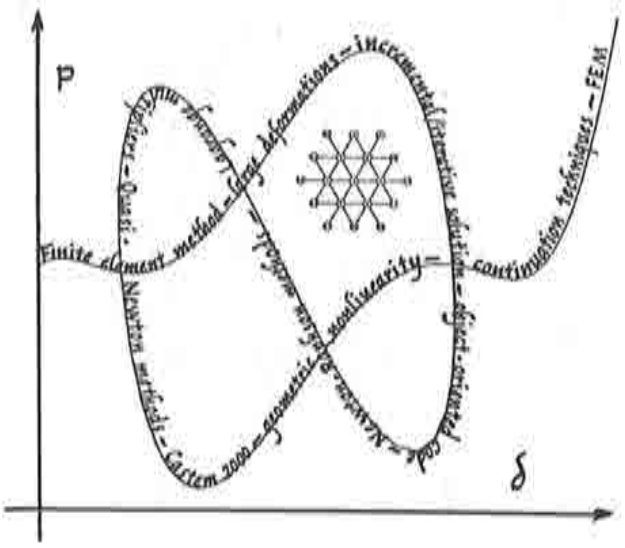
Nonlinear Finite Element Techniques Using an Object-Oriented Code

A. Vila
A. Rodríguez-Ferran
A. Huerta



Monograph CIMNE Nº 31, October 1995

Nonlinear Finite Element Techniques Using an Object-Oriented Code

Agnès Vila
Antonio Rodríguez-Ferran
Antonio Huerta

E.T.S. d'Enginyers de Camins, Canals i Ports
Universitat Politècnica de Catalunya

Monograph CIMNE Nº 31, October 1995

Internacional Center for Numerical Methods in Engineering
Gran Capitán s/n, 08034 Barcelona, España

The cover designed by: Jordi Pallí

First published, October 1995

Edited by:
International Center for Numerical Methods in Engineering
C/ Gran Capitán, s/n
08034 Barcelona, Spain

ISBN: 84-87867-64-2
Deposito Legal: B-41308-95

Contents

Summary

When treating nonlinear problems with the Finite Element formulation, the need to solve nonlinear sets of equations arises. Classically, those nonlinear sets of equations were solved by performing an incremental/iterative analysis with a full Newton–Raphson method.

Today, one can choose from a wide range of methods. The modifications to the original fNR method use tangent stiffness matrices, which are updated at most once per increment. Quasi–Newton methods use secant matrices instead of tangent matrices. Secant–Newton methods are simplifications of the Quasi–Newtons.

This work deals, in particular, with the implementation of these methods in an object-oriented code and with the comparison between them on several tests. The expected behaviour of the methods is observed from the results.

Also, an acceleration line-search technique is implemented to improve the results supplied by the previous methods.

To solve problems with limit points, arc-length techniques are employed to follow the equilibrium paths. The programmed procedures are used to solve

several snap-through and snap-back benchmark tests.

The computer code, Castem2000, is an object-oriented code which employs the Lagrange multiplier technique to impose boundary conditions. This feature leads to more than one possible approach to the Quasi–Newton and Secant–Newton methods. In particular, some specific techniques have been developed in this work to treat nonlinear problems with linear constraints.

Chapter 1

An introduction to nonlinearity

An exhaustive description of the Finite Element formulation for linear problems can be found in [20,30]. As for nonlinear analysis, [1,10] are obliged references.

In Section 1.1, two simple examples are firstly used to introduce nonlinearity. In Section 1.2, some basic notions on the FEM for linear and nonlinear problems are briefly reviewed.

1.1 Motivation: two simple examples of geometric and material nonlinearity

Figure 1.1 shows a bar of area A that is subject to a load F at one end and is fixed at the other. The bar is made of a plastic material with a bilinear strain-stress law, Figure 1.2.

Figure 1.1: First example: Plastic bar subject to axial load.

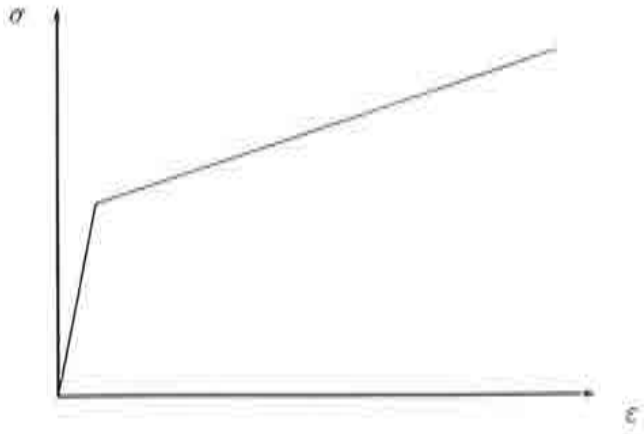


Figure 1.2: Strain-stress law for the bar in Figure 1.1.

If N is the axial force in the bar, then, for equilibrium

$$N = F. \tag{1.1}$$

Since $N = A\,\sigma$, we have

$$F = A\,\sigma. \tag{1.2}$$

If we denote by δ the increment in the length of the bar, the value of the strain ε can be computed as

$$\varepsilon = \frac{\delta}{l}. \tag{1.3}$$

Summing up: the force F is proportional to σ, equation (1.2), and σ is a nonlinear function of $\varepsilon = \delta/l$, see Figure 1.2. Therefore, F is a nonlinear function of the displacement δ; actually, the $F - \delta$ curve is identical to the $\sigma - \varepsilon$ law except for a scale factor.

This case is an example of *material nonlinearity*.

We now consider the elastic bar in Figure 1.3, [10]. This bar has area A and constant Young's modulus E.

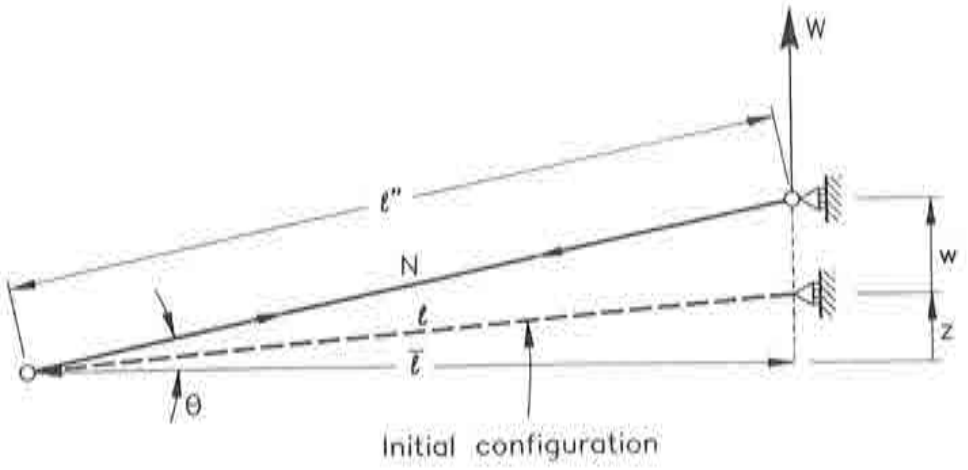


Figure 1.3: Second example: elastic bar subject to vertical load.

