & 0 & 0 & -9 & -7 & -4 & 0 & 0 & \cdots & 1 & 140
\end{array}$$

We further try to reduce this problem by a method known as Tucker's condensed tableau method. In this method we always have an assumed basis where the basic columns (shown in the left side of the matrix) do not need to be stored in computer memory. The matrix we now need is formed as follows:

$$\begin{array}{c|cccccccccc|c}
var\ in & a' & a'' & b' & b'' & \lambda_1 & \lambda_2 & \lambda_3 & e_1 & e_2 & e_3 & r.h.s \\
basis & 18 & -18 & 50 & -50 & 0 & 0 & 0 & -15 & -11 & -6 & 244 \\
\hline
a_1 & 2 & -2 & 2 & -2 & -1 & 0 & 0 & 0 & 0 & 0 & 12 \\
a_2 & 2 & -2 & 4 & -4 & 1 & -1 & 0 & -2 & 0 & 0 & 38 \\
a_3 & 2 & -2 & 6 & -6 & 0 & 1 & -1 & -2 & -2 & 0 & 24 \\
a_4 & 2 & -2 & \boxed{8} & -8 & 0 & 0 & 1 & -2 & -2 & -2 & 30 \\
a_5 & 10 & -10 & 30 & -30 & 0 & 0 & 0 & -9 & -7 & -4 & 140
\end{array}$$

We use Wolfe's method and Tucker's condensed tableau method to get the next tableau. In Tucker's updation method, the column corresponding to the entering variable is updated and its elements get the value of the column corresponding to the leaving variable. We see that in this method we always have the assumed basis which need not be saved in memory. This helps us reduce the size of the matrix to be solved and the space needed to store the matrix.

Tucker's Condensed Tableau Method and Wolfe's algorithm

1) The column corresponding to b' is most positive; therefore, it enters the basis and the variable in row 4, i.e., artificial variable a_4, leaves the basis obeying minimum ratio rule. Hence the pivot element is at (4,2).

2) The column corresponding to b' is updated as follows:
The pivot element **8** is saved and the elements in the column corresponding to b' are divided by the negative of the pivot element except the element in place of the pivot element(-8), which is made the reciprocal of the existing one. The rest of the elements in the matrix are changed according to the usual method. The tableau achieved is the first tableau listed below:

3) By Wolfe's method e_i's and λ_i's have to obey the rule:
$\lambda_i e_i = 0$, $i = 1, \ldots, 3$
That is, if e_1 is in the basis then λ_1 cannot enter the basis.

At each step the identity matrix corresponding to the basis is not saved and the column corresponding to the entering variable is updated by the method described above. Also the rule that $\lambda_i e_i = 0$, $i = 1, \ldots, 3$ is satisfied.

The First tableau

var in	a'	a''	a_4	b''	λ_1	λ_2	λ_3	e_1	e_2	e_3	*r.h.s*
basis	5.5	−5.5	−6.25	0	0	0	−6.25	−2.5	1.5	6.5	56.5
a_1	1.5	−1.5	−0.25	0	−1	0	−0.25	0.5	0.5	0.5	4.5
a_2	1	−1	0.5	0	1	−1	−0.5	−1	1	1	23
a_3	0.5	−0.5	−0.75	0	0	1	−1.75	−0.5	−0.5	$\boxed{1.5}$	1.5
b'	0.25	−0.25	0.125	−1	0	0	0.125	−0.25	−0.25	−0.25	3.75
a_5	2.5	−2.5	−3.75	0	0	0	−3.75	−1.5	0.5	3.5	27.5

The Final tableau

$$\begin{array}{c|cccccccccc|c}
var\ in & a' & a_5 & a_4 & b'' & a_2 & \lambda_2 & \lambda_3 & e_1 & a_1 & a_3 & r.h.s \\
basis & 0 & -1 & -1 & 0 & -1 & 0 & 0 & 0 & -1 & -1 & 0 \\
\hline
e_2 & 0 & 3 & -5 & 0 & -2 & -3 & -1 & -1 & -1 & -5 & 20 \\
\lambda_1 & 0 & -2 & 4 & 0 & 2 & 1 & 1 & 0 & 1 & 3 & 0 \\
e_3 & 0 & 2 & -5 & 0 & -1.5 & -1 & -2 & -1 & -1.5 & -2.5 & 10 \\
b' & 0 & 2 & -4 & -1 & -1.5 & -1.5 & -1 & -1 & -1.5 & -3 & 13 \\
a'' & -1 & 3 & -6 & 0 & -2.5 & -2 & -1.5 & -1 & -2.5 & -4.5 & 7
\end{array}$$

There were 5 pivots performed in the above example:

(4,2), (3,9), (1,8), (2,4), (5,1).

The user time taken to solve this problem was 00hr00m02s05.

The first row of the final tableau tells us of the variables which are non-basic and the first column tells of all the variables that are in the basis.

The optimal solution for the given data is:

$$a' = 0, \quad a'' = 7, \quad b' = 13, \quad b'' = 0, \quad \lambda_1 = 0 = \lambda_2 = \lambda_3, \quad e_1 = 0, \quad e_2 = 20, \quad e_3 = 10.$$

Then

$b = b' - b'' = 13$

$a_1 = a' - a'' = -7$

$a_2 = a_1 - e_1 = -7$

$a_3 = a_2 - e_2 = -27$

$a_4 = a_3 - e_3 = -36.$

The equations are as follows:

$y_1 = -7 + 13x_1$

$y_2 = -7 + 13x_2$

$y_3 = -27 + 13x_3$

$y_4 = -37 + 13x_4.$

In Figure 5 the data points are circled and we see that instead of one line as in the conventional least squares method, we get three lines satisfying these points. The monotonic shifts are seen, these shifts represent the presence of hidden causal factor in the data.

