U N I V E R S I T Y O F R E A D I N G

INEQUALITY CONSTRAINED FINITE ELEMENTS AND CONJUGATE GRADIENTS

N.R.C. Birkett

Numerical Analysis Report 16/89

D E P A R T M E N T O F M A T H E M A T I C S

INEQUALITY CONSTRAINED FINITE ELEMENTS AND CONJUGATE GRADIENTS

N.R.C Birkett

Numerical Analysis Report 16/89

Department of Mathematics

P.O. Box 220

University of Reading

Whiteknights

Reading RG6 2AX

U.K.

ABSTRACT

An extension of the Conjugate Gradient Iterative method is described for inequality constrained minimisation of a positive definite linear/quadratic functional. Several applications to Finite Element approximation are discussed , and the described methods are applied to an obstacle problem, and two constrained approximation problems. The use of constrained minimisation in Moving Finite Element methods is also detailed.

ACKNOWLEDGEMENTS

I would like to thank Dr. Mike Baines for his perserverence in programming, his enthusiasm, and his faith in the assuming that most if not all bugs are eventually detected. Thanks also to Carol Reeves for pointing out quite alot of bugs, for assistance with UNIRAS graphics, and for the use of the Moving Element program. I just hope it was all worthwhile. Finally I would like to thank SERC and the University of Reading for much appreciated sources of income.

CONTENTS

SECTION 1. INTRODUCTION

In this report we present the application of simple quadratic programming techniques to constrained optimisation problems where the matrix describing the quadratic form is symmetric, positive definite, large and sparse. It might be thought that the solution of such problems would be computationally expensive. We aim to show, however, that there are many constrained optimisation problems, occuring in Finite Element work, which can be solved efficiently using sparse matrix techniques. This opens the door to the practical solution of a whole variety of problems other than the "first variation equals zero" type.

In section 2 a brief account of constrained optimisation theory is included for the benefit of those unaquainted with it, and in order to explain easily how methods can be constructed to perform the required task. Examples useful in later sections are presented.

The application of the Conjugate Gradient Method to Finite Element problems is discussed in section 3, together with an observation on how to avoid the requirement of "overwriting" essential boundary conditions. An algorithm based on Conjugate Gradient iteration for the minimisation of quadratic functionals subject to non-negativity constraints on the variables is described. It is proved that the algorithm successfully terminates only if the solution of the quadratic programming problem has been found.

The algorithm presented in section 3 is generalised in section 4 to allow minimisation subject to bounds on the variables, and to the case where some of the variables are required to satisfy an ordering constraint.

In section 5 applications to some Finite Element problems are presented with numerical examples, and in section 6 a numerical scheme for a practical moving grid element method is proposed.

SECTION 2. CONSTRAINED OPTIMISATION THEORY.

In this section we give a brief outline of the optimisation theory which will be required in later sections, together with some relevant examples.

2.1 Convex Sets.

Since we are concerned here with discrete approximations to continuous problems, we will deal with optimisation theory only as it pertains to a finite dimensional vector space V.

Definition 1: A subset X of V is called *convex* if it satisfies the property :

$$\theta\, x + (1 - \theta)\, y \in X \quad \forall\, x,y \in X,\ \forall\, \theta \in [0,1].$$

Examples of convex subsets of $\mathbb{R}^n$ of interest in Finite Element work are :

(i) $X = \mathbb{R}^n$.

(ii) $X = \{\, x \in \mathbb{R}^n : x_{j_1} = a_1,\ x_{j_2} = a_2, \ldots, x_{j_k} = a_k \,\}$,

i.e. X is an n-k dimensional subspace of $\mathbb{R}^n$ in which k of the n components of x are fixed.

(iii) $X = \{\, x \in \mathbb{R}^n : x_{j_1} \geq 0\ ,\ x_{j_2} \geq 0\ , \ldots, x_{j_k} \geq 0 \,\}$,

i.e. k of the components of x are required to be non-negative.

(iv) $X = \{\, x \in \mathbb{R}^n : x_{j_1} \leq x_{j_2} \leq \ldots \leq x_{j_k} \,\}$,

i.e. k of the n components of x are required to appear in numerical order.

(v) $X = \{\, x \in \mathbb{R}^n : a_m \leq x_{j_m} \leq A_m \ ,\ \text{for } m=1,2,..,k \,\}$,

i.e. k of the n variables are required to lie in a given interval (including the case $a_m = A_m$).

2.2 Fundamental Optimality Conditions.

Let Ψ be a continuous functional on a closed convex set $X \subseteq V$.

Definition 2. $v \in X$ is called a *minimising element* if

$$\Psi(v) \leq \Psi(x) \quad \forall\, x \in X.$$

Since X is closed, Ψ is continuous, and V is finite dimensional, then v exists. Note that none of the sets (i) - (v) is necessarily closed. However in Finite Element work Ψ will often be of such a nature as to guarantee at least one minimising element.

Optimality Theorem

Suppose Ψ is differentiable on X, then a necessary condition that $v \in X$ is a minimising element is

$$\Psi'(v)(x - v) \geq 0 \quad \forall\, x \in X\,. \tag{2.1}$$

Proof: Suppose $v \in X$ is a minimising element and let x be any element of X, then

$$y = (1 - \theta)\, v + \theta\, x \in X \quad \forall\, \theta \in [0,1],$$

as X is convex. Hence

$$\Psi(y) \geq \Psi(v)\,,$$

and hence

$$\frac{\Psi(v + \theta(x - v)) - \Psi(v)}{\theta} \geq 0 \qquad \forall\, \theta \in (0,1]$$

Since Ψ is differentiable, we therefore have in the limit as $\theta \to 0$

$$\Psi'(v)(x - v) \geq 0,$$

and since $x \in X$ was arbitrary, inequality (2.1) is established. Inequality (2.1) is called the Kuhn-Tucker condition for the non-linear programming problem [8]. In most cases of interest in this report, (2.1) is also a sufficient condition for v to be

minimising as the following standard result shows.

Sufficiency Condition.

If Ψ is twice differentiable on X and is such that Ψ'' is positive semi-definite for all $x \in X$ (i.e. $(v-w)\Psi''(x)(v-w) \geq 0$ $\forall\ x,v,w \in X$), then (2.1) is also a sufficient condition for $v \in X$ to be a minimising element.

Proof: Suppose $v \in X$ satisfies condition (2.1) and suppose Ψ'' is positive semi-definite on X, then if w is any element of X, we have by Taylor's theorem

$$\Psi(w) = \Psi(v) + \Psi'(v)(w-v) + \tfrac{1}{2}(w-v)\Psi''(z)(w-v) \qquad (2.2)$$

where $z = \theta v + (1-\theta)w$ for some $\theta \in [0,1]$. Hence $\Psi(w) \geq \Psi(v)$ and $\Psi(v)$ must be the minimum value of Ψ on X.

Uniqueness: If Ψ'' is strictly positive definite on X, then the solution is unique since then (2.2) implies that $\Psi(w) > \Psi(v)$ whenever $w \neq v$.

Thus, whenever Ψ'' is positive semi-definite, condition (2.1) gives us a rigorous test as to whether or not a given element v is minimising. Condition (2.1) is a generalisation of the test : "Is the the gradient of Ψ zero at v ?". In many cases condition (2.1) is easy to apply as we now show with a few examples.

2.3 Examples.

A routine problem faced in Finite Element work is

(i) $$\min_{x \in \mathbb{R}^n} \Psi(x)$$

where $$\Psi(x) = \tfrac{1}{2}\, x^T A x - x^T b,$$

A is a real symmetric positive definite n X n matrix, and b is a

real n-vector.

In this case

$$\Psi'(x) = Ax - b,$$

and hence condition (2.1) is then

$$(w-v)^T(Av - b) \geq 0 \qquad \forall\, w \in \mathbb{R}^n.$$

Since this inequality has to hold for every w, and since the components of (w-v) can be of either sign, we conclude that

$$Av - b = 0,$$

which is both a necessary and sufficient condition on the minimising element v in this case.

A slightly less trivial example is when we have to perform the minimisation of Ψ subject to some of the variables taking on specified values. We then have the problem

(ii) $$\min_{x \in \mathbb{R}^n} \Psi(x)$$

subject to $x_{j_k} = a_k$, for k= 1,2,..., m ≤ n. Condition (2.1) for this problem is simply

$$[\, Av - b \,]_i = 0 \ ,$$

whenever $i \neq j_k$, k=1,2,,,, m, since the vector (w-v) in this case has zeros in the components j_k. In other words the values of the components of the "residual" Ax-b corresponding to the free variables must be zero at the minimum, the other components of the residual can have any value. This idea will be used to some advantage when we examine inequality constrained minimisation problems.

The next example is concerned with data fitting by Finite Elements. Suppose we are given a Finite Element mesh on which we have to approximate some initial function u(s) before applying a time-stepping scheme. Because of error considerations we would normally take the best least squares Finite Element fit to u(s) on the solution domain Ω. If we have n Finite Element basis functions $\phi_1,\phi_2,.....,\phi_n$, then the Finite Element fit to u(s) is given by

$$u_h(s) = \sum_1^n u_j \phi_j(s),$$

where the L_2 error, E is given by

$$E = \int_\Omega (u_h(s) - u(s))^2 d\Omega \quad .$$

The best L_2 fit is then given by minimising E with respect to the nodal parameters u_j, which is equivalent to minimising the functional

$$\Psi(v) = \tfrac{1}{2}\, v^T A\, v - v^T b,$$

where $v = (u_1,u_2,....,u_n)^T$,

$$[\ A\ ]_{ij} = \int_\Omega \phi_i \phi_j \, d\Omega \ , \text{ and}$$

$$[\ b\ ]_j = \int_\Omega u\, \phi_j \, d\Omega \ .$$

The symmetric n X n positive definite matrix A is the usual "mass" matrix. On some occaisions the solution of the linear system Av = b, may yield formally unacceptable values for the nodal parameters u_i. For example if u(s) represents an initial pressure or density profile, then unconstrained minimisation of Ψ may lead to some of the nodal parameters being negative. In this case it

might be better to solve the problem

(iii) $$\min_{x \in \mathbb{R}^n} \Psi(x)$$

subject to $x_j \geq 0$, for $j = 1,2,\ldots,n$.