When a vertical load W is applied, the axial force in the bar can be computed after the equilibrium equation

$$W = N \ \sin\theta. \tag{1.4}$$

If we assume that θ is small, the following approximation can be made:

$$\sin\theta \approx \tan\theta = \frac{z+w}{l}. \tag{1.5}$$

Equation (1.5) can now be substituted in (1.4) yielding

$$W = N \ \frac{z+w}{l}. \tag{1.6}$$

Using now that $N = A\ \sigma = E\ A\ \varepsilon$, equation (1.6) becomes

$$W = E\ A\ \frac{z+w}{l}\ \varepsilon. \tag{1.7}$$

Since $l'' - l$ is the increment of length in the bar associated to the displacement w, the strain ε is computed as

$$\varepsilon = \frac{\delta}{l} = \frac{l''-l}{l} = \frac{\sqrt{(z+w)^2+l^2} - \sqrt{z^2+l^2}}{\sqrt{z^2+l^2}}. \tag{1.8}$$

Equation (1.8) can be manipulated, [10], to yield the approximation

$$\varepsilon \approx \left(\frac{z}{l}\right)\left(\frac{w}{l}\right) + \ \frac{1}{2}\ \left(\frac{w}{l}\right)^2. \tag{1.9}$$

Finally, substituting (1.9) into (1.7) gives

$$W = \frac{E\ A}{l^3}\ \left(z^2\ w + \frac{3}{2}\ z\ w^2 + \frac{1}{2}\ w^3\right). \tag{1.10}$$

Thus, although E is constant, the load W is expressed from (1.10) as a nonlinear function of the displacement w.

This is a case of *geometric nonlinearity*.

1.2 Finite element formulation of the problem

In Section 1.1 two examples of nonlinear problems have been presented.

In this section, we begin by stating in general terms the problem we want to solve. We have a certain body Ω (essentially a solid or a structure, in our case) which is subject to some *external forces* $\boldsymbol{f}_e$ and also to some *boundary conditions*. After a Finite Element formulation, the body Ω is divided into several *elements* and the basic unknown $\boldsymbol{d}$ (continuous field of displacements) is transformed into a certain vector $\boldsymbol{u}$ of *nodal displacements*

$$\boldsymbol{d} = \boldsymbol{N}\,\boldsymbol{u}, \tag{1.11}$$

where $\boldsymbol{N}$ is a matrix of *shape functions*.

After equation (1.11), the vector of strains, $\boldsymbol{\varepsilon}$, can be obtained as

$$\boldsymbol{\varepsilon} = \boldsymbol{B}\,\boldsymbol{u}, \tag{1.12}$$

where $\boldsymbol{B}$ is the *deformation matrix*, which *depends on the displacements* in a general case.

As for the vector of stresses $\boldsymbol{\sigma}$, it can also be expressed in terms of previously used variables. For instance, in a *linear* problem, $\boldsymbol{\sigma}$ can be put directly in terms of $\boldsymbol{\varepsilon}$ through a constant matrix $\boldsymbol{D}$, called the *constitutive matrix*

$$\boldsymbol{\sigma} = \boldsymbol{D}\,\boldsymbol{\varepsilon}. \tag{1.13}$$

For materially nonlinear cases, stresses will have to be obtained by integrating the *constitutive equation*.

Equilibrium — the principle of virtual work

So far we have described briefly the problem we want to solve. In this section, we are going to complete the formalization of that problem. That is, we are going to write the specific equations that govern it.

Let us consider a single element from the discretization, which will be denoted by (e). If some *virtual* displacements $\boldsymbol{\delta u}^{(e)}$ are imposed on this element, then after the *Principle of virtual work*, [20,30], we have that

The element is in equilibrium if the virtual work undertaken by external forces equals the work undertaken by internal forces.

Considering that the latter is actually the work produced by stresses over the strains that result from virtual displacements, the principle of virtual work can be written as follows:

Element (e) *is in equilibrium if*

$$V = \int_{\Omega^{(e)}} \boldsymbol{\delta\varepsilon}^T \boldsymbol{\sigma}\, dV - \left(\boldsymbol{\delta u}^{(e)}\right)^T \boldsymbol{f}_e^{(e)} = 0, \tag{1.14}$$

for any vector $\boldsymbol{\delta u}^{(e)}$ *of virtual nodal displacements, where* $\boldsymbol{\delta\varepsilon}$ *is the vector that contains the corresponding virtual strains,* $\boldsymbol{\sigma}$ *is the stress vector and* $\boldsymbol{f}_e^{(e)}$ *includes all external nodal forces applied to the element (including reaction forces of neighbour elements).*

After a Finite Element formulation, equation (1.14) is rewritten as

$$V = \int_{\Omega^{(e)}} (\boldsymbol{\delta u}^{(e)})^T \boldsymbol{B}^T \boldsymbol{\sigma}\, dV - (\boldsymbol{\delta u}^{(e)})^T \boldsymbol{f}_e^{(e)} = (\boldsymbol{\delta u}^{(e)})^T \left\{ \int_{\Omega^{(e)}} \boldsymbol{B}^T \boldsymbol{\sigma}\, dV - \boldsymbol{f}_e^{(e)} \right\} = 0 \tag{1.15}$$

for any $\boldsymbol{\delta u}^{(e)}$. This means that the term $\boldsymbol{\delta u}^{(e)}$ in equation (1.15) can be dropped and the equilibrium equation finally reads

$$\int_{\Omega^{(e)}} \boldsymbol{B}^T \boldsymbol{\sigma}\, dV - \boldsymbol{f}_e^{(e)} = \boldsymbol{f}_i^{(e)} - \boldsymbol{f}_e^{(e)} = \boldsymbol{r}^{(e)} = \boldsymbol{0}, \tag{1.16}$$

where $\boldsymbol{f}_i^{(e)}$ is the vector of internal nodal forces and $\boldsymbol{r}^{(e)}$ is consequently the vector of *out-of-balance forces*, both of them associated to element (e).

Up to this point, we have dealt with a single element (e). To obtain an equilibrium equation for the whole body, an assembly process must be performed over all the elements, [20,30]. This results in the equation

$$\int_{\Omega} \boldsymbol{B}^T \, \boldsymbol{\sigma} \, dV - \boldsymbol{f}_e = \boldsymbol{f}_i - \boldsymbol{f}_e = \boldsymbol{r} = \boldsymbol{0}, \tag{1.17}$$

where $\boldsymbol{f}_i$ and $\boldsymbol{f}_e$ represent respectively the entire vectors of internal and external forces and it must be noted that vector $\boldsymbol{B}^T \, \boldsymbol{\sigma}$ is now integrated over the whole body Ω.

A brief recall of the linear case. The linear stiffness matrix

Equation (1.17) expresses equilibrium for a solid under the application of a certain set of forces. This equation is general in the sense that it holds whether the problem is linear or not.

To recover the linear case, we need to substitute in equation (1.17) some of the linear FEM relationships that have already been introduced.

We saw at the beginning of this section that the displacement vector $\boldsymbol{u}$ and the strain vector $\boldsymbol{\varepsilon}$ can be connected by the expression $\boldsymbol{\varepsilon} = \boldsymbol{B} \, \boldsymbol{u}$, where $\boldsymbol{B}$ is the *deformation matrix*, see equation (1.12).

If we make the hypothesis that the problem is *linear*, we will consequently have that

a) Strains can be written as a linear function of the displacements.

b) Stresses can be expressed as a linear function of the strains.

Condition b) allows us to write $\boldsymbol{\sigma} = \boldsymbol{D} \, \boldsymbol{\varepsilon}$, where $\boldsymbol{D}$ is the *constitutive matrix* presented in equation (1.13).