We see that the matrix we solved is of size 5 x 10 in the above example. If we had added the artificial variables, the size of the matrix would have been 10 x 10. Thus Tucker's updation method helps save space.

If there are **T** data points:

$\boxed{1}$. Matrix **A**(with artificial variables): (2T) x (5T-1)

$\boxed{2}$. Matrix **A** with the use of Tucker's condensed tableau method: (2T) x (3T-1)

$\boxed{3}$. Matrix **B**(with artificial variables): (T+1) x (3T+3)

$\boxed{4}$. Matrix **B** with the use of Tucker's condensed tableau method : (T+1) x (2T+2).

Hence, we see that the size of the problem is approximately reduced by one-fifth the size of original problem $\boxed{4}$.

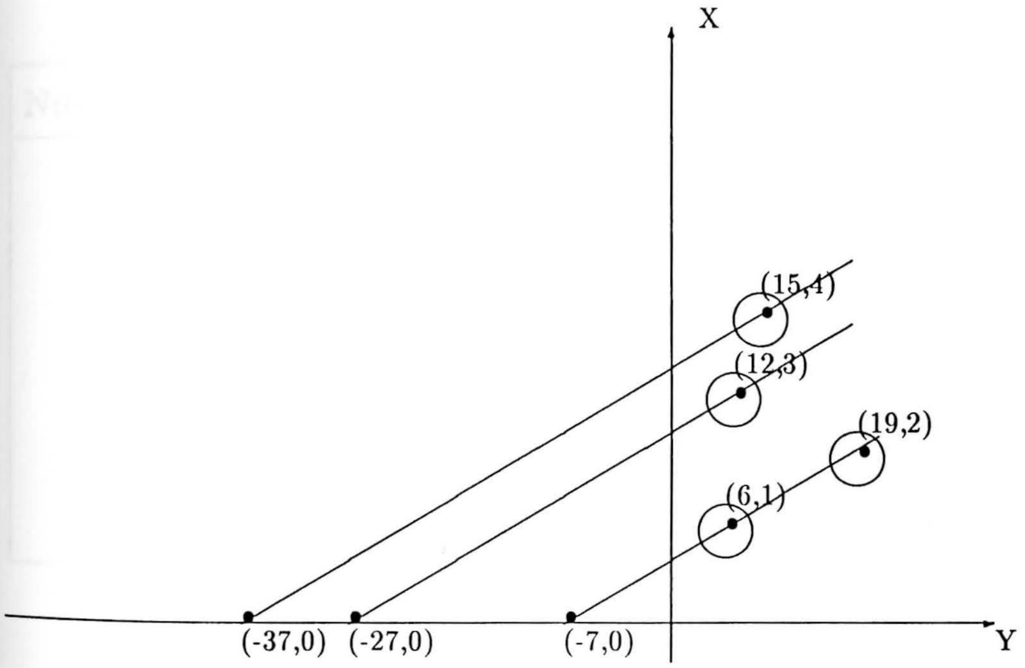


Figure 5.

3.2 Computer Implementation of the Method

The computer programs listed in the Appendix give the equations of the lines satisfying the data points. The program1 reads the input file which is written in the format (x_t, y_t) and creates the matrix of the form **Matrix B**. The program2 performs the pivots on this matrix according to the Wolfe's method of solving a quadratic programming problem. It makes sure that $\lambda_i e_i = 0$, i.e., the complementary slackness conditions are maintained throughout. Other than these conditions, the rest of the program follows the Simplex Method. It also updates the columns by Tucker's condensed tableau method. The solution has been found when the value of the objective function, z, becomes zero. The results are given in the form of equations.

These programs were written in 'C' on the Encore Multimax. The memory of this system is 4 to 12 Megabytes. This memory allows a sufficiently large matrix to be solved. When we run the programs with different sets of data points, we can see how the user time spent with the computer varies with the computer varies with the number of data points. These are shown in a tabular form below:

Number of points	Avg time taken to solve	Avg no of pivots
5	0hr.00m.03s.00	6
10	0hr.00m.16s.75	12
20	0hr.01m.02s.88	21
30	0hr.04m.30s.00	35
50	0hr.20m.00s.00	58
100	1hr.50m.00s.00	105

Table 1.

The results tabulated above are used to plot the following graph (Number of points -vs- Avg time taken to solve) If there are more points, the matrix created is larger and the time taken to solve a larger matrix is more. Observe in figure 6 that for small number of points, the