Here, the components of the vectors v and w in (2.1) may take on only non-negative values. Thus condition (2.1) is seen to be equivalent to the following rules :

At the solution v of problem (iii) either

$$[\,Av - b]_i > 0 \text{ and } [v]_i = 0 \qquad (2.3)$$
$$\text{or} \quad [\,Av - b]_i = 0,$$

for each component i = 1,2,..,n.

This result can be extended to the problem

(iv) $$\min_{x \in \mathbb{R}^n} \Psi(x)$$

subject to $a_j \leq x_j \leq A_j$, for j=1,2,..,n. Using condition (2.1), the solution of problem (iv) can be characterised by the following :

$$\begin{aligned} \text{if} \quad & [\,Av - b]_i > 0 \quad \text{then } [v]_i = a_i, \\ \text{if} \quad & [\,Av - b]_i < 0 \quad \text{then } [v]_i = A_i, \qquad (2.4) \\ \text{otherwise} \quad & [\,Av - b]_i = 0 \quad \text{and} \quad [v]_i \in [a_i, A_i]. \end{aligned}$$

SECTION 3. ALGORITHMS.

In this section we present a constrained quadratic programming algorithm which is based on the Conjugate Gradient (CG) method, a detailed description of which may be found in [6]. In order to develop ideas it is useful to give a brief summary of the CG iterative algorithm.

3.1 The Conjugate Gradient Method.

For the problem

$$\min_{x \in \mathbb{R}^n} \Psi(x),$$

where $$\Psi(x) = \frac{1}{2} x^T A x - x^T b,$$

A is an n X n symmetric positive definite matrix and b is a real n-vector, the CG iterative method for finding the solution x can be described by the following steps :

Algorithm 1.

STEP 0 : Choose any $x_0 \in \mathbb{R}^n$. Let $r_0 = A x_0 - b$; $k = 1$.

STEP 1 : If $r_{k-1} = 0$ then

$x = x_{k-1}$

stop

endif

STEP 2 : If $k = 1$ then

$\beta_k = 0$

else

$$\beta_k = \frac{r_{k-1}^T r_{k-1}}{r_{k-2}^T r_{k-2}}$$

endif

STEP 3 : $q_k = r_{k-1} + \beta_k q_{k-1}$

STEP 4 : $$\alpha_k = \frac{r_{k-1}^T r_{k-1}}{q_k^T A q_k}$$

STEP 5 : $x_k = x_{k-1} - \alpha_k q_k$; $r_k = r_{k-1} - \alpha_k A q_k$

STEP 6 : k := k + 1; go to STEP 1

It is known that, using infinite precision arithmetic, Algorithm 1 will terminate in at most n loops of the iteration [6]. However, in practice, the method is used iteratively and the iterations are terminated when $|r_{k-1}| < tol\ |b|$ in STEP 1, where tol is some small machine dependent tolerance. For each iteration the method requires 1 matrix/vector product (Aq_k), 2 inner products, and 3 vector/vector additions. It is the matrix/vector product calculation which requires most attention if the method is to be used as an effective "sparse" solver. When using Finite Elements it is usual to set up an element-node "connectivity" table [5]. This can be conveniently used to set up the required data structure needed to create an efficient sparse matrix/vector product routine, and requiring the storage of only the non-zero elements of the upper triangle of A.

3.2 Enforcement of essential boundary conditions.

In many Finite Element problems the function Ψ has to be minimised subject to some of the variables taking on prescribed values (essential boundary conditions). The usual procedure is to replace the vector b by $b - A\hat{x}$ where $\hat{x}$ is a vector containing the prescribed values in the appropriate components and zeros everywhere else. The matrix A is then "overwritten" with a unit sub-matrix along rows and columns corresponding to the known values, and the vector $b - A\hat{x}$ is overwritten with the known values in the correct positions [14]. This produces a new set of

equations $\hat{A}\, x = \hat{b}$ which are then solved by some method. If the CG method is being used to solve the problem then this overwriting precedure can be completely avoided by using the following modification of Algorithm 1.

Replace STEPs 0,1 of Algorithm 1 by

STEP 0^*: Choose any *feasible* $x_0 \in \mathbb{R}^n$; $r_0 = A\, x_0 - b$; $k = 1$.

STEP 1^*: For j = 1 to n

$[r_{k-1}]_j = [r_{k-1}]_j * [ib]_j$

next j

If $r_{k-1} = 0$ then

$x = x_{k-1}$

stop

endif

A *feasible* x_0 is simply a vector containing the prescribed values written into the appropriate components, and the n-vector ib is defined by

$[ib]_j = 0$ if $[x]_j$ is given

$[ib]_j = 1$ if $[x]_j$ is free, j=1,2,...,n.

STEP 1^* has the effect of overwriting the components of r_{k-1} (and hence of q_k) with zeros at the positions corresponding to known values. It is easy to show that this method is equivalent to the overwriting process, but requires an extra n multiplications per iteration of the CG method. However it is a useful device since we have only to change the vector ib to redefine which variables are fixed and which are free. This idea will be used in subsequent

constrained minimisation procedures.

3.3 A CG type solver for minimisation with positivity constraints.

Our basic stategy for devising a constrained minimisation routine around the conventional CG algorithm will be as follows :

1. Apply the CG Algorithm until one or more constraints are violated.

2. Apply the the CG Algorithm to a sub-problem in which some of the variables are held on the constraint set boundary.

The main problems in applying these ideas are how to carry on with the CG method once a constraint has been violated, and how to bring variables back into a sub-problem once they have been fixed.

We start by looking at the minimisation of Ψ, subject to $x_j \geq 0$, for $j=1,2,.....,n$. Suppose we attempt to use Algorithm 1 to solve this problem. We start at STEP 0 with a feasible x_0 so that $[x_0]_j \geq 0$ for $j=1,2,...,n$. A problem may occur in STEP 5 where we compute

$$x_k = x_{k-1} - \alpha_k q_k,$$

which may render some of the components of x_k negative. This can be prevented by calculating α_{max} where

$$\alpha_{max} = \min_{[q_k]_j > 0} [x_{k-1}]_j / [q_k]_j ,$$

then setting $\alpha_k := \min(\alpha_{max}, \alpha_k)$. In the case that $\alpha_{max} < \alpha_k$, the CG algorithm has been modified and we can no longer continue. When this occurs we must have at least one component of x_k equal to zero. If, in addition, the corresponding component(s) of the gradient vector (r_k) is greater than zero then we can fix the appropriate component of x_k at zero and restart the CG algorithm

on the new problem of minimising Ψ subject to

$$[x]_j \geq 0, \text{ with } [x]_m = 0 \text{ for some of the components } m.$$

If we find that a component of x_k is zero but the corresponding component of r_k is ≤ 0 then we can restart the CG iteration without fixing the component of x_k to zero. This is because the first CG iteration is equivalent to a classical "gradient" step which must improve the value of the functional without violating the non-negativity constraint since α_k is always a non-negative number. These ideas are summarised in the following :

Algorithm_2.

STEP 0 : Choose any feasible $x_0 \in \mathbb{R}^n$.

$r_0 = Ax_0 - b$

$I_0 = I$ (the n X n identity matrix)

iflag = 1

k = 1

STEP 1 : If iflag = 1 then

$v_{k-1} = I_{k-1} r_{k-1}$

$s_{k-1} = v_{k-1}^T v_{k-1}$

$\beta_k = 0$

else

$s_{k-1} = v_{k-1}^T v_{k-1}$

$\beta_k = s_{k-1}/s_{k-2}$

endif

iflag = 0

STEP 2 : $q_k = v_{k-1} + \beta_k q_{k-1}$

STEP 3 : If $s_{k-1} = 0$ then

$x = x_{k-1}$

stop

STEP 4 : $p_k = A\, q_k$

$\alpha_k = s_{k-1} / p_k^T q_k$

STEP 5 $\alpha_{max} = \max_{[q_k]_j > 0} [x_{k-1}]_j \;/\; [q_k]_j$

If $\alpha_{max} < \alpha_k$ then

$\alpha_k = \alpha_{max}$

iflag = 1

endif

STEP 6 : $x_k = x_{k-1} - \alpha_k q_k$

$r_k = r_{k-1} - \alpha_k p_k$

$v_k = I_{k-1} r_k$

STEP 7 : for j = 1 to n

If $[x_k]_j = 0$ and $[r_k]_j > 0$ then

$[I_k]_{jj} = 0$

else

$[I_k]_{jj} = 1$

next j

STEP 8 : $\Delta = | I_k - I_{k-1} |$

If $\Delta > 0$ then

iflag = 1

endif

STEP 9 : k := k + 1 ; go to STEP 1

It is noted that, in practice, I_k is stored as a vector of flags ib(j) = 0 or ib (j) = 1, as described in section 3.2. The

vector v_k is simply the residual for the sub-problem of minimising Ψ subject to some of the variables being held constant. We now show that if Algorithm 2 terminates successfully at STEP 3, then x is the solution of the constrained minimisation problem.