Thus, by a) we have that $\boldsymbol{B}$ does not depend on the displacements; and, by a) and b), that $\boldsymbol{D}$ does not depend on $\boldsymbol{u}$, either.

We can now use equations (1.12) and (1.13) to express stresses directly in terms of the displacements

$$\boldsymbol{\sigma} = \boldsymbol{D}\,\boldsymbol{\varepsilon} = \boldsymbol{D}\,\boldsymbol{B}\,\boldsymbol{u}, \tag{1.18}$$

and then substitute (1.18) into (1.17) obtaining

$$\int_\Omega \boldsymbol{B}^T\,\boldsymbol{\sigma}\,dV - \boldsymbol{f}_e = \int_\Omega \boldsymbol{B}^T\,\boldsymbol{D}\,\boldsymbol{B}\,\boldsymbol{u}\,dV - \boldsymbol{f}_e = \left\{\int_\Omega \boldsymbol{B}^T\,\boldsymbol{D}\,\boldsymbol{B}\,dV\right\}\boldsymbol{u} - \boldsymbol{f}_e = \boldsymbol{0}. \tag{1.19}$$

Let us focus now on the integral term in (1.19).

As $\boldsymbol{B}$ and $\boldsymbol{D}$ are constant, the matrix $\boldsymbol{K}_{\text{elas}}$ defined by

$$\boldsymbol{K}_{\text{elas}} = \int_\Omega \boldsymbol{B}^T\,\boldsymbol{D}\,\boldsymbol{B}\,dV \tag{1.20}$$

must be constant for a given linear problem. This matrix $\boldsymbol{K}_{\text{elas}}$ is precisely the *linear stiffness matrix* $\boldsymbol{K}_{\text{elas}}$ used in linear analysis.

Finally, using the notation established in equation (1.20), equation (1.19) can be rewritten as

$$\boldsymbol{K}_{\text{elas}}\,\boldsymbol{u} - \boldsymbol{f}_e = \boldsymbol{0}$$

or

$$\boldsymbol{K}_{\text{elas}}\,\boldsymbol{u} = \boldsymbol{f}_e, \tag{1.21}$$

which, under the assumption of conservative forces (*i. e.*, forces which do not depend on the displacements) represents a *set of linear equations*.

Hence, when performing a linear analysis, the general mechanical problem reduces to finding a vector $\boldsymbol{u}$ of nodal displacements such that

1.- satisfies the boundary conditions

2.- verifies the set of linear equations $\boldsymbol{K}_{\text{elas}}\,\boldsymbol{u} = \boldsymbol{f}_e$.

It is important to remark that conditions **1.-** and **2.-** cannot be treated separately. As a matter of fact, the stiffness matrix $\boldsymbol{K}_{\text{elas}}$ defined in (1.20) is a *singular* matrix, [20,30]. Consequently, an infinite family of solutions could be found for (1.21). If the problem is well-posed, only one of these solutions satisfies the prescribed boundary conditions. Two possible options to obtain directly the solution of the problem defined by **1.-** and **2.-** are discussed in Section 3.5.

Hence, each time that an instruction like

Solve <set of linear equations>

appears in the text, we will assume that some method has been adopted to treat boundary conditions so that the unique solution of the whole problem can be computed.

When dealing with a *nonlinear* problem, the vector of residual forces is written as

$$\boldsymbol{r}(\boldsymbol{u}) = \boldsymbol{f}_i(\boldsymbol{u}) - \boldsymbol{f}_e, \tag{1.22}$$

where the external forces $\boldsymbol{f}_e$ are conservative and internal forces have become dependent on the displacements.

The problem of finding $\boldsymbol{u}$ for which

$$\boldsymbol{r}(\boldsymbol{u}) = \boldsymbol{0} \tag{1.23}$$

passes through the definition of a *tangent stiffness matrix*

$$\boldsymbol{K} = \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{u}} = \frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{u}},$$

which is the equivalent of the linear stiffness matrix $\boldsymbol{K}_{\text{elas}}$ for the linear case (see Section 3.1).

We can use the description of the internal forces

$$\boldsymbol{f}_i = \int_\Omega \boldsymbol{B}^T \boldsymbol{\sigma} \, dV, \tag{1.24}$$

which yields the following expression of $\boldsymbol{K}$

$$\boldsymbol{K} = \frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{u}} = \int_\Omega \frac{\partial \boldsymbol{B}^T}{\partial \boldsymbol{u}} \boldsymbol{\sigma} \, dV + \int_\Omega \boldsymbol{B}^T \frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{u}} \, dV. \tag{1.25}$$

Equation (1.25) is of a special interest because the two possible causes of nonlinearity can be observed in it:

1) In a *geometrically* nonlinear problem, matrix $\boldsymbol{B}$ is not constant, and thus the term $\partial \boldsymbol{B}^T / \partial \boldsymbol{u}$ is nonzero.

2) In a *materially* nonlinear problem, $\boldsymbol{\sigma}$ is a nonlinear function of $\boldsymbol{\varepsilon}$. That means, in particular, that $\partial \boldsymbol{\sigma} / \partial \boldsymbol{\varepsilon}$ is not constant. Therefore, decomposing $\partial \boldsymbol{\sigma} / \partial \boldsymbol{u}$ as

$$\partial \boldsymbol{\sigma} / \partial \boldsymbol{u} = \frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{\varepsilon}} \frac{\partial \boldsymbol{\varepsilon}}{\partial \boldsymbol{u}}$$

it is easy to see that $\partial \boldsymbol{\sigma} / \partial \boldsymbol{u}$ cannot be a constant matrix, either.

Chapter 2

Getting acquainted with the programming environment

2.1 Why an Object-Oriented Code?

Object-oriented codes are based on the use of objects of a complex nature. This complex nature is basically due to the fact that more information about the object's structure is saved. This results in an easier handling of the information.

Castem2000, [3,5,28] is an object-oriented finite element code. Thus, it handles objects such as points, meshes, stiffness matrices, stress fields, etc.

The solving of a problem is divided into several elemental processes; a basic instruction line looks like

```
new_object = OPERATOR old_object_1 old_object_2 ...
```

where an operator manipulates one or more old objects to form a new one.

The process typically begins with the building of some simple geometrical objects. These objects are manipulated to form meshes, to which a model is associated; the process goes on to define the loads, the computing options, etc.

As users, we need not worry about the complexity of the operations or of the created objects (for instance, a "solve" instruction or a "displacement field" object may turn up to be very complicated items).

Data and results may be studied mainly by drawing or listing; also, components of vectors can be extracted, maximum values can be computed, etc.

Furthermore, the code can be extended to incorporate operators that accomplish specific purposes.

In all, object oriented codes, Castem2000 in particular, supply a very efficient and pedagogical tool to solve structural problems.

2.2 Solving linear problems with Castem2000

In this section, an example of a program that solves a simple linear mechanical problem is presented. We will be following the process line by line to see how easily and naturally objects are being made up and also how close the whole process is to what we would do if we attempted to solve the problem manually.

The example we have chosen is the axisymmetrical shell that is represented in Figure 2.1. This shell is subject to a point load, an internal pressure and its self-weight.

In the following program, see Table 2.1, we intend to compute the displacements of all the nodes of the structure; in particular, we will extract the radial displacement of the central node and compare it to its theoretical value $u_r = 4.677\ \mu$m. Some comments have also been introduced in the file (those lines beginning with "*" are interpreted by Castem as commentary lines).