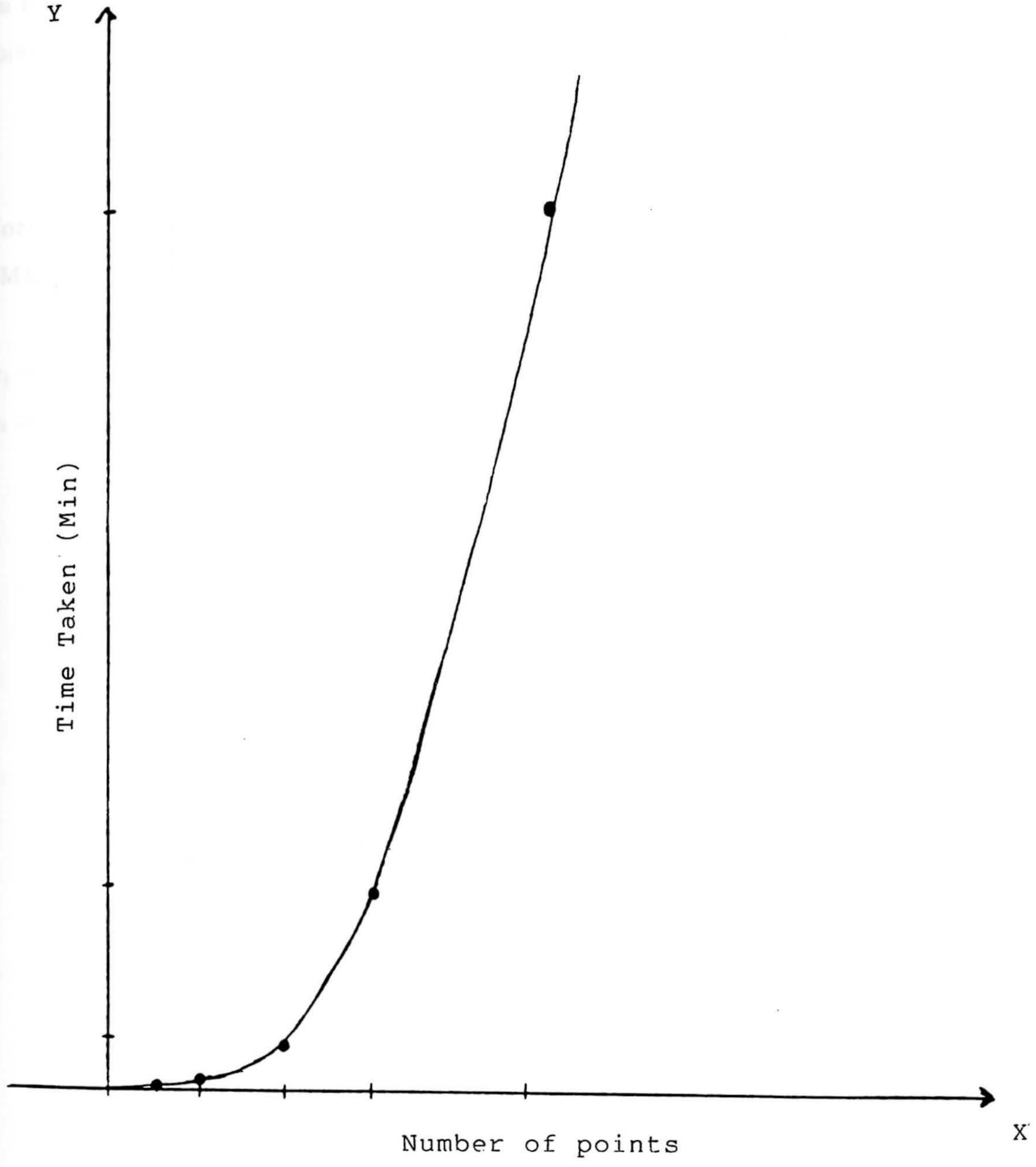


Figure 6

time taken by the computer to solve the problem is very small as compared to the time taken to solve bigger problems. Also observe that the number of pivots increase as the number of points increase.

3.3 Demand for Cigarettes in the U.S (AN EXAMPLE)

Consider the following cigarette data (Harris, 1986, Massachusetts Institute of Technology and Massachusetts General Hospital)

Let

x_t = price of cigarettes per pack, for the years 1964 to 1986,

y_t = Consumption of cigarettes, in packs per day.

year	x_t	y_t
1964	0.996	0.575
1965	1.031	0.583
1966	1.046	0.587
1967	1.064	0.586
1968	1.093	0.573
1969	1.099	0.547
1970	1.136	0.546
1971	1.103	0.553
1972	1.109	0.554
1973	1.051	0.568
1974	1.000	0.567
1975	0.971	0.565
1976	0.959	0.561
1977	0.943	0.555
1978	0.918	0.543
1979	0.874	0.529
1980	0.830	0.527
1981	0.807	0.525
1982	0.851	0.512

1983	0.996	0.478
1984	1.025	0.472
1985	1.047	0.462
1986	1.099	0.449

We see from the data that the price of cigarettes increased until 1973, From 1973 until 1982 it decreased. After that, it increased again. We want to determine the relationship between the cost of cigarettes and consumption of cigarettes from the data available to us. From the data we also see that the consumption declined throughout. Estimating the line of "best fit " using the conventional regression, we get

$y_t = 0.494 + 0.045x_t + u_t$, $R^2 = 0.011.$

This regression is quite meaningless. If we interpret this equation, we see that the positive value of the b-coefficient means that lower price of cigarettes would discourage the consumer from smoking. Therefore, the conventional least squares method seems unappropriate in this example. We suspect that there is some hidden intervening causal factor in the data. It is obvious that the consumption of cigarettes is not totally dependent on price of cigarettes but some of the other factors involved may be difficult to measure. Awareness of the harmful effects of cigarettes among general public is one such factor. In S.Thore's model there is no assumption of a linear trend over time except for the monotonic shifts of the demand over time. The way our problem is formulated, the data itself will dictate the magnitude of the shifts each year. These shifts may occur at a few discrete points of time, or in leaps and bounds, with intermittent periods of change. S.Thore formulated the problem as follows:

$$Min \sum_{t=1964}^{1986} (y_t - a_t - bx_t)^2$$

subject to

$$a_{1964} \geq a_{1965} \geq \cdots \geq a_{1986}$$

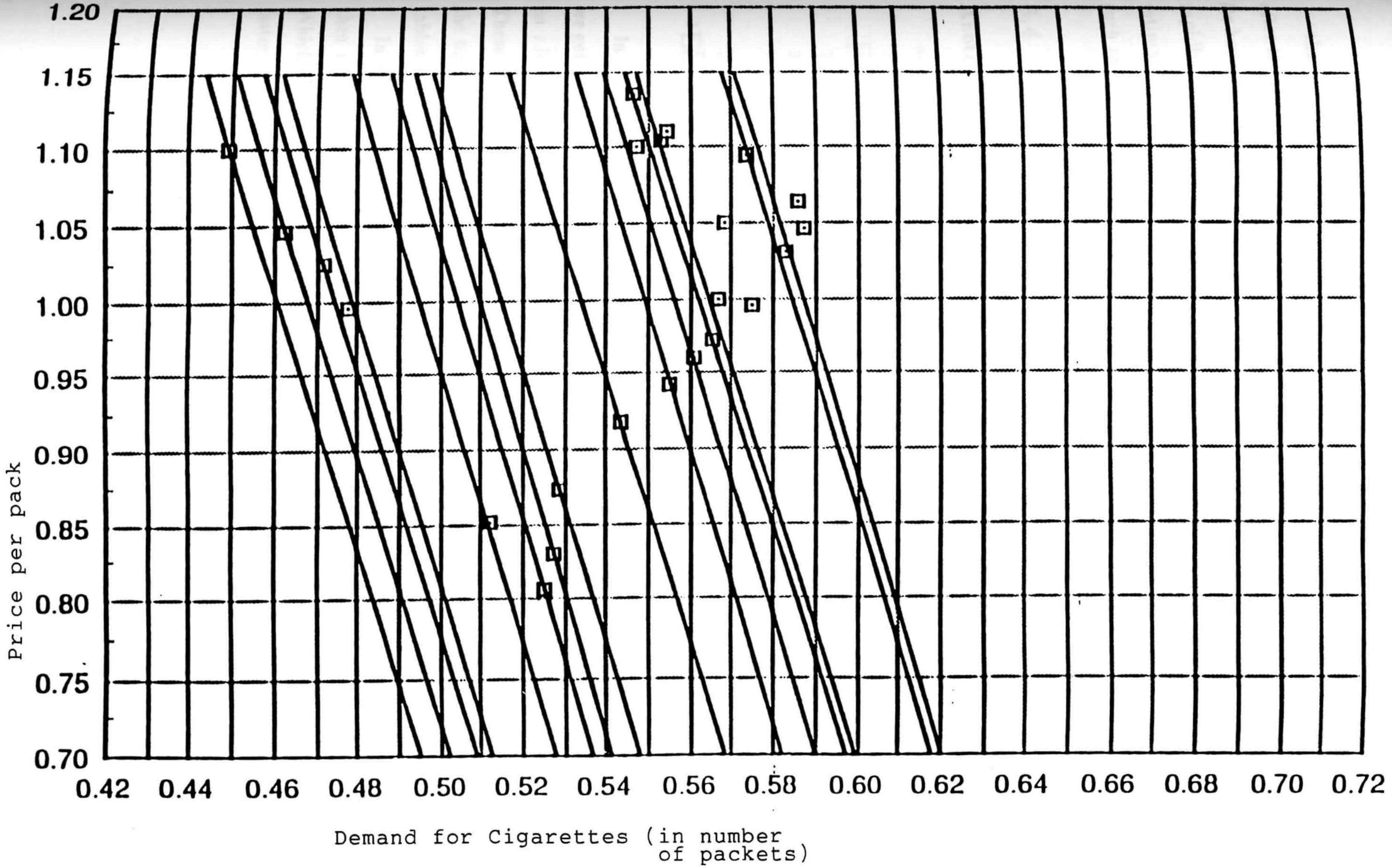
$$and \quad b, a_{1964}, \ ..., \ a_{1986} \quad unrestricted.$$

We use the program 1 and program 2 in the Appendix and get the following results:

1. y_t = 0.699655 -0.113033 x_t
2. y_t = 0.699655 -0.113033 x_t
3. y_t = 0.699655 -0.113033 x_t
4. y_t = 0.699655 -0.113033 x_t
5. y_t = 0.696546 -0.113033 x_t
6. y_t = 0.678249 -0.113033 x_t
7. y_t = 0.678249 -0.113033 x_t
8. y_t = 0.678249 -0.113033 x_t
9. y_t = 0.678249 -0.113033 x_t
10. y_t = 0.678249 -0.113033 x_t
11. y_t = 0.678249 -0.113033 x_t
12. y_t = 0.674757 -0.113033 x_t
13. y_t = 0.669401 -0.113033 x_t
14. y_t = 0.661593 -0.113033 x_t
15. y_t = 0.646767 -0.113033 x_t
16. y_t = 0.627793 -0.113033 x_t
17. y_t = 0.620819 -0.113033 x_t
18. y_t = 0.616220 -0.113033 x_t
19. y_t = 0.608193 -0.113033 x_t
20. y_t = 0.590581 -0.113033 x_t
21. y_t = 0.587856 -0.113033 x_t
22. y_t = 0.580344 -0.113033 x_t
23. y_t = 0.573222 -0.113033 x_t.

Since there were 23 points, the matrix that we created was of size 24 x 48 and the number of pivots performed to get the final solution was 28. The time taken to solve this problem was 0hr02m39s53.

The effect of price on consumption is determined by the co-efficient -0.1130 and the gradual shift of the demand curve is measured by the drop of the constant term. The results indicate



1.20
1.15
1.10
1.05
1.00
0.95
0.90
0.85
0.80
0.75
0.70
Price per pack
0.42
0.44
0.46
0.48
0.50
0.52
0.54
0.56
0.58
0.60
0.62
0.64
0.66
0.68
0.70
0.72
Demand for Cigarettes (in number of packets)

Figure 7

that the demand curve stayed unchanged for the first three years of the data and perhaps after that the public became more aware of the harmful effects of the cigarettes and conscious of its health and the consumption declined. This change was initially quite slow but changed more rapidly during late 1970's and early 1980's. The calculations done did not provide a validated estimation of the demand function for cigarettes in U.S. To find such a function other factors such as income, demographic characteristics of the population, etc., are required.