Suppose at some stage $s_{k-1} = 0$ in STEP 3, then r_{k-1}, x_{k-1} must have the following properties :

$$\text{either} \quad [r_{k-1}]_j > 0 \text{ and } [x_{k-1}]_j = 0$$

$$\text{or} \quad [r_{k-1}]_j = 0.$$

However, $r_{k-1} = A\, x_{k-1} - b = \Psi'(x_{k-1})$, and hence $s_{k-1} = 0$ implies that x_{k-1} satisfies conditions (2.3), which are both necessary and sufficient conditions on the solution x.

SECTION 4.EXTENSIONS.

In this section we present some straightforward extensions to Algorithm 2.

4.1 Problems with bounded variables.

Here we consider the problem of minimising the functional Ψ subject to $a_j \leq x_j \leq A_j$, for $j=1,2,\ldots,n$, where the a_j's and A_j's are given constants.

It is seen that the same ideas used in deriving Algorithm 2 can be applied to the current problem, provided that we change our notion of a feasible vector, recast STEP 5 so that the updated approximation x_k is always feasible, and rework STEP 7 so that variables are fixed or brought back into the CG iteration in line

with condition (2.4). If these tasks are carried out then we get a CG type algorithm which can be described by the following steps.

Algorithm 3.

Complete all the steps of Algorithm 2, section 3.3, with the exception of STEP 5 and STEP 7, which are replaced by

$$\text{STEP } 5^{*}: \alpha = \min_{j} \left\{ \frac{[x_{k-1}]_j - A_j}{[q_k]_j} \text{ if } [q_k]_j < 0 \; ; \; \frac{[x_{k-1}]_j - a_j}{[q_k]_j} \text{ if } [q_k]_j > 0 \right\}$$

If $\alpha < \alpha_k$ then

$\alpha_k = \alpha$

iflag = 1

endif

STEP 7^{*} : for j=1 to n

If $[x_k]_j = A_j$ and $[r_k]_j < 0$ then

$[I_k]_{jj} = 0$

else

$[I_k]_{jj} = 1$

endif

If $[x_k]_j = a_j$ and $[r_k]_j > 0$ then

$[I_k]_{jj} = 0$

else

$[I_k]_{jj} = 1$

endif

next j

It is seen that STEP 7^{*} ensures that Algorithm 3 only terminates

in STEP 3 if x_{k-1} satisfies necessary and sufficient conditions (2.4).

4.2 Problems with ordered variables.

The last problem we consider in this section is that of minimising Ψ subject to the ordering constraint

$$0 \leq x_1 \leq x_2 \leq \ldots\ldots \leq x_n.$$

This problem can be recast as an equivalent problem where the variables are required to be non-negative by a change of variables as follows:

$$\text{Let } y_1 = x_1,$$

$$y_j = x_j - x_{j-1}, \text{ for } j = 2,3,\ldots\ldots,n,$$

so that we get the new problem :

$$\min_{y \in \mathbb{R}^n} \Phi(y) = \frac{1}{2} y^T B y - y^T d$$

subject to $y_j \geq 0$, for $j = 1,2,\ldots,n$,

where $B = D^{-T} A D^{-1}$, $d = D^{-T} b$,

and D is an n X n matrix describing the change of variables from x to y given by

$$D = \begin{pmatrix} 1 & 0 & 0\ldots\ldots & 0 \\ -1 & 1 & 0\ldots\ldots & 0 \\ 0 & -1 & 1\ 0\ \ldots. & 0 \\ \ldots\ldots\ldots\ldots & & & \\ 0\ldots\ldots & & 0\ -1 & 1 \end{pmatrix}. \qquad (4.1)$$

The only problem now is to show how to form the matrix vector product B y in such a way as to take advantage of the possible sparsity of matrix A. This problem is easily overcome if the product is built up in three stages :

(i) $z = D^{-1}y$

(ii) $w = A\,z$ (sparse matrix /vector product)

(iii) $p = D^{-1}w$,

the vector p being the required product. As D is lower triangular and bi-diagonal then steps (i) and (iii) may be efficiently implemented using the following algorithms :

$(z = D^{-1}y)$ $z_1 = y_1$

for j = 2 to n

$z_j = y_j + z_{j-1}$

next j

$(p = D^{-T}w)$ $p_n = w_n$

for j = n-1 to 1 step -1

$p_j = w_j + p_{j+1}$

next j

It is seen that the computational overhead incurred is an extra 2n-2 additions compared with the simple product A w .

4.3 Other extensions.

Finally it may be noted that all the algorithms discussed so far can be very easily modified for use on problems where the constraints apply only to a subset of the n variables.

4.4 Practical considerations.

In order to implement Algorithms 1,2, and 3 it is necessary to define what we mean by a quantity being < 0, $= 0$, or > 0 on a given machine. We illustrate with reference to Algorithm 2 section 3.3. The key places where the use of a machine zero tolerance is important are in steps 3, and 7. As in the conventional CG method,

step 3 should be replaced by

$$\text{If } |s_{k-1}| < \text{tol } |b| \text{ then } x = x_{k-1}; \text{ stop,}$$

where tol is a small positive machine dependent parameter. Care should be taken in step 7 to set a component equal to zero if it is within a small tolerance of zero and the corresponding component of the residual vector is positive. This will prevent the algorithm from setting the restart flag "iflag" each time through the loop when a component is really zero. We can rewrite this step as :

$$\text{If } |[x_k]_j| < \text{tol and } [r_k]_j > 0 \text{ then}$$
$$[x_k]_j = 0$$

SECTION 5. APPLICATIONS.

In this section we illustrate the use of the Algorithms discussed in sections 3 and 4, by applying them to some simple problems. Unless otherwise indicated we will use one-dimensional Lagrange quadratic elements [5] on a uniform discretisation of the unit interval $0 \le s \le 1$. If $\phi_1(s),\phi_2(s),\ldots\phi_n(s)$ are the Finite Element basis functions, then we define the stiffness matrix K, the mass matrix M, and the load vector b as follows :

$$[\ K\ ]_{ij} = \int_0^1 \phi_i'(s)\phi_j'(s)\ ds, \qquad \text{for } i,j = 1,2,\ldots n$$

$$[\ M\ ]_{ij} = \int_0^1 \phi_i(s)\phi_j(s)\ ds,$$

$$[\ b\ ]_i = \int_0^1 \phi_i(s)\ f(s)\ ds, \qquad \text{for } i = 1,2,\ldots,n,$$

where f(s) is a given function continuous on [0,1]. The matrices K and M are n X n, real and symmetric. The stiffness matrix K is positive semi-definite, of rank n-1, and the mass matrix is positive definite. All examples are coded in Fortran 77 and run using double precision arithmetic on a SUN 3/60 workstation. The machine tolerance, tol, as discussed in section 4.3, is taken to be 10^{-10}.

5.1 The Obstacle Problem.

We consider the application of Finite Elements to the problem of finding the displacement of a uniform elastic string when it is stretched between two fixed points over an obstacle [4,11], and subject to an external load f(s). If g(s) ≥ 0 is a given continuous function on [0,1] defining the upper surface of the obstacle and u(s) is the displacement of the string at the point s (fig.1), then the problem can be shown to be equivalent to that of minimising the elastic energy in the string subject to u(0), and u(1) being prescribed and the constraint u(s) ≥ g(s).

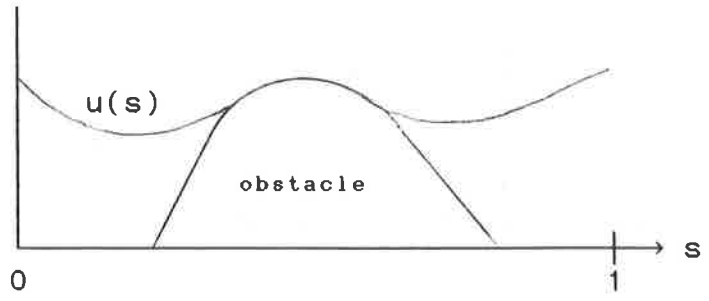


Figure 1.

We thus have the variational problem :

$$\min_{v \in D} E(v), \text{ where}$$

$$E(v) = \frac{1}{2} \int_0^1 v'(s)^2 - 2\, f(s)\, v(s)\, ds,$$

$$D = \{ v \in H_E^1(0,1) : v(s) \geq g(s) \quad \forall\, s \in (0,1) \},$$

and $H_E^1(0,1) = \{ v : \int_0^1 v'^2 ds < \infty$ and $v(0), v(1)$ given $\}$.

The solution of this problem is known to be characterised by the variational inequality

$$\int_0^1 u'(s)(v'(s) - u'(s)) - f(s)(v(s) - u(s))\, ds \geq 0 \quad \forall\, v \in D. \quad (5.1)$$

An obvious Finite Element approximation to this problem is

$$\min_{u_h \in D_h} E(u_h)$$

where $u_h(s) = \sum_1^n u_j\, \phi_j(s)$ and

$$D_h = \{ u_h : \; u_h \in D \}.$$

However, this is not very practical since it would be difficult to find the feasible set of nodal parameters $\{ u_j \}$ such that $u_h(s) \geq g(s)$ for $s \in (0,1)$. Instead we look at a simpler discrete problem

$$\min_{u_h \in D_h'} E(u_h), \text{ where}$$

$$D_h' = \{ u_h : u_j \geq g(s_j) \text{ , for } j=2,3,\ldots,n-1 \}.$$

In other words the constraint on the Finite Element solution is only applied at each node s_j. Without giving any details, this can be seen to be equivalent to perturbing the data g(s) into the

Finite Element approximation space.