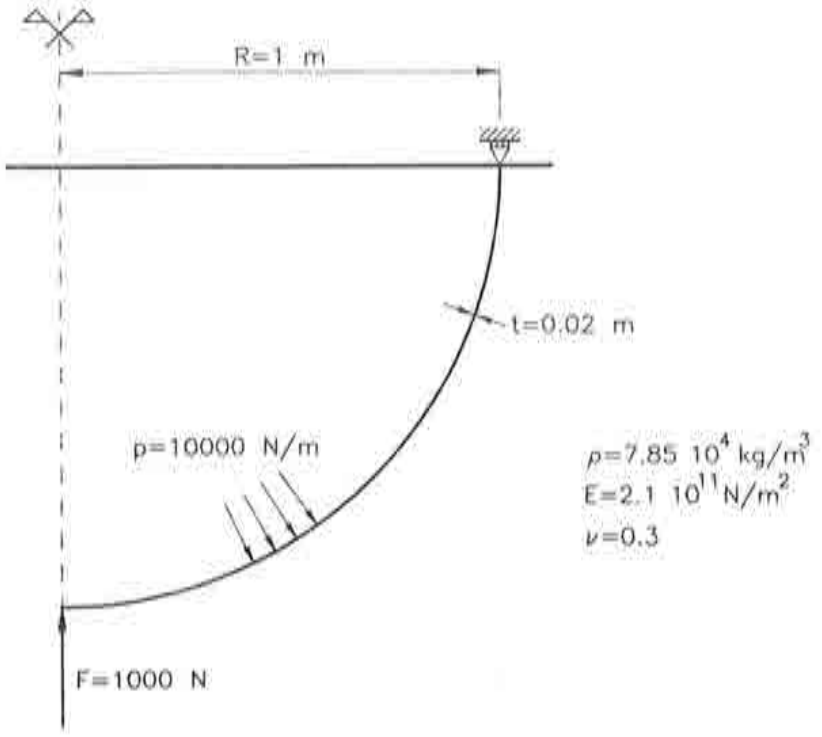


Figure 2.1: Linear example — axisymmetrical shell.

```
* A spherical shell is subject to :
* - an internal pressure
* - its self-weight
* - a point load at the centre
* Assuming that the problem is elastic and linear, we intend to
* compute the radial displacement of the central point of the shell.
* The theoretical value of the radial displacement
* of the central point of the shell is 4.677 micrometres.
*
*
* First of all, we define the general computational options:
* Axisymmetrical rotation of a
* bidimensional problem where linear two-node elements are being used
*
  OPTION DIMENSION 2 ELEMENT SEG2 MODE AXIS ;
*
* ------------------ GEOMETRY ----------------------------------------
*
* We need a center point and two additional points
* to define a circular arc
  o = 0 1 ; a = 0 0 ; b = 1 1 ;
* We now split this arc into 100 linear elements
  l1 = CERCLE 100 a o b ;
*
* ------------------ BOUNDARY CONDITIONS -----------------------------
*
* We want the border of the shell not to move in the vertical direction.
* As for point a, we must impose the proper conditions
* to correctly simulate axisymmetry
  bc1 = BLOQUER b UZ ;
  bc2 = BLOQUER a UR ;
* Boundary conditions bc1 and bc2 are put together in a new object bc
* via operator ET.
  bc = bc1 ET bc2 ;
*
* ------------------ MODEL AND MATERIAL ------------------------------
*
* An elastic model is associated to the basic mesh with operator MODL.
* COQ2 is the type of element that results from rotating a SEG2
* around the symmetry axis.
  mo = MODL l1 MECANIQUE ELASTIQUE COQ2 ;
* The material is initially defined with operator MATR
* by its Young modulus E, its Poisson coefficient nu and its density rho
* Finally, a value for the thickness is included with operator CARB
  ma = MATR mo YOUN 2.1E11 NU 0.3 RHO 7.85E4 ;
  ca = CARB mo EPAI 0.02 ;
  ma = ma ET ca ;
```

Table 2.1: Input file for the shell example.

```
* ------------------ STIFFNESS MATRIX ---------------------------------
*
* The internal stiffness sti1 is computed first with operator RIGIDITE.
* Then, boundary conditions are incorporated to form,
* again with operator ET, the global stiffness matrix sti2
  sti1 = RIGIDITE mo ma ;
  sti2 = sti1 ET bc ;
*
* ------------------ FORCES -------------------------------------------
*
*
* ------------------ UNIFORM INTERNAL PRESSURE ------------------------
*
* A uniform pressure is applied to the model with operator PRES
  fo1 = PRES COQU mo -10000 NORM ;
*
* ------------------ SELF-WEIGHT --------------------------------------
*
* The force fo2 associated to self-weight is obtained by computing
* the mass matrix first with operator MASSE
* and then multiplying it by a unit nodal field
  mas = MASSE mo ma ;
  el1 = CHANGER li POI1 ;
  pop = MANUEL CHPO el1 1 UZ -1 ;
  fo2 = mas*pop ;
*
* ------------------ RADIAL FORCE AT CENTRAL POINT --------------------
*
* a vertical force at point b is introduced via operator FORCE
  fo3 = FORCE FR 1000 b ;
*
* Finally, all the forces are joined with operator ET
  fo = fo1 ET fo2 ET fo3 ;
*
* ------------------ COMPUTATION OF DISPLACEMENTS ---------------------
*
* The linear set of equations defined by matrix sti2 and vector fo
* is solved via operator RESOUDRE
  re = RESOUDRE sti2 fo ;
* The radial displacement of point b is obtained by extracting the
* corresponding component of the solution vector re
  rd = EXTRAIRE re UR b ;
* To end the execution of the program, we print the obtained results
  rd = 1000000*rd ;
  MESS 'RADIAL DISPLACEMENT AT CENTRAL POINT (UR)' ;
  MESS 'THEORETICAL: 4.677 MICROMETRE' ;
  MESS 'COMPUTED: ' rd 'MICROMETRE' ;
*
```

Table 2.1: Input file for the shell example (continued).

Running the program listed in Table 2.1 causes Castem to calculate the vector of nodal displacements of the shell and print the computed value for the radial displacement of its centre point.

To end this section and now that we have become familiarized with the original objective of the test, we present Castem's final answers, see Table 2.2.

```
RADIAL DISPLACEMENT AT CENTRE (UR)
THEORETICAL: 4.677 MICROMETRE
COMPUTED: 4.6685 MICROMETRE
```

Table 2.2: Castem's answers to the input file in Table 2.1.

The result supplied by Castem ($u_r = 4.6685\ \mu$m) represents less than a 0.2% error, which is considered fully satisfactory.

Chapter 3

A classical approach to nonlinear problems: Newton–Raphson methods

3.1 An incremental solution. The tangent stiffness matrix

As detailed in Section 1.2, the general equilibrium equation

$$\int_{\Omega} \boldsymbol{B}^T \, \boldsymbol{\sigma} \, dV - \boldsymbol{f}_e = \boldsymbol{f}_i - \boldsymbol{f}_e = \boldsymbol{r} = \boldsymbol{0} \tag{3.1}$$

leads, in the case of a linear problem, to a certain set of linear equations.