3.4 Further Research

Another method of Reduction

$$\left|\begin{array}{cccccccccccc|c}
a_1' & a_1'' & b' & b'' & \lambda_1 & \lambda_2 & \cdots & \lambda_{T-1} & e_1 & \cdots & e_{T-1} & & r \\
. & . & . & . & 0 & 0 & \cdots & 0 & . & \cdots & -(2+2x_1) & & \\
\boxed{2} & -2 & 2x_1 & -2x_1 & -1 & 0 & \cdots & 0 & 0 & \cdots & 0 & & 2y \\
2 & -2 & \boxed{2x_2} & -2x_2 & 1 & -1 & \cdots & 0 & -2 & \cdots & 0 & & 2y \\
2 & -2 & 2x_3 & -2x_3 & \boxed{0} & 1 & -1 & .. & -2 & \cdots & 0 & & 2y \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & & \\
2 & -2 & 2x_T & -2x_T & 0 & 0 & \cdots & 1 & -2 & \cdots & -2 & & 2y \\
2\sum_{t=1}^{T} x_t & -2\sum_{t=1}^{T} x_t & 2\sum_{t=1}^{T} x_t^2 & -2\sum_{t=1}^{T} x_t^2 & 0 & 0 & \cdots & \boxed{0} & -2\sum_{t=2}^{T} x_t & \cdots & -2x_1 & & 2\sum_{t=1}^{T}
\end{array}\right.$$

In the above matrix (Matrix B) if we pivot on the cells (1,0), (2,2), (4,3),..., (T+2,T+1), we get the basis we are looking for. Then we can meet the feasibility conditions(all the values on r.h.s should be positive) by using the Dual-Simplex method to 'fix' the right hand side. These pivots should be performed keeping complementary slackness conditions in mind. Again, the basis need not be saved in the computer memory, because we can use Tucker's condensed tableau method.

In the above method for **T** points we will perform **T** pivots for sure to create a basis and then more pivots are performed to meet the feasibility condition by Dual-Simplex method. Which also means more solution time. It is not, however, obvious that which reduction is faster. Future work could answer this question.

3.5 Conclusion

We have developed an efficient and compact method to solve the constrained least squares problem. We saw that conventional least squares method does not provide good results for some data. The need for constrained least squares method was recognized in such cases. We used the theory behind the non-linear programming: Kuhn-Tucker's theory and Wolfe's method of solving the quadratic programming problem to develop a method which now makes it possible to find an unknown factor which shifts the regression hyperplane monotonically downwards/upwards. With the help of a cigarette example our method was illustrated. The programs in the appendix solved this problem. The data itself determined the magnitude of the possible shift each year. The gradual shift of the demand curve is measured by the drop of the constant term. We were able to find out exactly when the scare of smoking set in, when it peaked and what the general trend of demand was. The graph from Figure 7 showed these shifts very clearly.

The size of the problem was reduced considerably using the substitutions and Tucker's updation method. If we solve a big problem using this method, say a 100-points problem, we see that the size of the Matrix A is 200 x 500 as compared to Matrix B which is 100 x 200. Hence the new method can solve the problem in less time and needs less space to solve it.

BIBLIOGRAPHY

1) Barankin, E. W., and R. Dorfman, ' *Towards Quadratic Programming*' ,Office of Naval Research Logistics Projects at Columbia University and University of California, Berkeley, 1955.

2) Barrodale, I.,'L_1 *Approximation and the Analysis of Data*', University of Liverpool, 1968, pp. 51.

3) Beale, E.M.L.,"On minimizing a Convex Function Subject to Linear Inequalities," '*Journal of Royal Statistical Society (B)*', 17, 1955, pp. 173-184.

4) Bloomfield, Peter and William L. Steiger, '*Least Absolute Deviations* (Theory, Applications and Algorithms)'. Birkhauser Boston, Inc., 1983, pp. 39.

5) Boot, John C. G. , '*Quadratic Programming*'. North - Holland Publishing Company, 1964.

6) Frank, M., and P.Wolfe," An algorithm for Quadratic Programming ,"'*Naval Research Logistics Quarterly*, 3, 1956, pp. 95-110, pp. 481-492.

7) Gupta S. C. and V. K. Kapoor, ' *Fundamentals of Mathematical Statistics*'. Sultan Chand & Sons, 1988, pp. 566-567.

8) Hadley. G, '*Nonlinear and Dynamic Programming*'. Addison - Wesley, Publishing Company, 1964.

9) Harris, J. E., Massachusettes Institute of Technology and Massachusetts General Hospital, 1986.

10) Hildreth, C., " A Quadratic Programming Procedure," '*Naval Research Logistics Quarterly*', 14, 1957, pp. 79-85.

11) Houthakker, H., "The Capacity Method of Quadratic Programming,"'*Econometrica*', 28, 1960, pp. 62-87.

12) Kuhn, H. W., and A. W. Tucker, "Nonlinear Programming,"'*Proceedings Second Berkeley Symposium on Mathematical Statistics and Probability*', 1951,

13) Markovitz, H., "The Optimization of a Quadratic Function Subject to Linear Constraints,"'*Naval Research Logistics Quarterly*', 3, 1956, pp. 111-133.

14) Simmons, H. Donal,'*Nonlinear Programming for Operations Research* '. Prentice - Hall, Inc.,Englewood Cliffs, New Jersey, 1975, pp. 13-16, pp.229-230.

15) Taha, A. Hamdy, '*Operations Research (An Introduction)*'. 4th ed. Macmillan Publishing Company. 1987, pp. 793-795.

16) Thore Thore, '*A Constrained Least Squares Method For Estimating The Effects of An Unknown Monotonically Intervening Factor*', working paper, 1988.

17) Winston, L. Wayne, ' *Operations Research Applications and Algorithms* ', 2nd ed ition. PWS - Kent Publishing Company, Boston, 1991, pp. 662-664.

18) Wolfe, P., "The Simplex Method for Quadratic Programming,"'*Econometrica*', 27, 1959, pp. 382-398.