The Finite Element problem is then

$$\min_{U \in \mathbb{R}^n} \Psi(U)$$

subject to $u_j \geq g(s_j)$, u_1, u_n given, where

$$\Psi(U) = \frac{1}{2} U^T K U - b^T U,$$

$U = (u_1, u_2, \ldots\ldots, u_n)^T$, K is the stiffness matrix and b is the load vector. It is interesting to note that inequality (2.1), which characterises the solution to this finite dimensional problem, can be written in this case as

$$\int_0^1 u_h'(s)(v_h'(s) - u_h'(s)) - f(s)(v_h(s) - u_h(s))\, ds \geq 0 \quad \forall\, v_h \in D_h'$$

which may be compared with condition (5.1) on the exact solution.

As an example we take an obstacle with two "peaks" defined by

$$g(s) = \sin^2(2\pi s)/(1+s) \qquad \forall\, s \in [0,1].$$

and the fixed points to be

$$u(0) = \frac{1}{2} \ , \ u(1) = 0.$$

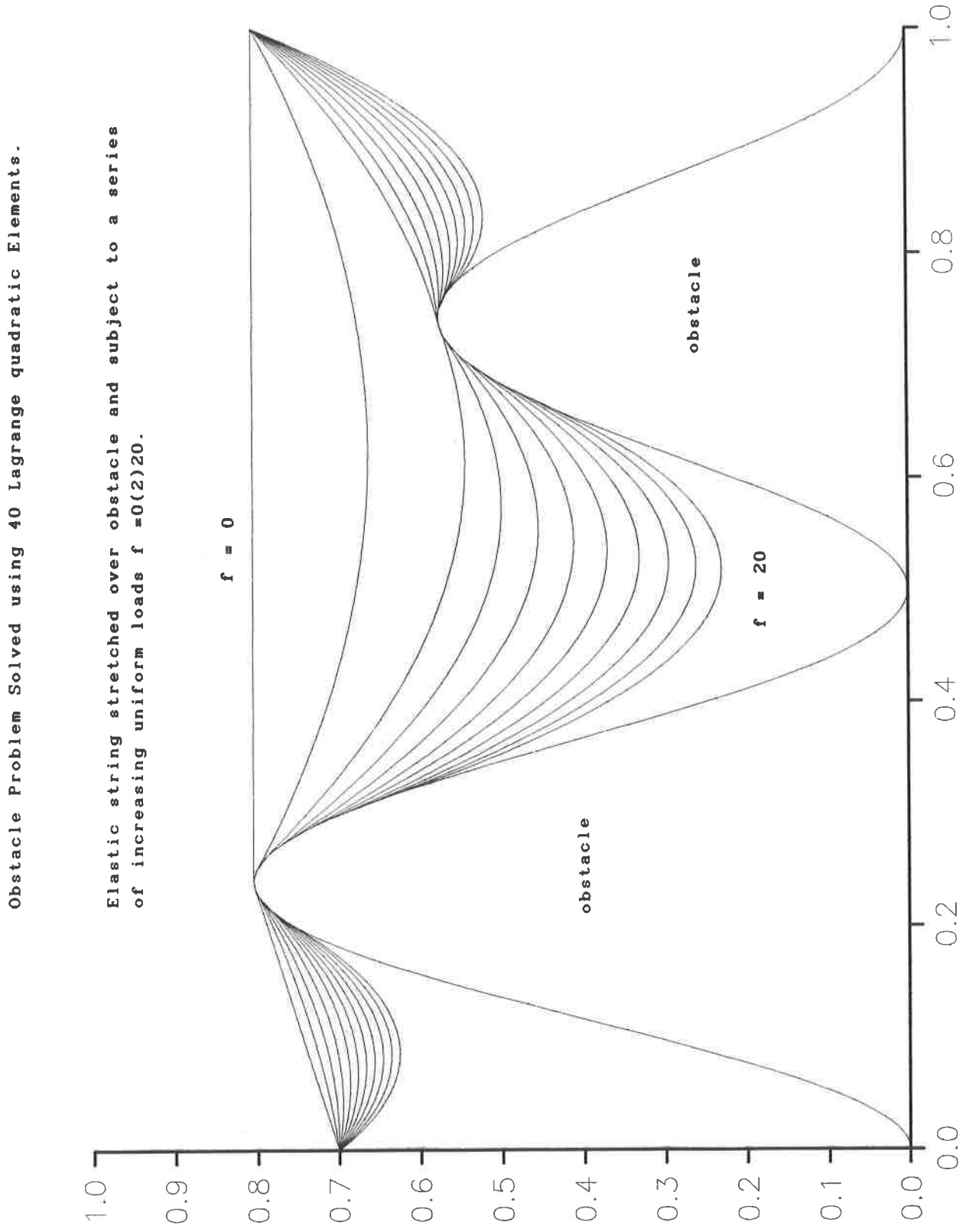
The results of a series of calculations using Algorithm 3 from section 4 are presented in Table 1 for a zero imposed load $f(s) \equiv 0$. As a comparison with the unconstrained CG solver Algorithm 1 section 3, we also present the number of CG iterations required to solve the same problem but without the obstacle. For each calculation the initial feasible vector was taken to be $u_1 = \frac{1}{2}$, $u_n = 0$, and $u_j = 2$, for $j = 2,3,\ldots,n-1$. It is seen from Table 1 that the constrained CG solver requires up to 7 times the work of a conventional CG solver. A typical set of Finite Element

solutions is presented in Figure 2 for a set of increasing uniform loads f = 0 (2) 20.

CG tolerance = 10^{-10}			
No. of elements	No. of variables	No. of iterations	NCG
4	9	12	7
8	17	28	15
32	65	246	63
70	141	667	141
150	301	2073	316
NCG is the number of CG iterations needed to solve the unconstrained problem.			

Table 1. Obstacle problem with zero load.

Figure 2.


Obstacle Problem Solved using 40 Lagrange quadratic Elements.
Elastic string stretched over obstacle and subject to a series of increasing uniform loads f =0(2)20.
f = 0
obstacle
f = 20
obstacle
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
0.2
0.4
0.6
0.8
1.0

5.2 Best non-negative approximations.

The next problem we consider is that of approximating a given continuous function, f(s), by Finite Elements in such a way that the nodal values are non-negative. If the Finite Element approximation is given by

$$f_h(s) = \sum_1^n f_j \phi_j(s),$$

then the finite dimensional problem is

$$\min_{F \in \mathbb{R}^n} \Psi(F),$$

subject to $f_j \geq 0$ for $j = 1,2,\ldots,n$, where

$F = (f_1, f_2, \ldots, f_n)^T$, and

$$\Psi(F) = \frac{1}{2} F^T M F - F^T b,$$

and M,b are the mass matrix and load vector respectively.

For the purposes of illustration, we take

$$f(s) = \begin{cases} 1 - 10s, & \text{if } s \leq 0.1, \\ 0, & \text{if } 0.10 \leq s \leq 0.25 \\ \sqrt{2s - .50}, & \text{if } 0.25 \leq s \leq 0.50 \\ f(s-0.50), & \text{if } 0.50 \leq s \leq 1.00 \end{cases},$$

and apply Algorithm 2 section 3 to minimise Ψ. Table 2 shows the number of iterations required to obtain the solution using various numbers of quadratic elements. The initial feasible vector was taken as $f_j = 2$, $j=1,2,\ldots,n$, in each case and we again provide a comparison between the constrained and unconstrained case of finding the best Finite Element least-squares fit to f(s). It is interesting to note, in this case, that the number of CG iterations required for the unconstrained solution is almost independent of the number of variables. This is a reflection of

the fact that, on a regular mesh, the condition number k_2 of the mass matrix M, where

$$k_2 = \| M \|_2 \| M^{-1} \|_2,$$

satisfies $k_2 \leq C$, where C is a constant [12]. It is known that the rate of convergence of the iterative CG method is directly related to k_2. It is also of note that the constrained CG method seems to depend only weakly on the number of variables in this case. We therefore speculate that "preconditioning" [6] of the matrix will benefit the convergence rate of a constrained CG solver. The only problem is to find a good preconditioning matrix which preserves the simplicity of the constraints (an example is a diagonal preconditioning matrix).

Figures 3 and 4 show some typical fits to f(s) in the unconstrained and constrained cases.

CG tolerance = 10^{-10}			
No. of elements	No. of variables	No. of iterations	NCG
5	11	9	6
10	21	11	9
20	41	11	9
40	81	15	7
NCG is the number of CG iterations needed to find the best L_2 Finite Element fit to f(s).			

Table 2. Best non-negative data fitting.