If the problem does not happen to be linear, we will not be able to write $\boldsymbol{B}^T \, \boldsymbol{\sigma}$

as a linear function of $\boldsymbol{u}$, as we did in Section 1.2., and equation (3.1) will yield a set of *non*linear equations. Furthermore, this set of nonlinear equations needs to be treated differently than a linear set.

To illustrate this point, let us go back to the linear problem.

Figure 3.1 shows several items associated to this problem. It must be remarked that, in this figure, as well as in all the figures of this kind that appear throughout this work, both forces and displacements are treated, for the sake of visual interpretation, as if the problem had just one degree of freedom. In the general case, scalars will have to be seen as vectors, lines as hyperplanes, and so on.

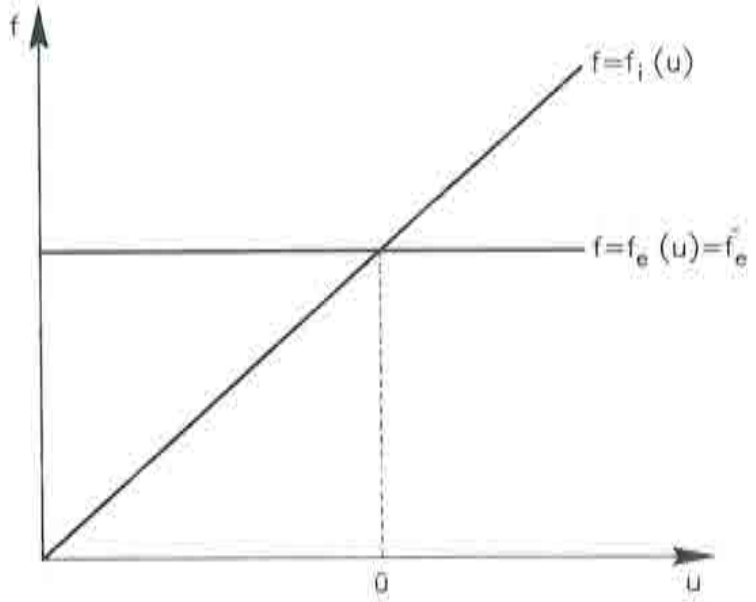


Figure 3.1: Solution of the linear problem for a fixed external force.

The vector of external forces $\boldsymbol{f}_e$ is assumed to be independent of the displacements. Thus, in a force–displacement plane, the curve $\boldsymbol{f} = \boldsymbol{f}_e(\boldsymbol{u})$ is seen as a horizontal line. In Figure 3.1 we have denoted this constant force by $\hat{\boldsymbol{f}}_e$, to make it clear that it is a conservative force.

In a linear analysis, internal forces can be expressed as a linear function of the displacements ($\boldsymbol{f}_i(\boldsymbol{u}) = \boldsymbol{K}\ \boldsymbol{u}$). Therefore, the curve $\boldsymbol{f} = \boldsymbol{f}_i(\boldsymbol{u})$ is a slanted line that passes through the origin.

When solving this linear problem, we will obtain the vector of displacements $\hat{\boldsymbol{u}}$, for which external forces equal the internal forces ($\hat{\boldsymbol{f}}_e = \boldsymbol{f}_i(\hat{\boldsymbol{u}})$). So, this vector would be *the* solution to our problem.

But suppose now that we are designing a structure to work under a certain range of loads and we therefore need to obtain things like its maximum displacements, the evolution of its stiffness as the load increases, etc. In this case, we will want to know the *behaviour* of the solid under the application of *any* external force proportional to the original force $\hat{\boldsymbol{f}}_e$ (*i. e.*, a force $\boldsymbol{f}_e$ that can be expressed as $\boldsymbol{f}_e = \alpha\, \hat{\boldsymbol{f}}_e$, with α being any real constant value). Thanks to the fact that the problem is linear, the response of the solid is simply represented by the *straight* line that passes through the origin and point ($\hat{\boldsymbol{u}}$, $\hat{\boldsymbol{f}}_e$), and consequently no additional calculations are required.

However, similar conclusions cannot be drawn in the nonlinear analysis. Figure 3.2 represents the one degree of freedom case. Conservative external forces are again considered but internal forces are not linear on the displacements any more. The solution will also be noted ($\hat{u}, \hat{f}_e$).

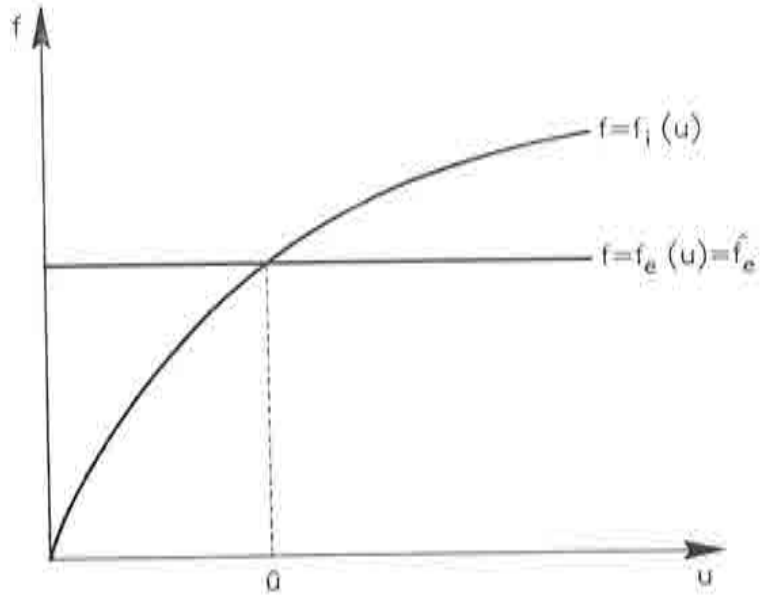


Figure 3.2: Solution of the nonlinear problem for a fixed external force.

In this case, to determine the general response of the solid under an external load proportional to $\hat{\boldsymbol{f}}_e$, several nonlinear problems must be solved, the solutions of which *cannot* be obtained directly from $\hat{\boldsymbol{u}}$.

Furthermore, even if we only need the solution for one external force value, our nonlinear solver may not be capable of finding vector $\hat{\boldsymbol{u}}$ unless the external force is split into several, smaller fractions, each of them being applied at a time. The reason for this is that we will use some simplified strategy to "aim" at point $\hat{\boldsymbol{u}}$; if this point is too "far away", we may need to follow the curve, aiming from one intermediate point to the next, until the final point $\hat{\boldsymbol{u}}$ is reached.

The last two paragraphs impose a change in the approach to the nonlinear problem. The result of this change is what is called an *incremental approach*, which is illustrated in Figure 3.3 and can be described as follows.

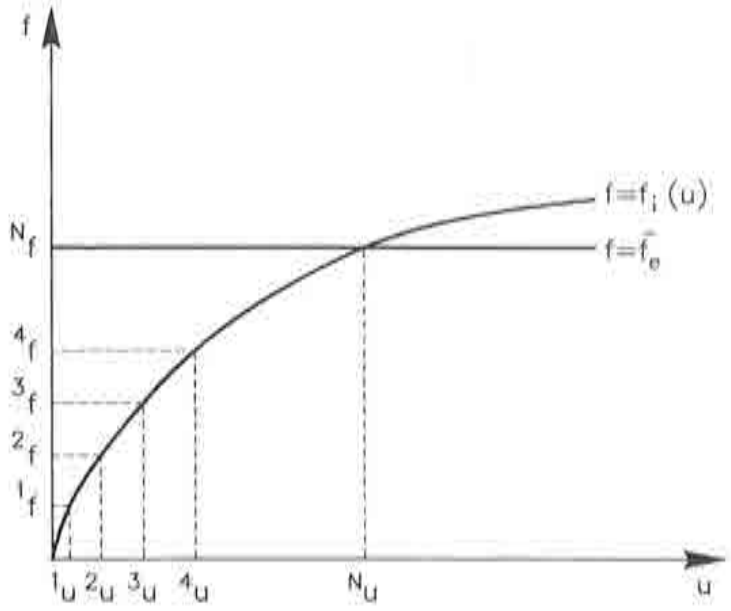


Figure 3.3: Incremental scheme.