APPENDIX

```
  /*****************/
  /**   PROGRAM1  **/
  /*****************/
/**********************************************************/
/* THIS PROGRAM CREATES THE INITIAL MATRIX FROM THE SET OF GIVEN
DATA POINTS FOR THE CONSTRAINED LEAST SQUARES PROBLEM */
/**********************************************************/
/* THIS FILE READS THE DATA FROM THE FILE 'data.in'.
IT RUNS WITH THE COMMAND 'create '.
**********************************************************/

#include <stdio.h>

main()
{
/* VARIABLE DEFINITIONS : */

        double p,q,r,s,h;
        int i,t,T,tt,tl,j,k,m,n;
        double l,X[500],Y[500],suml[500],totl[500];
        double sum,tot,square,mult,ll,mm;
        char temp_str[512];

        /* FILE DEFINITIONS : */

        FILE  *in_file,*out_file,*outl_file;
        /*******************************/
        /* THIS FILE HAS THE SET OF DATA POINTS IN IT */

        in_file = fopen("data.in","r");


        /* THIS FILE HAS THE FILE IN WHICH THE MATRIX IS
        CREATED */

        out_file = fopen("matrix","w");


        /* THIS FILE HAS THE SIZE OF THE MATRIX TO BE READ
        FOR THE PROGRAM IN WHICH PIVOTS ARE PERFORMED */

        outl_file=fopen("size.file","w");

        /* INITIALIZATION OF CERTAIN VARIABLES */

        sum = 0.0;
        tot = 0.0;
        square = 0.0;
        mult = 0.0;
        l = 0.0;
        p=2.0;
        q=0.0;
        r=1.0;
        h=2.0;
        T=0;


        /* TO KNOW THE NUMBER OF DATA POINTS INTHE DATA FILE
            AND THE VARIABLE T DENOTES IT                    */
```

```
while(fgets(temp_str,512,in_file)) T++;
fclose(in_file);

/*  SIZE OF THE MATRIX TO BE CREATED              */
in_file=fopen("data.in","r");
m=T+1; /* number of rows in the matrix */
n=2*T+2; /* number of columns in the matrix */
fprintf(out1_file,"%d  %d  %d ",m,n,T);

/*********************************************/
/* the set of data points are read in the array X[t] and
Y[t] */

for(t=1; t<=T; ++t){
        fscanf(in_file,"%lf %lf\n",&X[t],&Y[t]);
        sum += X[t];
        tot += Y[t];
        square += X[t]*X[t];
        mult += X[t]*Y[t];
}
for(i=1; i<=T; ++i){
        sum1[i] = 0.0;
}



for(i=2; i<=T; ++i){
        for(j=i; j<=T; ++j){
                sum1[i] += X[j];
                tot1[i] += Y[j];
        }
}
/*********************************
         for creating the first equation
        ********************************/


ll=2*T+sum;
mm=2*sum+square;
fprintf(out_file," %lf %lf ",ll,-ll);
fprintf(out_file," %lf %lf",mm,-mm);
for(i=1; i<=T-1; ++i){
        fprintf(out_file," %lf ",q);
}
for(i=2; i<=T; ++i){
        fprintf(out_file,"%lf ",-p*(T-(i-1))-sum1[i]);
}
fprintf(out_file," %lf ",2*tot+mult);
fprintf(out_file," \n");
/***************************************
        for creating the next T-1 equations
        ***************************************/
for(i=1; i<=T; ++i){


fprintf(out_file,"%lf %lf %lf %lf",p,-p,p*X[i],-p*X[i]);

        for(j=3; j<=i; ++j){
                fprintf(out_file," %lf",q);
                }
                if ( i== 1 )
                        fprintf(out_file," %lf ",-r);
                else if( i== T)
                        fprintf(out_file," %lf ",r);
                else
                        fprintf(out_file," %lf %lf ",r,-r);


                for(j=i+1; j<=T-1; ++j){
                        fprintf(out_file," %lf ",q);
                }

                for(k=1; k<=i-1; ++k){
                        fprintf(out_file,"  %lf ",-h);
                }

                for(k=i; k<=T-1; ++k){
                        fprintf(out_file," %lf ",q);
                }

                fprintf(out_file," %lf \n ",2*Y[i]);
        }

        /********************************
                for creating the last  equation
                ********************************/


fprintf(out_file,"  %lf  %lf %lf %lf",sum,-sum,square,-square);
        for(i=1; i<=T-1; ++i){
                fprintf(out_file,"   %lf ",1);
        }
        for(i=2; i<=T; ++i){
                fprintf(out_file,"   %lf",-(suml[i]));
        }
        fprintf(out_file,"  %lf \n ",mult);


}/* main */
```

```
   /*****************/
   /**   PROGRAM2  **/
   /*****************/
/************************************************************/
/* THIS PROGRAM SOLVES THE CONSTRAINED LEAST SQUARES PROBLEM.
 IT USES THE MATRIX CREATED IN FILE ' matrix ' IN PROGRAM1.c  */
/************************************************************/
/* THIS PROGRAM RUNS WITH THE COMMAND 'pivot'.
   THE RESULTS ARE IN THE FILE NAME ' results '.
************************************************************/
#include <stdio.h>
#define unmark -10

int bubble();
int chec[500];
int chec1[500];

main()
{
        double a[290][500],ratio[250],z[500],zrow[500];
        double r1,r2,save,save1,hold,hold1,hold2,bet;
        double e[250],lambda[20],beta[20],alpha[20],alp[250];
        double diff1,diff2,difference,X[100],Y[100],solution;
        int   pivotno,rat,max,T,i,j,k,w,t,m,n,m1,n1,col;
        int    column[500],ans,cool,MAX,index0,index1,index2;
        int    c1,set,select1,get,flag,coo,save2,save3;
      /***********************************************************/
      /* FILE DEFINITIONS :*/
        FILE   *i_file,*o1_file,*oo_file,*o3_file,*in_file;
        FILE   *o2_file;
/**********************************************************/
        /* THIS FILE HAS THE SIZE OF THE MATRIX TO BE SOLVED
                AND ALSO THE NUMBER OF DATA POINTS */

        o1_file=fopen("size.file","r");

/* READS THE SIZE OF THE MATRIX AND THE NO OF DATA POINTS*/
        fscanf(o1_file," %d %d %d ",&m1,&n1,&T);

/* THIS FILE HAS THE FINAL SOLUTION FROM THE FINAL MATRIX */
        o2_file=fopen("results","a+");

/* THIS FILE HAS THE LATEST MATX AFTER THE PIVOT IS PERFORMED */
        i_file= fopen("matrix", "r+");

/*  TO READ THE MATRIX FROM THE matrix IN AN ARRAY a[][]*/
        for(k=0 ; k<= m1 ; ++k){
                for(t=0 ; t<= n1 ; ++t){
                        fscanf(i_file," %lf  ",&a[k][t]);
                }
        }
        diff1= a[0][n1];
/**********************************************************/
        /*TO INITIALIZE THE ARRAY FOR THE CHECKING*/
        for(i=0; i<500; ++i){
                chec[i]=unmark;
                chec1[i]=unmark;
        }
        pivotno = 0;
        difference = 0.01;
/*******************************************************/

/* THIS WHILE LOOP PERFORMS THE PIVOT TILL Z IS GREATER THAN 0.00001,