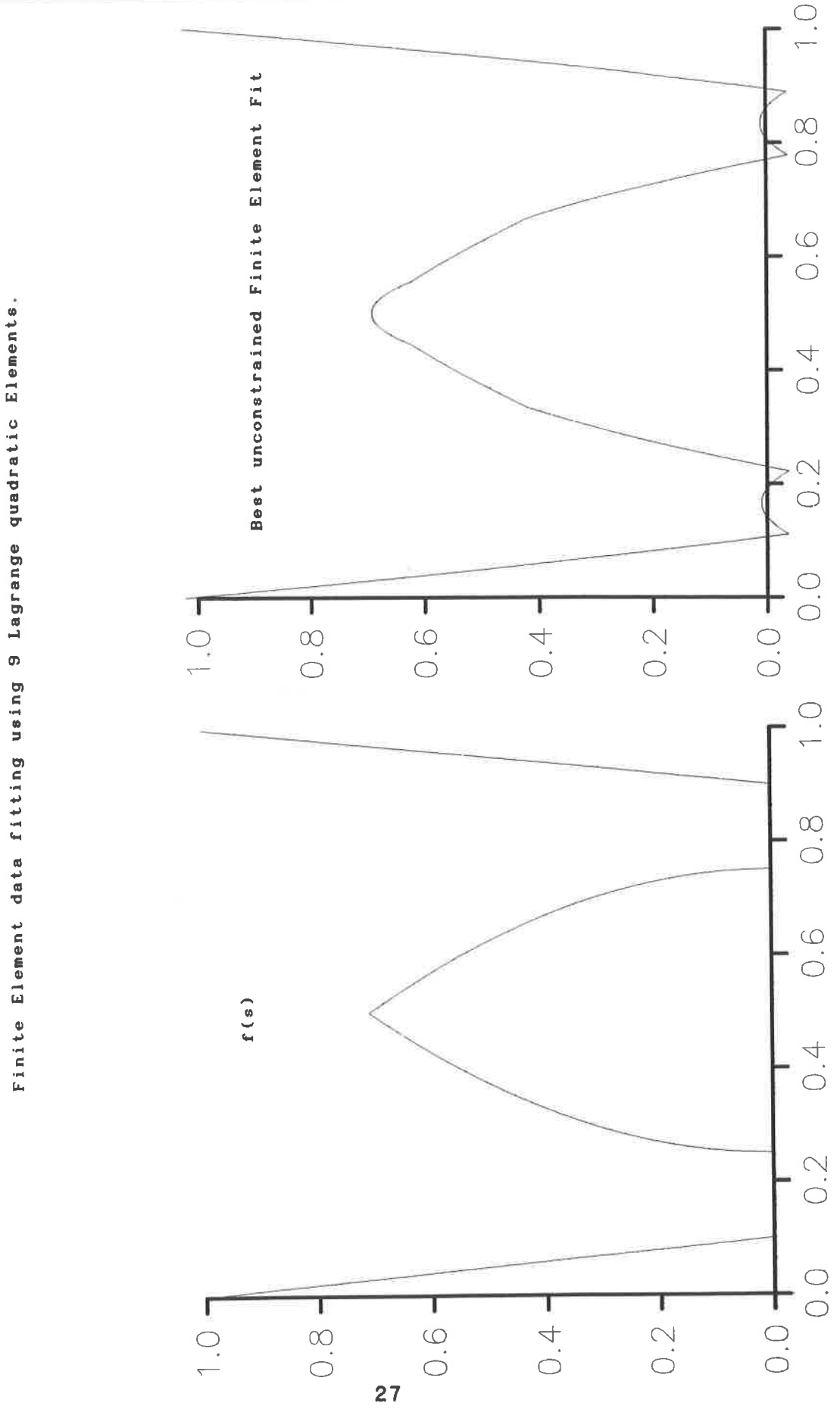

Finite Element data fitting using 9 Lagrange quadratic Elements.
Best unconstrained Finite Element Fit
f(s)
0.0
0.2
0.4
0.6
0.8
1.0

Figure 3.

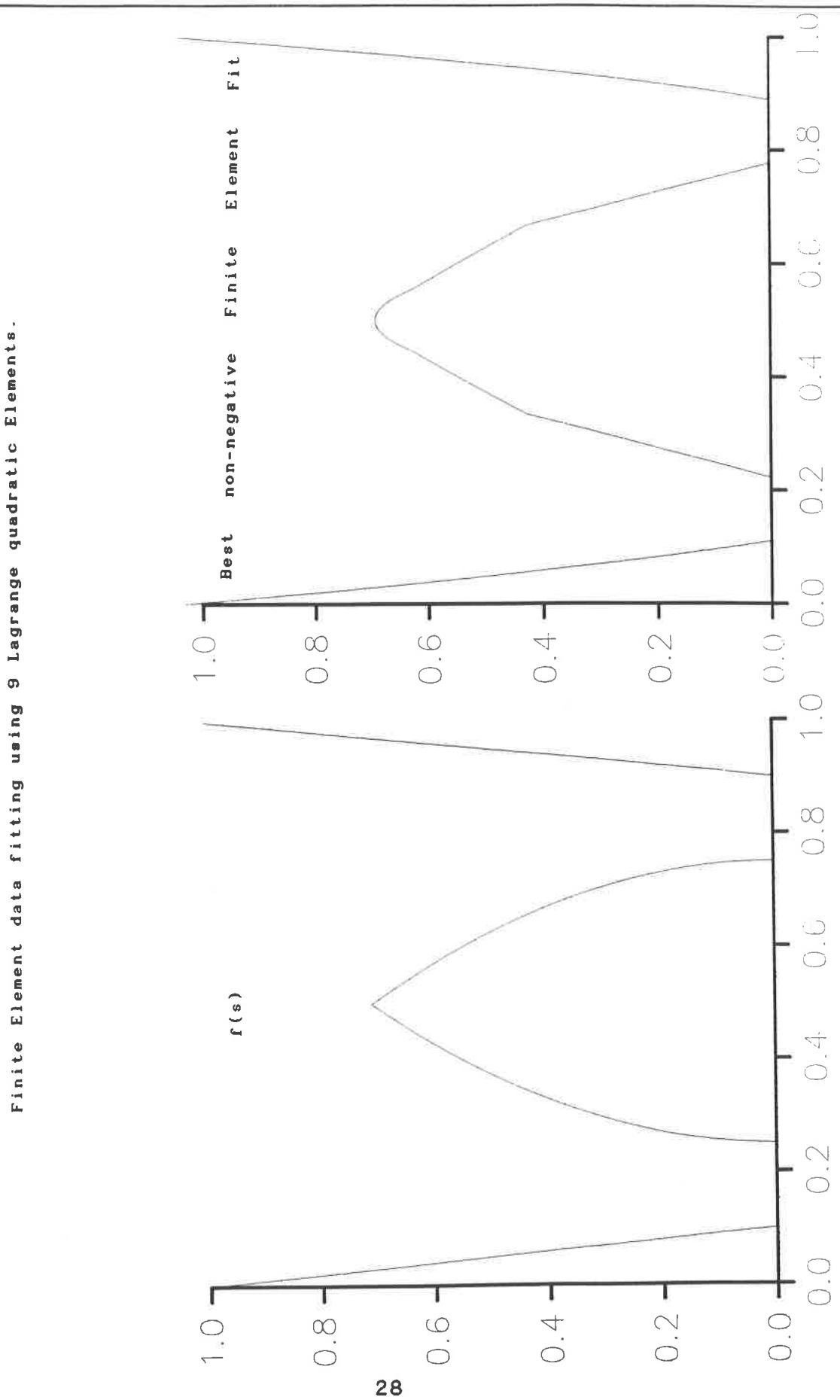

Finite Element data fitting using 9 Lagrange quadratic Elements.
Best non-negative Finite Element Fit
f(s)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Figure 4.

5.3 Monotonic data fitting.

The next appliciation is that of finding the best Finite Element least-squares fit to a continuous function f(s), subject to the fit being monotonic increasing. Again it is practical to apply the constraints node-wise giving the problem

$$\min_{F \in \mathbb{R}^n} \Psi(F),$$

subject to $0 \leq f_1 \leq f_2 \leq \ldots\ldots \leq f_n$, where

$F = (f_1, f_2, \ldots\ldots, f_n)^T$, and

$$\Psi(F) = \frac{1}{2} F^T M F - F^T b,$$

and M,b are again the mass matrix and load vector respectively. We take the following data

$$f(s) = 0.95 - e^{-10s} + 0.05 \cos(20\pi s),$$

and an initial feasible vector $f_j = j$, for $j = 1,2,\ldots n$. We apply Algorithm 2 section 3, together with the transformation of variables discussed in section 4.2. Table 3 gives the required number of iterations required to solve the constrained and unconstrained (best least-squares fit) problems. In this case the constrained CG method needs an unfavourably large number of iterations compared with the unconstrained case. As was remarked in section 5.2, the mass matrix M is very well conditioned in this case. However the constrained minimisation is achieved by applying Algorithm 2 to the functional

$$\Phi(y) = \frac{1}{2} y^T B y - y^T d$$

where $B = D^{-T} A D^{-1}$, $d = D^{-T} b$,

and D is the n X n matrix given in (4.2).
Therefore the convergence of the constrained CG method is likely

to be governed by the size of the condition number $k_2(B)$, which can estimated by

$$k_2(B) \leq k_2(D^TD)\ k_2(M).$$

As stated previously, $k_2(M)$ is almost independent of n, but

$$D^TD = \begin{pmatrix} 2 & -1 & 0 & 0 & \ldots & 0 \\ -1 & 2 & -1 & 0 & \ldots & 0 \\ 0 & -1 & 2 & -1 & \ldots & 0 \\ & & & & & \\ 0 & \ldots & & -1 & 2 & -1 \\ 0 & \ldots & & 0 & -1 & 2 \end{pmatrix},$$

and a simple calculation reveals here that

$$k_2(D^TD) = \cot^2\left(\frac{\pi}{n}\right),$$

which grows very rapidly with n, so that the condition of B can be quite large. Fortunately in this example the matrix M is well conditioned. This approach is not recommended for finding a best monotonic solution in the case where the matrix describing the quadratic form is poorly conditioned. Figures 5 and 6 give some solution curves for the constrained and unconstrained best fits.