First, we divide the total external force into N *increments* or *load steps*. Denoting each increment by ${}^n\Delta\boldsymbol{f}$ and the intermediate forces by ${}^n\boldsymbol{f} = {}^{n-1}\boldsymbol{f} + {}^n\Delta\boldsymbol{f}$ we will have that $\boldsymbol{f}_e = {}^1\Delta\boldsymbol{f} + {}^2\Delta\boldsymbol{f} + \ldots + {}^{N-1}\Delta\boldsymbol{f} + {}^N\Delta\boldsymbol{f} = {}^N\boldsymbol{f}$. The original problem is then transformed into the N nonlinear problems:

1) Find ${}^1\boldsymbol{u}$ such that

$$\boldsymbol{r}({}^1\boldsymbol{u}) = \boldsymbol{f}_i({}^1\boldsymbol{u}) - {}^1\boldsymbol{f} = \boldsymbol{0},$$

2) Find ${}^2\Delta\boldsymbol{u}$ such that, if ${}^2\boldsymbol{u}$ is built as ${}^2\boldsymbol{u} = {}^1\boldsymbol{u} + {}^2\Delta\boldsymbol{u}$, then

$$\boldsymbol{r}({}^2\boldsymbol{u}) = \boldsymbol{f}_i({}^2\boldsymbol{u}) - {}^2\boldsymbol{f} = \boldsymbol{0},$$

etcetera.

For $n = 0, \ldots, N-1$

and assuming we know a vector ${}^n\boldsymbol{u}$ of nodal displacements for which

$\boldsymbol{r}({}^n\boldsymbol{u}) = \boldsymbol{f}_i({}^n\boldsymbol{u}) - {}^n\boldsymbol{f} = \boldsymbol{0}$,

we want to obtain a vector ${}^{n+1}\Delta\boldsymbol{u}$ of *incremental* displacements such that

for point ${}^{n+1}\boldsymbol{u}$, calculated as ${}^{n+1}\boldsymbol{u} = {}^n\boldsymbol{u} + {}^{n+1}\Delta\boldsymbol{u}$,

we have that

$\boldsymbol{r}({}^{n+1}\boldsymbol{u}) = \boldsymbol{f}_i({}^{n+1}\boldsymbol{u}) - {}^{n+1}\boldsymbol{f} = \boldsymbol{0}$

Table 3.1: Incremental approach.

Now we must concentrate in finding ${}^{n+1}\Delta\boldsymbol{u}$.

What we are going to do is *linearise* each set of nonlinear equations listed in Table 3.1; and then (Section 3.2) we are going to see if the solution of the equivalent linear set of equations is similar to that of the original, nonlinear set.

Let us begin by considering the function defined by

$$\boldsymbol{r}(\boldsymbol{u}) = \boldsymbol{f}_i(\boldsymbol{u}) - {}^{n+1}\boldsymbol{f}$$

and *making a Taylor expansion of* $\boldsymbol{r}({}^{n+1}\boldsymbol{u})$ *around point* ${}^n\boldsymbol{u}$. This expansion consists of an infinite number of terms of increasing order. However, since a

linearisation of the problem is performed, only terms of first order are taken into account. We will be approximating $\boldsymbol{r}({}^{n+1}\boldsymbol{u})$ by

$$\boldsymbol{r}({}^{n+1}\boldsymbol{u}) \approx \boldsymbol{r}({}^{n}\boldsymbol{u}) + \left.\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{u}}\right|_{{}^{n}\boldsymbol{u}} {}^{n+1}\Delta\boldsymbol{u},$$

which has to be null. Reorganizing terms in the resulting equation

$$\boldsymbol{r}({}^{n}\boldsymbol{u}) + \left.\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{u}}\right|_{{}^{n}\boldsymbol{u}} {}^{n+1}\Delta\boldsymbol{u} = \boldsymbol{0} \tag{3.2}$$

we obtain our desired *set of linear equations*

$${}^{n}\boldsymbol{K}\ {}^{n+1}\Delta\boldsymbol{u} = -{}^{n}\boldsymbol{r}, \tag{3.3}$$

where ${}^{n}\boldsymbol{r}$ stands for $\boldsymbol{r}({}^{n}\boldsymbol{u})$ and we recover the expression of the *tangent stiffness matrix*

$${}^{n}\boldsymbol{K} = \left.\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{u}}\right|_{{}^{n}\boldsymbol{u}}$$

that was presented in Chapter 1.

3.2 The need for incremental/iterative solutions

In the previous section, our N nonlinear problems were transformed into N linear problems. The idea was to find the *solution* ${}^{n+1}\Delta\boldsymbol{u}$ to the incremental problem, see Section 3.1, by solving a certain set of linear equations that was written symbollically as

$${}^{n}\boldsymbol{K}\ {}^{n+1}\Delta\boldsymbol{u} = -{}^{n}\boldsymbol{r}.$$

However, what happens is that the solution of this linearised set of equations is just an *approximation* to the solution of the original, nonlinear set defined by equation (3.1). Thus, the vector of displacements

$${}^{n+1}\boldsymbol{u} = {}^{n}\boldsymbol{u} + {}^{n+1}\Delta\boldsymbol{u}$$

does not give null residual forces $\boldsymbol{r}({}^{n+1}\boldsymbol{u})$.

We have tried to illustrate this last point in Figure 3.4. The solution to the nonlinear problem is denoted by ${}^{n+1}\hat{\boldsymbol{u}}$.

The obtention of vector ${}^{n+1}\boldsymbol{u}$ begins by performing a Taylor first-order expansion of the function

$$\boldsymbol{r}(\boldsymbol{u}) = \boldsymbol{f}_i(\boldsymbol{u}) - {}^{n+1}\boldsymbol{f}$$

around point ${}^{n}\boldsymbol{u}$. As we are dealing with conservative forces, ${}^{n+1}\boldsymbol{f}$ is constant. Therefore, the nonlinear function $\boldsymbol{f} = \boldsymbol{f}_i(\boldsymbol{u})$ will be approximated by the *linear* function

$$\boldsymbol{f} = {}^{n}\boldsymbol{f} + {}^{n}\boldsymbol{K}\,(\boldsymbol{u} - {}^{n}\boldsymbol{u}),$$

which is tangent to the curve $\boldsymbol{f} = \boldsymbol{f}_i(\boldsymbol{u})$ at point $({}^{n}\boldsymbol{u},\ {}^{n}\boldsymbol{f})$.