 OR WHEN THE VALUE OF Z STOPS IMPROVING MUCH AFTER IT IS CONSIDERABLY
  SMALL */
        while(difference >= 0.00001 ){
                save2=0;
                save3=0;
                set=0;
                pivotno=pivotno + 1;
/* to find the total no of pivots performed*/
                for(i=0; i<nl; ++i){
                        column[i]=i;
                }
/*******************************************************/
/* TO READ THE ROW 0 IN THE ARRAY zrow TO BE SORTED */
                for(i=0; i<=nl; ++i){
                        zrow[i]=a[0][i];
                        z[i]=a[0][i];
                }
                flag=0;
                max=0;
/*******************************************************/
/* THE FUNCTION bubble SORTS THE ROW 0 AND SAVES THE
            COLUMN NOS OF THE SORTED ROW IN THE ARRAY col */
                bubble(zrow,column,nl);
/*******************************************************/
/* THE FOLLOWING MODULE HELPS IN CHOOSING THE COLUMN
   TO PERFORM THE PIVOT ON ACCORDING TO THE WOLFE'S METHOD .*/
/* THE FLAG IS SET WHEN THE COLUMN ON WHICH THE PIVOT IS
   TO BE PERFORMED IS FOUND */
/* THE ARRAY chec[] and checl[] HELPS TO KEEP TRACK OF THE
   COLUMNS WHICH ARE IN THE BASIS AND IT ALSO KEEPS TRACK
   OF WHICH VARIABLE IS IN WHICH ROW */

                while(flag==0){
                        col=column[max];
/* If the value of the column in z-row  is almost zero we
   don't perform the pivot  and set the flag */
                        if(z[col]<=.0000001){
                                set=unmark;
                        }
                        if(col<4){
                                flag=1;
                                for(i=0; i<=nl; ++i){
                                        if(checl[i]==col){
                                                save3=unmark;
                                        }
                                }
                                if(save3 != unmark){
                                        if(set != unmark){
                                                chec[col]=col;
                                        }
                                }
                        }
                        if(col>=4){
                                if(col<=4+T-2){
                                        coo=col+T-1;
                                        max = max + 1;
```

```
                                          if(chec[coo]==unmark){
                                            if(set != unmark){
                                              chec[col]=col;
                                              flag=1;
                                              chec[coo]=100;
                                            }
                                          }
                                  }
                          }
                          if(col>=4+T-1){
                                  if(col<4+2*T-2){
                                          coo=col-T+1;
                                          max = max + 1;
                                    if(chec[coo]==unmark){
                                        if(set != unmark){
                                          chec[col]=col;
                                          flag=1;
                                          chec[coo]=100;
                                    }
                                  }
                          }
                  }
          }/* while*/

/**********************************************************/
/* THE SELECTION OF COLUMN ENDS HERE */
/**********************************************************/
/* NOW WE DECIDE ON THE ROW TO PERFORM THE FINAL PIVOT
   ON THE BASIS OF  MINIMUM RATION RULE , r1 CONTAINS
   THE ELEMENT OF THE LAST COLUMN AND r2  CONTAINS THE
   ELEMENT OF THE SELECTED COLUMN */

                  if(set != unmark){
                          for(k=1; k<=m1; ++k){
                                  r1= a[k][n1];
                                  r2=a[k][col];
                  if(r2 != 0.0){  /* to avoid division by zero */
                                          ratio[k]=r1/r2;
                                  }
                                  else {
                                          ratio[k]=1000000.000;
                                  }
                          }
/**********************************************************/

                          for(i=1; i<=m1; ++i){
              if(ratio[i]<0.0) /* to avoid the negative ratio */
                                          ratio[i]=1000000.00;
                          }
/**********************************************************/

   /* THIS PART FINDS THE MINIMUM OF THE RATIOS IN THE ARRAY ratio[]*/
       /*  THE MINIMUM RATION RULE */
                          save=ratio[1];
                          select1=1;
                          for(i=2; i<=m1;++i)
                          {
                                  if(ratio[i] < save){
                                          save=ratio[i];
                                          select1=i;
                                        }
                                }

/*******************************************************/
/* THIS PART TAKES CARE OF THE UNMARKING OF THE COLUMNS
   IN CASE AFTER SOME PIVOTS, ACCORDING TO WOLFE'S METHOD,
   THE VARIABLE WHICH COULD NOT ENTER AT SOME POINT CAN
   NOW ENTER */

        /* If the column re-enters the basis then we set
                                  flag save3 */
                                m=select1;
                                n=col;
                                for(i=0; i<=n1; ++i){
                                        if(chec1[i]==n){
                                                save3=unmark;
                                                chec[i]=m;
                                        }
                                }
        /**********************************************/

                                if(save3!=unmark){
                                        for(i=0; i<=n1; ++i){
                                                if(chec[i]==m){
                                                   chec[i]=unmark;
                                                   chec1[i]=n;
                                                    if(i>=4){
                                                        if(i<=4+T-2){
                                                         cool=i+T-1;