CG tolerance = 10^{-10}			
No. of elements	No. of variables	No. of iterations	NCG
5	11	149	9
10	21	164	9
20	41	391	10
40	81	793	10
99	199	1002	10
NCG is the number of CG iterations needed to find the best L_2 Finite Element fit to f(s).			

Table 3. Best monotonic data fitting.

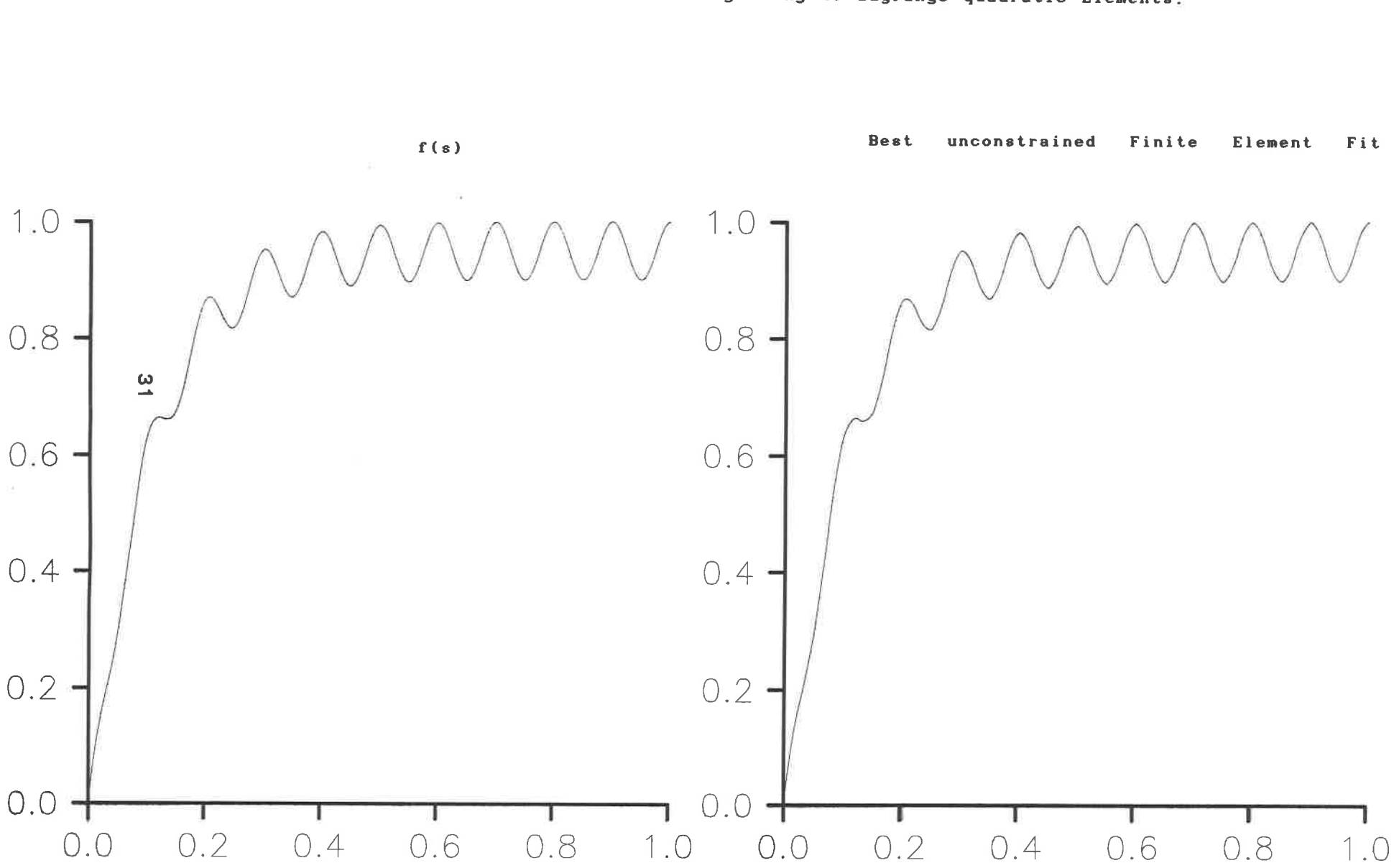

Finite Element data fitting using 40 Lagrange quadratic Elements.
f(s)
31
Best unconstrained Finite Element Fit
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Figure 5.

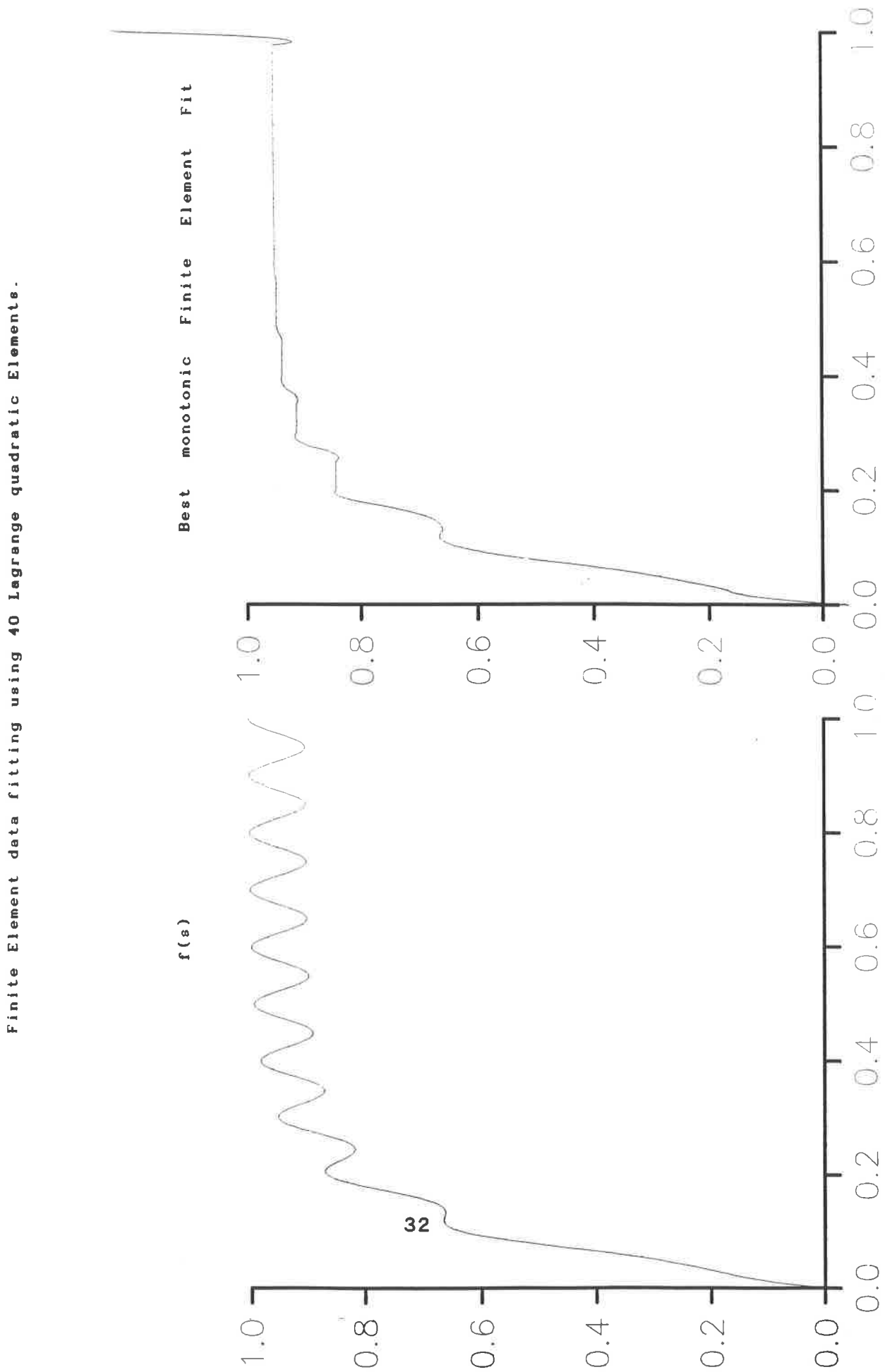

Finite Element data fitting using 40 Lagrange quadratic Elements.
Best monotonic Finite Element Fit
f(s)
32
0.0
0.2
0.4
0.6
0.8
1.0

Figure 6.

SECTION 6. APPLICATION TO MOVING GRID METHODS.

In this section we outline the application of constrained CG methods to moving grid Finite Element approximations, taking as an example the Moving Finite Element (MFE) method as described by Wathen, Baines, and Johnson [1,3,7]. We start with a very brief description of the MFE technique.