The intersection of this linear function with the constant function $\boldsymbol{f} = {}^{n+1}\boldsymbol{f}$ gives point ${}^{n+1}\boldsymbol{u}$, for which

$${}^{n}\boldsymbol{f} + {}^{n}\boldsymbol{K}\,({}^{n+1}\boldsymbol{u} - {}^{n}\boldsymbol{u}) = {}^{n+1}\boldsymbol{f},$$

(or also ${}^{n}\boldsymbol{K}\,{}^{n+1}\Delta\boldsymbol{u} = {}^{n}\boldsymbol{K}\,({}^{n+1}\boldsymbol{u} - {}^{n}\boldsymbol{u}) = {}^{n+1}\boldsymbol{f} - {}^{n}\boldsymbol{f} = -{}^{n}\boldsymbol{r}$).

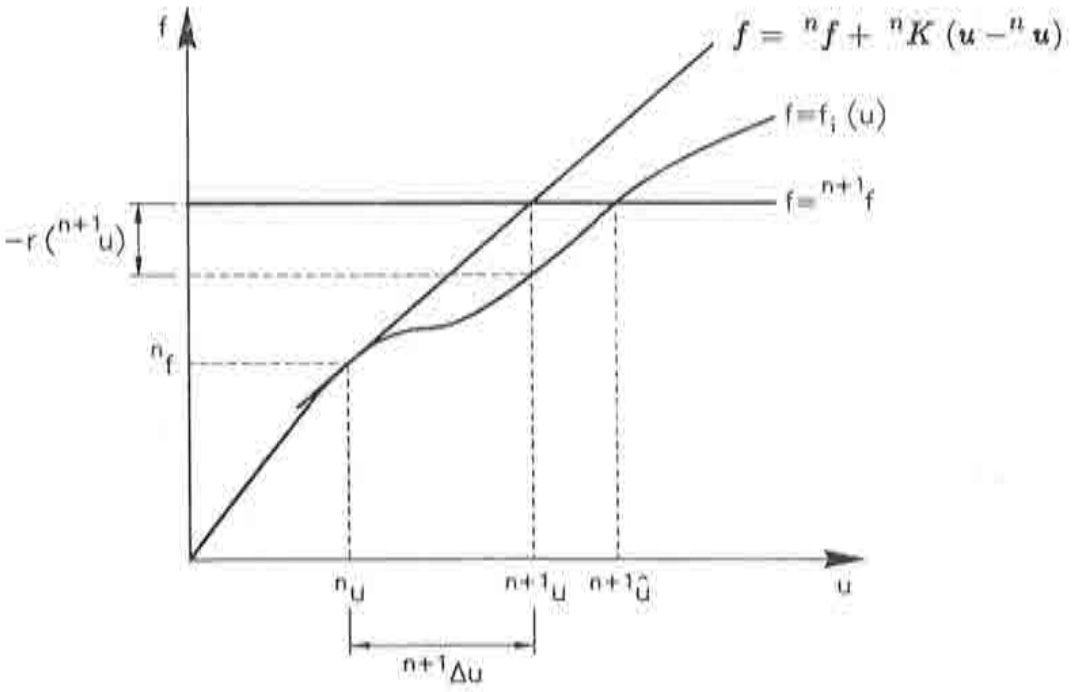


Figure 3.4: Solutions to the nonlinear and linearised problems.

So, the procedure we used in Section 3.1 seems, as itself, rather disappointing. Nevertheless, what we can do is take profit of this idea to approximate more and more to the solution within each load step. These approximations are called *iterations*; the final scheme is called thereby an *incremental/iterative solution* and it works as follows.

- For a certain load step ${}^{n+1}\Delta\boldsymbol{f}$, we solve the linear set of equations

$$ {}^{n}\boldsymbol{K}\ {}^{n+1}\Delta\boldsymbol{u} = -{}^{n}\boldsymbol{r}, \tag{3.4} $$

which is identical to the one in the incremental scheme.

Writing the out-of-balance forces associated to a certain displacement vector $\boldsymbol{u}$ as

$$ \boldsymbol{r}(\boldsymbol{u}) = \boldsymbol{f}_i(\boldsymbol{u}) - {}^{n+1}\boldsymbol{f} \tag{3.5} $$

we have that, for equation (3.4), the residual forces ${}^{n}\boldsymbol{r}$ correspond to the total increment of load:

$$ {}^{n}\boldsymbol{r} = \boldsymbol{r}({}^{n}\boldsymbol{u}) = \boldsymbol{f}_i({}^{n}\boldsymbol{u}) - {}^{n+1}\boldsymbol{f} = -{}^{n+1}\Delta\boldsymbol{f}. $$

The solution to equation (3.4) will be designated by ${}^{n+1}\Delta\boldsymbol{u}^1$ to indicate that it yields just a *prediction* ${}^{n+1}\boldsymbol{u}^1 = {}^{n}\boldsymbol{u} + {}^{n+1}\Delta\boldsymbol{u}^1$ to the solution of the nonlinear problem.

- This prediction might however be a good solution for equation (3.4). To check this possibility, we need to compute at this stage the value of vector

$$ \boldsymbol{r}({}^{n+1}\boldsymbol{u}^1) = \boldsymbol{f}_i({}^{n+1}\boldsymbol{u}^1) - {}^{n+1}\boldsymbol{f}. $$

If this value is close enough to zero, no iterations will be needed.

- If, on the contrary, $\boldsymbol{r}({}^{n+1}\boldsymbol{u}^1)$ is too different from zero, we start the iterative part of the process.

What we do is simply consider that point ${}^{n+1}\boldsymbol{u}^1$ plays the same role now as ${}^{n}\boldsymbol{u}$ did in the prediction phase.

Therefore, our out-of-balance forces are computed as

$$ \boldsymbol{r}({}^{n+1}\boldsymbol{u}^1) = \boldsymbol{f}_i({}^{n+1}\boldsymbol{u}^1) - {}^{n+1}\boldsymbol{f} $$

and we consequently need to solve the linear set

$$ {}^{n+1}\boldsymbol{K}^1 \, {}^{n+1}\boldsymbol{\delta u}^2 = -{}^{n+1}\boldsymbol{r}^1, \tag{3.6} $$

where ${}^{n+1}\boldsymbol{r}^1 = \boldsymbol{r}({}^{n+1}\boldsymbol{u}^1)$ and ${}^{n+1}\boldsymbol{K}^1$ is the tangent matrix calculated at point $({}^{n+1}\boldsymbol{u}^1, \boldsymbol{f}_i({}^{n+1}\boldsymbol{u}^1))$, namely

$$ {}^{n+1}\boldsymbol{K}^1 = \left.\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{u}}\right|_{{}^{n+1}\boldsymbol{u}^1}. $$

The solution ${}^{n+1}\boldsymbol{\delta u}^2$ to equation (3.6) is a *correction* of the one obtained for equation (3.4); we can construct after this correction a *better* approximation ${}^{n+1}\boldsymbol{u}^2$ to the solution of the nonlinear set by a simple update such as the following:

$$ {}^{n+1}\boldsymbol{\Delta u}^2 = {}^{n+1}\boldsymbol{\Delta u}^1 + {}^{n+1}\boldsymbol{\delta u}^2 $$

$$ {}^{n+1}\boldsymbol{u}^2 = {}^{n}\boldsymbol{u} + {}^{n+1}\boldsymbol{\Delta u}^2. $$

• After this first correction, we have to check if point ${}^{n+1}\boldsymbol{\Delta u}^2$ supplies a good enough solution of the problem by evaluating the residual vector

$$ \boldsymbol{r}({}^{n+1}\boldsymbol{u}^2) = \boldsymbol{f}_i({}^{n+1}\boldsymbol{u}^2) - {}^{n+1}\boldsymbol{f}. $$

• Otherwise, we look for a new vector ${}^{n+1}\boldsymbol{\delta u}^3$ such that

$$ {}^{n+1}\boldsymbol{K}^2 \, {}^{n+1}\boldsymbol{\delta u}^3 = -{}^{n+1}\boldsymbol{r}^2 $$

and so on.