                                                chec[cool]=unmark;
                                                        }
                                        }

                                                    if(i>4+T-1){
                                                       if(i<=4+2*T-2){
                                                          cool=i-T+1;
                                                     chec[cool]=unmark;
                                                        }
                                                    }
                                    }
                        }
                        chec[n]=m;
                }
        printf(" %d %d \n\n",m,n);
/*******************************************************/

/* TUCKER-UPDATION and GAUSSIAN ELIMINATION METHOD TO GET
                          THE NEXT TABLEAU : */
                                hold = a[m][n];
                                for(k=0 ; k<=n1; ++k) {
                                        if(k==n){
                                                a[m][n]=1/a[m][n];
                                        }
                                        else
                                                a[m][k] = a[m][k]/hold;
                                }
                                for(i=0 ; i < m ; ++i){
                                        hold2=(-(a[i][n]));
                                        for(j=0 ; j<=(n1) ; ++j){
                                                if(j==n){
                                        a[i][j]=a[i][j]/-hold;
                                        }
                                        else
                              a[i][j] = a[m][j] * hold2 + a[i][j];
                                   }
                         }
                         for(i=(m+1); i<= ml ; ++i){
                                   hold1=(-(a[i][n]));
                                   for(j=0 ; j<=nl ; ++j){
                                        if(j==n){
                                        a[i][j]=a[i][j]/-hold;
                                        }
                                        else
                              a[i][j] = a[m][j] * hold1 + a[i][j];
                                   }
                         }
                         i_file= fopen("matrix", "w");

                         for(k=0 ; k<= ml ; ++k){
                                   for(t=0 ; t<= nl ; ++t){
                                   fprintf(i_file,"%lf ",a[k][t]);
                                   }
                                   fprintf(i_file,"  \n ");
                         }
                         fclose(i_file);
                         i_file= fopen("matrix", "r+");
                         for(k=0 ; k<= ml ; ++k){
                                   for(t=0 ; t<= nl ; ++t){
                         fscanf(i_file," %lf  ",&a[k][t]);
                                   }
                         }
                         fclose(o2_file);
               }/* set */
               diff2=a[0][nl];
               difference = diff1 - diff2;
               diff1 = diff2;
               if (diff1 <= .00001){
                         difference = 0.0000001;
               }
               if (diff1 >= .001){
                         difference = 0.001;
               }
               if(set==unmark){
                         printf(" the set is unmarked \n ");
                         difference=.0000001;
               }
          }/* while*/
          o2_file=fopen("results","w");
/*********************************************************/

/* TO WRITE THE FINAL SOLUTION IN THE FORM OF THE EQUATIONS WE
FIND THE VARIABLES THAT ARE IN THE BASIS AND EVALUATE ALPHA AND
LAMBDA AND BETA AND PUT THEM IN THE EQUATION FORM */

for(i=0; i<=1; ++i){
          ans=chec[i];
          index0=i+1;
          if(ans!=100){
                    alpha[index0]=a[ans][nl];
          }
          else{
```

```
                        alpha[index0]=0.00000;
                }
        }
        for(i=2; i<=3; ++i){
                ans=chec[i];
                index1=i-1;
                if(ans!=100){
                        beta[index1]=a[ans][n1];
                }
                else{
                        beta[index1]=0.00000;
                }
        }
        for(i=4; i<=4+T-2;++i){
                ans=chec[i];
                if(ans!=100){
                        lambda[i-3]=a[ans][n1];
                }
                else{
                        lambda[i-3]=0.00000;
                }
        }
        for(i=4+T-1; i<=4+2*T-2; ++i){

                ans=chec[i];
                index2=i-T-2;
                if(ans!=100){
                        e[index2]=a[ans][n1];
                }
                else{
                        e[index2]=0.00000;
                }
        }
        alp[1]=alpha[1]-alpha[2];

        /*      printf("the val of alpha %lf \n",alp[1]); */

        bet=beta[1]-beta[2];
        /*      printf("beta val %lf \n ",bet); */
        for(i=2; i<=T; ++i){
                alp[i]=alp[i-1]-e[i-1];
        }
        for(i=1; i<=T; ++i){
                if(bet <0.0){
        fprintf(o2_file," %d. yt =  %lf %lf xt\n ",i,alp[i],bet);
                }
                else
                {
        fprintf(o2_file," %d. yt =  %lf + %lf xt\n ",i,alp[i],bet);
                }
        }
        fprintf(o2_file,"\n \n");
        fprintf(o2_file," TOTAL NO OF PIVOTS PERFORMED : %d ",pivotno);
}/* main*/
        /*****************************************************/
           /* THIS FUNCTION PERFORMS BUBBLE SORT ON THE ELEMENTS IN THE FIRST
        ARGUMENT AND SAVES THEM ACCORDING TO THE COLUMN NUMBERS */
        int bubble(x,co,SIZE)
        double x[];
        int SIZE,co[];
```

```
{
        int i,j,temp;
        double templ;

        for (i = 0; i < SIZE - 1; ++i){    /* bubble sort */
                for (j = SIZE - 1; j > i; --j){
                if (x[j - 1] < x[j]) {  /* check the order */
                                temp = co[j - 1];
                                templ=x[j-1];
                                co[j - 1] = co[j];
                                x[j-1]=x[j];
                                co[j] = temp;
                                x[j]=templ;
                        }
                }
        }
}
/**********************************************************/
```