6.1 The MFE method.

Given a well-posed initial value problem

$$u_t = L(u), \quad \& \text{ boundary conditions} \qquad (6.1)$$
$$u(x,0) = u_0(x),$$

where L is an operator involving only spatial derivatives, the MFE method may be summarised, for a single space variable scalar problem, as follows. The solution u is approximated by the Finite Element spline

$$v(x,t) = \sum_1^n u_j(t)\ \phi_j(x,s_j(t)),$$

where the basis functions ϕ_j depend on the space variable x, and the nodal positions $s_j(t)$. At a fixed time t, the norm of the residual, given by

$$|| v_t - L(v) ||_2, \qquad (6.2)$$

is minimised with respect to the nodal amplitude derivatives $\dot{u}_j$ and the nodal position derivatives $\dot{s}_j$, leading to a set of ordinary differential equations

$$A(y)\dot{y} = g(y) \qquad (6.3)$$

where $y^T = (u_1,s_1,u_2,s_2,.....,u_n,s_n)$, and A(y) is the MFE mass

matrix. The equations (6.3) are then integrated forward in time by some time-stepping method. Problems occur in this procedure when either A becomes singular, or when the nodes s_j become disordered. Wathen and Baines [13] have shown that if (6.1) is a hyperbolic problem, then both of these difficulties can be effectively overcome using numerical techniques which detect the onset of the singularity of A and also recognise the presence of a developing shock or discontinuity. When (6.1) is a parabolic problem, other techniques have to be employed to stop the nodes "overtaking". This is because Wathen's method depends on replacing the differential equation by shock jump conditions as the nodes come together. There is no obvious analogue of this idea for parabolic problems. Instead, a number of ad hoc methods have been employed to stop the disordering of the nodes. One such method, proposed by Miller [9], has been to add penalty terms to the functional (6.2) in such a way as to prevent the nodes from coming too close together. This process results in equations (6.3) becoming extremely stiff, and necessitates the use of computationally expensive implicit integration schemes for their solution. Another approach used by Johnson et al. [7], and Moody [10], is to use the simple explicit Euler time stepping method on (6.3) and restrict the timestep Δt^m so that, given a set of nodal velocities $\dot{s}_j^m$, at time t^m, the new nodal positions

$$s_j^{m+1} = s_j^m + \Delta t^m \, \dot{s}_j^m \ ,$$

at time $t^m + \Delta t^m$ remain ordered. There is no guarantee with this method that the sequence of timesteps $\{ \Delta t^m \}$ will not have the

disastrous property

$$\Delta t^{m+1} \leq a^m \Delta t^m, \quad m=1,2,.....$$

where a is a constant such that $0 < a < 1$, thus making it impossible to integrate (6.3) on a fixed time interval [0,T] in the case that a is a small number. It is also noted that the time-step will in general be constrained by the "worst" part of the grid. Thus even this method will effectively make (6.3) stiff if there are very few nodes near the point of "overtaking".

6.2 A constrained nodal movement method.

An examination of the residual in (6.2) shows that if we constrain all the nodes to be fixed then the MFE method reduces to the usual Fixed Finite Element method [2]. Allowing complete freedom for the nodal positions, we arrive back at the MFE method. We now propose a method which is intermediate between these two extremes. For simplicity we examine the case of an Euler time-stepping scheme. That is, if the time derivative parameters $\dot{u}_j^m$, and $\dot{s}_j^m$ are given at time t^m, then the new nodal parameters u_j^{m+1}, s_j^{m+1} at time $t^{m+1} = t^m + \Delta t^m$ are given by

$$\begin{aligned} u_j^{m+1} &= u_j^m + \Delta t^m \dot{u}_j^m \\ s_j^{m+1} &= s_j^m + \Delta t^m \dot{s}_j^m \end{aligned} \qquad j = 1,2,...,n \ .$$

Thus each nodal position s_j^m is moved by a distance $\Delta t^m \dot{s}_j^m$ during the m^{th} time-step. The square of (6.2), at time t^m, when written in full is given by

$$\Psi(\dot{y}) = \dot{y}^T A(y)\dot{y} - 2\,\dot{y}^T g(y) + g(y)^T g(y) \qquad (6.4)$$

where $y = y^m$. We can thus apply algorithm 3 of section 4.1 to Ψ since we are seeking to minimise (6.4) with respect to the vector

of derivatives $\dot{y}$. This idea gives rise to the following method for preventing the nodes from overtaking.

Algorithm 4. At time t^m :

STEP 1 : Select any appropriate time-step Δt^m.

STEP 2 : for j = 2 to n - 1

$$ds_j^R = \frac{1}{2}(s_{j+1}^m - s_j^m) - \varepsilon/2$$

$$ds_j^L = \frac{1}{2}(s_j^m - s_{j-1}^m) - \varepsilon/2$$

next j

$$ds_1^L = ds_1^R = ds_n^L = ds_n^R = 0$$

STEP 3 : minimise $\Psi(\dot{y}^m)$ subject to the constraints

$$-\frac{ds^L}{\Delta t^m} \leq \dot{s}_j^m \leq \frac{ds^R}{\Delta t^m}$$

STEP 4 : for j = 1 to n

$$u_j^{m+1} = u_j^m + \Delta t^m \dot{u}_j^m$$

$$s_j^{m+1} = s_j^m + \Delta t^m \dot{s}_j^m$$

next j

$$t^{m+1} = t^m + \Delta t^m$$

$$m := m + 1$$

STEP 5 : If $t^m < T$ then go to STEP 1

In STEP 1 the time-step Δt^m may be selected on the basis of accuracy, stability, or some other criterion. The constrained minimisation in STEP 3 finds the smallest value of the residual (6.3) such that, in the worst case, the nodes get no closer than a user defined tolerance ε apart (fig. 7).

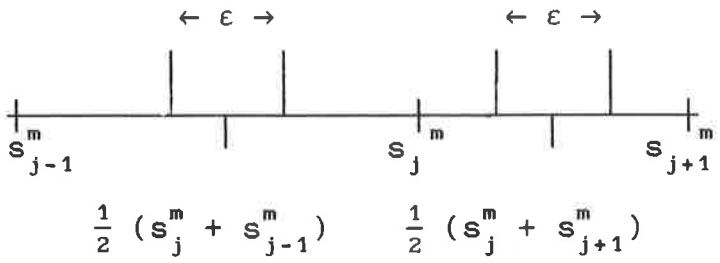


Figure 7.

Nodes 1 and n are assumed fixed here, though the method may be easily modified to cause any of the n nodes to remain fixed, or indeed to allow the end nodes to move if the boundary is in motion. In the case of (6.1) being a hyperbolic problem we may choose $\varepsilon = 0$. The useful properties of the method described can be summarised as follows :

(i) The time-step Δt^m can be selected without regard to the motion of the nodes.

(ii) Any of the nodes can be fixed at any point during the time integration.

(iii) There are no penalty terms, so the equations (6.3) can never get any stiffer.

(iv) There are very obvious generalisations to problems with more than one space variable.

(v) The method is suitable for moving boundary problems where the velocity of the boundary nodes is prescribed.

(vi) In the case that either all nodes are fixed or all nodes are free then the minimisation routine reduces to the ordinary CG method and hence the approximations reduce to either the FFE

method or the MFE method.

Of course, we still have to show that the constrained minimisation procedure, which has to be done at each time-step, is an efficient and practical method. However we note that the matrix A(y), describing the quadratic form in Ψ is symmetric, positive definite and well conditioned in most cases. If it is also remembered that the minimisation occurs in a time-stepping scheme, then we see that the solution from the last time-step can be used as a good starting value for the next minimisation. Preliminary results of using Algorithm 4 applied to a diffusion problem are so far encouraging and it is hoped that a report on this work will appear in the near future. A typical set of solution curves for a 1-D diffusion problem using a constant time step are given in figures 8,9.

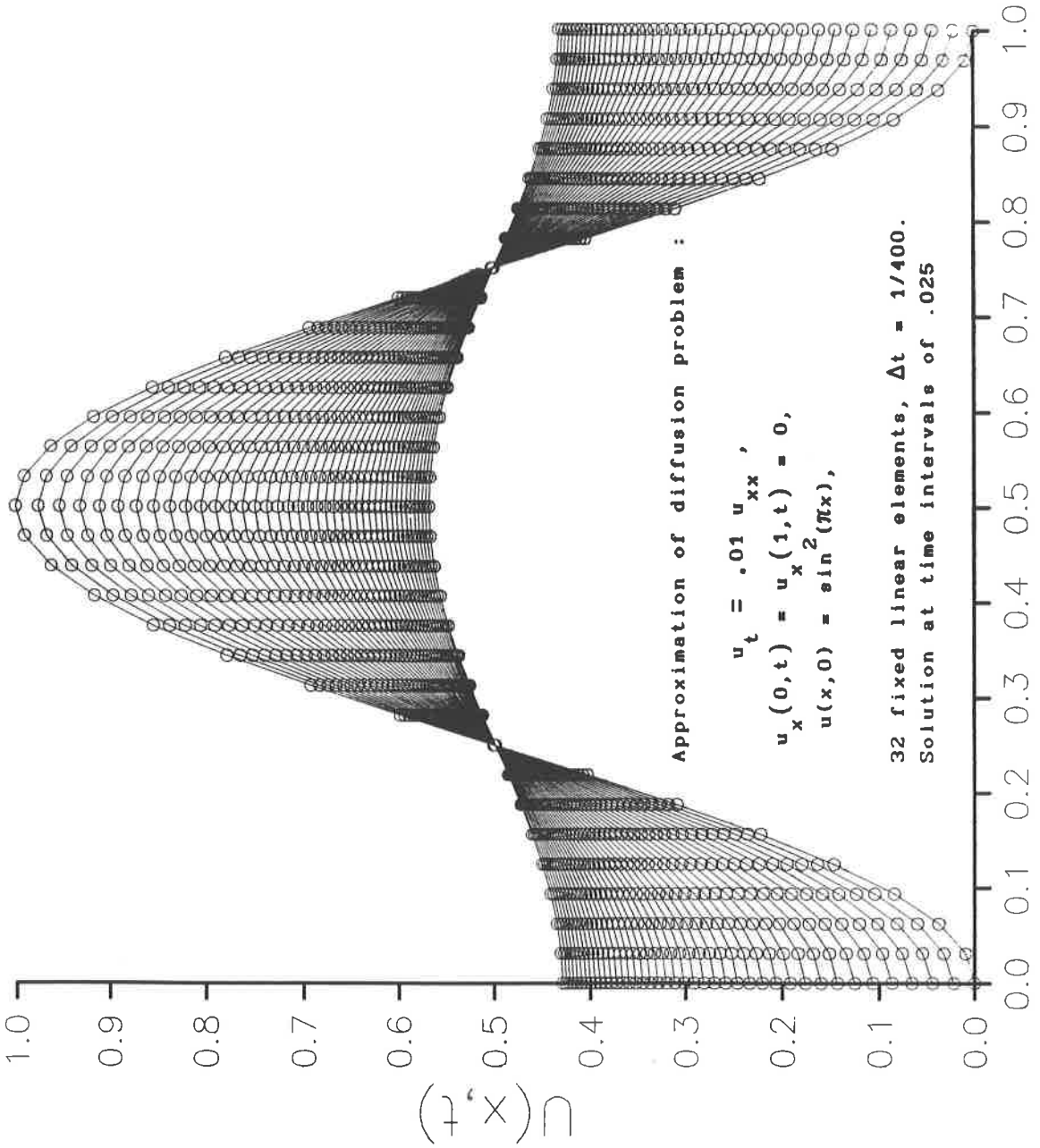

Approximation of diffusion problem :
$u_t = .01\, u_{xx}$,
$u_x(0,t) = u_x(1,t) = 0$,
$u(x,0) = \sin^2(\pi x)$,
32 fixed linear elements, $\Delta t = 1/400$.
Solution at time intervals of .025
U(X,t)
X
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

Figure 8.

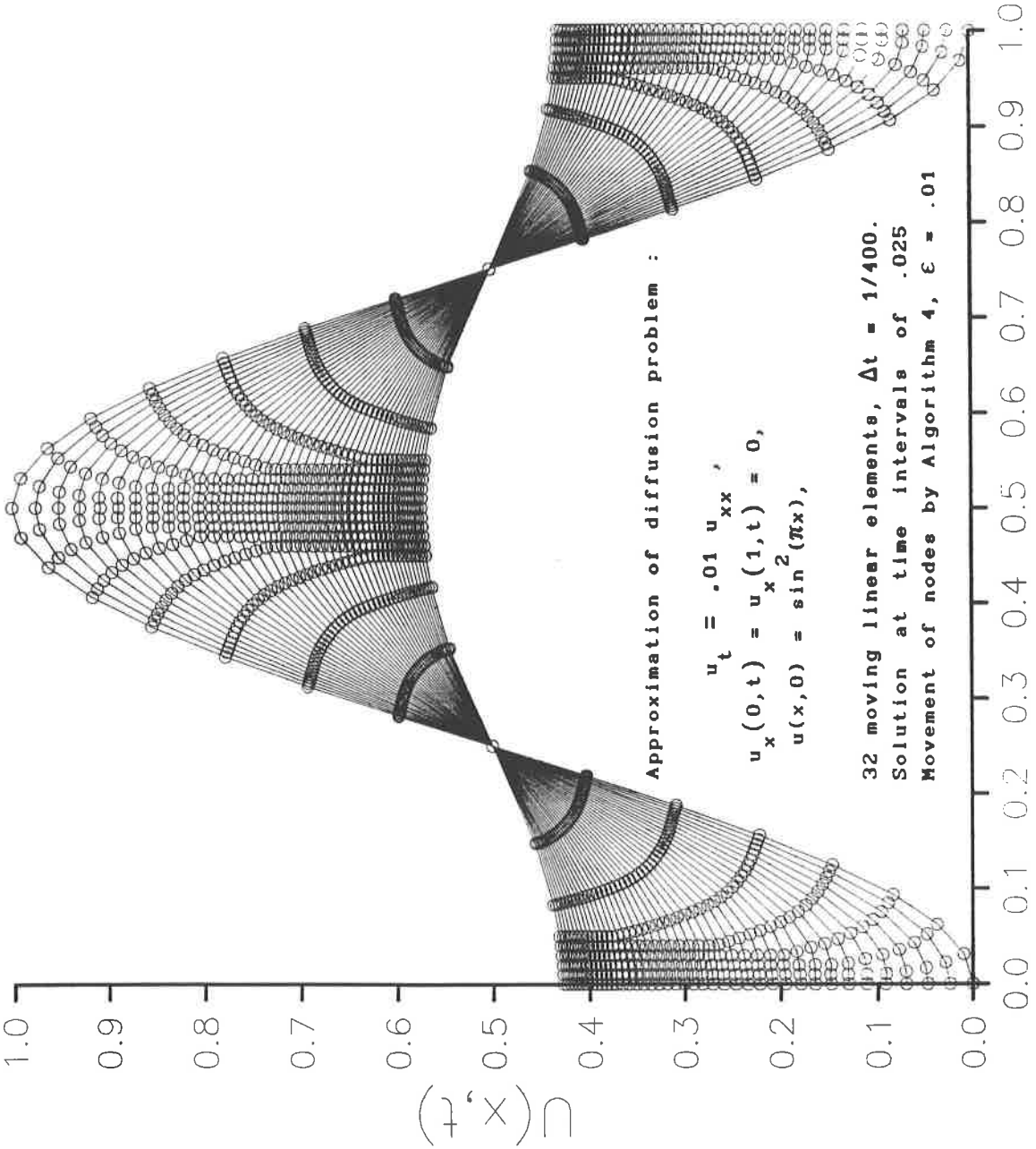

Approximation of diffusion problem :
$u_t = .01\, u_{xx}$,
$u_x(0,t) = u_x(1,t) = 0$,
$u(x,0) = \sin^2(\pi x)$,
32 moving linear elements, $\Delta t = 1/400$.
Solution at time intervals of .025
Movement of nodes by Algorithm 4, $\epsilon = .01$
$U(x,t)$
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
1.0 0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1 0.0

Figure 9.

SECTION 7. CONCLUSION.

Analogy is a useful concept in Numerical Analysis. We replace a stable system of initial value problems by a stable system of difference problems; an integral equation with symmetric kernel by a symmetric system of algebraic equations. The finite dimensional analogy of a variational problem is thus a problem in optimisation; an idea that is apt to be overlooked as we plunge headlong into solving an approximation to the Euler equations for a given variational problem. Taking a "step back" to the original problem of minimising an integral leads us to questions of how to minimise an approximate integral, rather than trying to solve everything with a linear algebra package. A rather ironic result of this report is that we *have* applied a "linear solver" to several awkward problems with some success. The key reason for this success is that the CG method is a natural extension of the "gradient method", a well known but inefficient procedure for minimising functionals. A simple strategy has been employed here, namely : apply the CG method as much as possible, otherwise take a step of the gradient method.

It has been shown that, for many examples, the algorithms described are effective, particularly when the symmetric matrix A describing the quadratic form is well conditioned. One rather glaring omission is that we have performed no analysis to indicate how many operations our procedures require. The only defence is to say that this work is in hand and that if no one ever used a method until they knew precisely how many "flops" are to be

incurred, then Newton's method for solving non-linear algebraic equations would be the least, instead of most widely, used method. The analogy between Newton's method and the methods described in this report is not drawn at random, since in all cases it is possible to say conclusively whether or not the given method has solved the given problem.

Although the applications discussed here are rather simple, they are a radical extension of (and hence include) the problem of finding unconstrained stationary points of functionals, using methods that are particularly suitable to the Finite Element approach. It is hoped that the ideas presented here will have many more applications.

REFERENCES

[1] BAINES, M.J. (1985) On Approximate Solutions of Time-dependent P.D.E's by The Moving Finite Element Method. Num. Anal. Rpt. 1/85, Dept. of Mathematics, University of Reading.

[2] BAINES, M.J. (1985) Projections and Constraints. Num. Anal. Rpt. 15/85, Dept. of Mathematics, University of Reading.

[3] BAINES, M.J., and WATHEN, A.J. (1988) Moving Finite Element Methods for Evolutionary Problems. I. Theory.J. Computational Phy., 79, No. 2, 245-269

[4] BAIOCCHI, C., and CAPELO, C. (1984) Variational and Quasivariational Inequalities. (Translation from the original Italian) The Universities Press, Belfast.

[5] FRIED, I. (1979) Numerical Solution of Differential Equations. Academic Press, New York.

[6] GOLUB, G.H., and VAN LOAN, C.F. (1985) Matrix Computations. John Hopkins Press, Baltimore.

[7] JOHNSON, I.W., WATHEN, A.J., and BAINES, M.J., (1988) Moving Finite Element Methods for Evolutionary Problems. II. Applications. J. Computational Phys. 79, No. 2, 270 -297.

[8] KUHN, H.W, and TUCKER, A.W. (1951) Nonlinear Programming. Proc. 2nd Berkley Symp. on Math.Stat., ed J. Neyman. Univ. Cal. Press

[9] MILLER, K. (1981) Moving Finite Elements, Part II. SIAM J. Numer. Anal. 18, 1033-1057.

[10] MOODY, R.O. (1988) Ph.D. Thesis, University of Reading.

[11] SEWELL, M.J. (1987) Maximum and Minimum Principles. Cambridge University Press.

[12] STRANG, G., and FIX, G.J. (1973) An Analysis of the Finite Element Method. Prentice-Hall, N.J.

[13] WATHEN, A.J., and BAINES, A.J., (1983) On the Structure of the Moving Finite Element Equations. IMA J. of NUM. Anal. 5, 161-182.

[14] ZIENKIEWICZ, O.C. (1971) The Finite Element Method in Engineering Science. McGraw-Hill, London.