After a certain amount of iterations, this process will hopefully come to an end. To know at which iteration to stop, we simply evaluate how close the approximation ${}^{n+1}\boldsymbol{u}^{k+1}$ is to the real solution (for example, by computing how different the residual vector $\boldsymbol{r}({}^{n+1}\boldsymbol{u}^{k+1}) = \boldsymbol{f}_i({}^{n+1}\boldsymbol{u}^{k+1}) - {}^{n+1}\boldsymbol{f}$ is from zero).

For a generic iteration $k+1$, the sequence is described in Table 3.2. For $k = 0$, we assume that $\boldsymbol{\delta u}^1 = \boldsymbol{\Delta u}^1$.

Solve	${}^{n+1}\boldsymbol{K}^{k}\ {}^{n+1}\boldsymbol{\delta u}^{k+1} = -{}^{n+1}\boldsymbol{r}^{k}$
Update	${}^{n+1}\boldsymbol{\Delta u}^{k+1} = {}^{n+1}\boldsymbol{\Delta u}^{k} + {}^{n+1}\boldsymbol{\delta u}^{k+1}$
Update	${}^{n+1}\boldsymbol{u}^{k+1} = {}^{n}\boldsymbol{u} + {}^{n+1}\boldsymbol{\Delta u}^{k+1}$
Convergence control : if the approximation ${}^{n+1}\boldsymbol{u}^{k+1}$ is good enough, exit.	

Table 3.2: Iterative scheme within load step $n+1$.

To end this section, we have to say that the *convergence control* step, which was introduced using a *force-based* criterion, can be performed in many other different ways. For instance, if we alternatively employ a *displacement-based* criterion, we can measure the variation in the solution vector when updating displacements: imagine we compute the scalar e defined by

$$e = \frac{\|{}^{n+1}\boldsymbol{\delta u}^{k+1}\|}{\|{}^{n+1}\boldsymbol{u}^{k+1}\|} \tag{3.7}$$

where $\|\cdot\|$ stands for some vectorial norm (typically, the Euclidean or maximum norm). A very small value of e means that the correction vector ${}^{n+1}\boldsymbol{\delta u}^{k+1}$ is almost not changing the total displacement vector at all and that we can consequently pass on to the next load step.

Force-based and displacement-based convergence criteria are the most common. Other criteria, such as those involving *energies*, need to be used with more precaution [10].

3.3 Two interesting flowcharts: procedures NONLIN and INCREME

In Castem2000, the task of performing the incremental/iterative scheme described in Section 3.2 is accomplished by two *procedures* — NONLIN and INCREME.

Procedure NONLIN (NON LINéaire) essentially manages the incremental strategy, that is

1. it computes the total external force that corresponds to the current load step,

2. it passes the information over to INCREME, which iterates to equilibrium,

3. and, finally, it updates displacements, strains and stresses and prepares for the next load step.

Figure 3.5 gives a flowchart for this procedure.

As for procedure INCREME, it is the one that carries out all the iterative calculations. These calculations are described step-by-step in Figure 3.6.

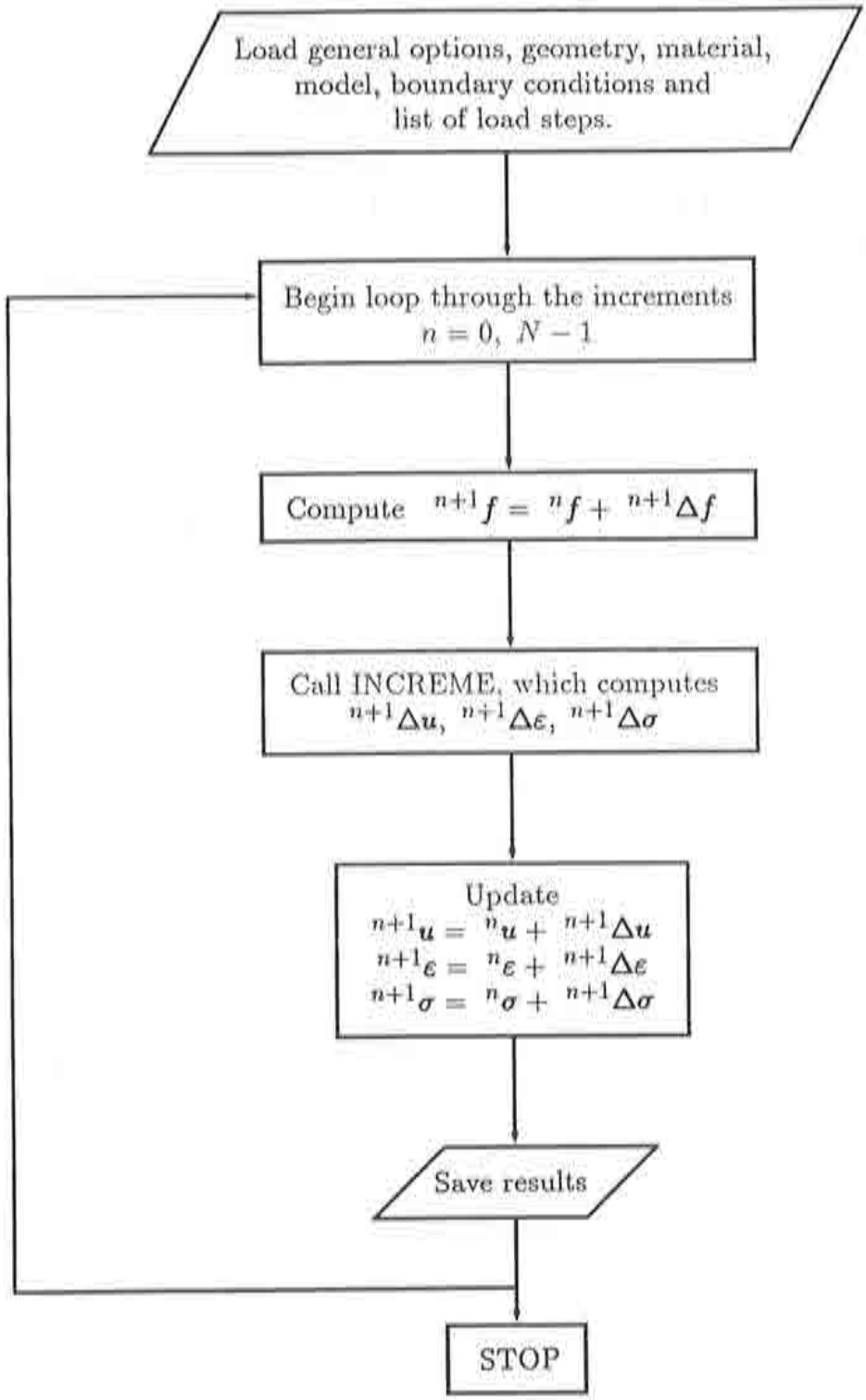


Figure 3.5: Flowchart for *procedure* NONLIN.

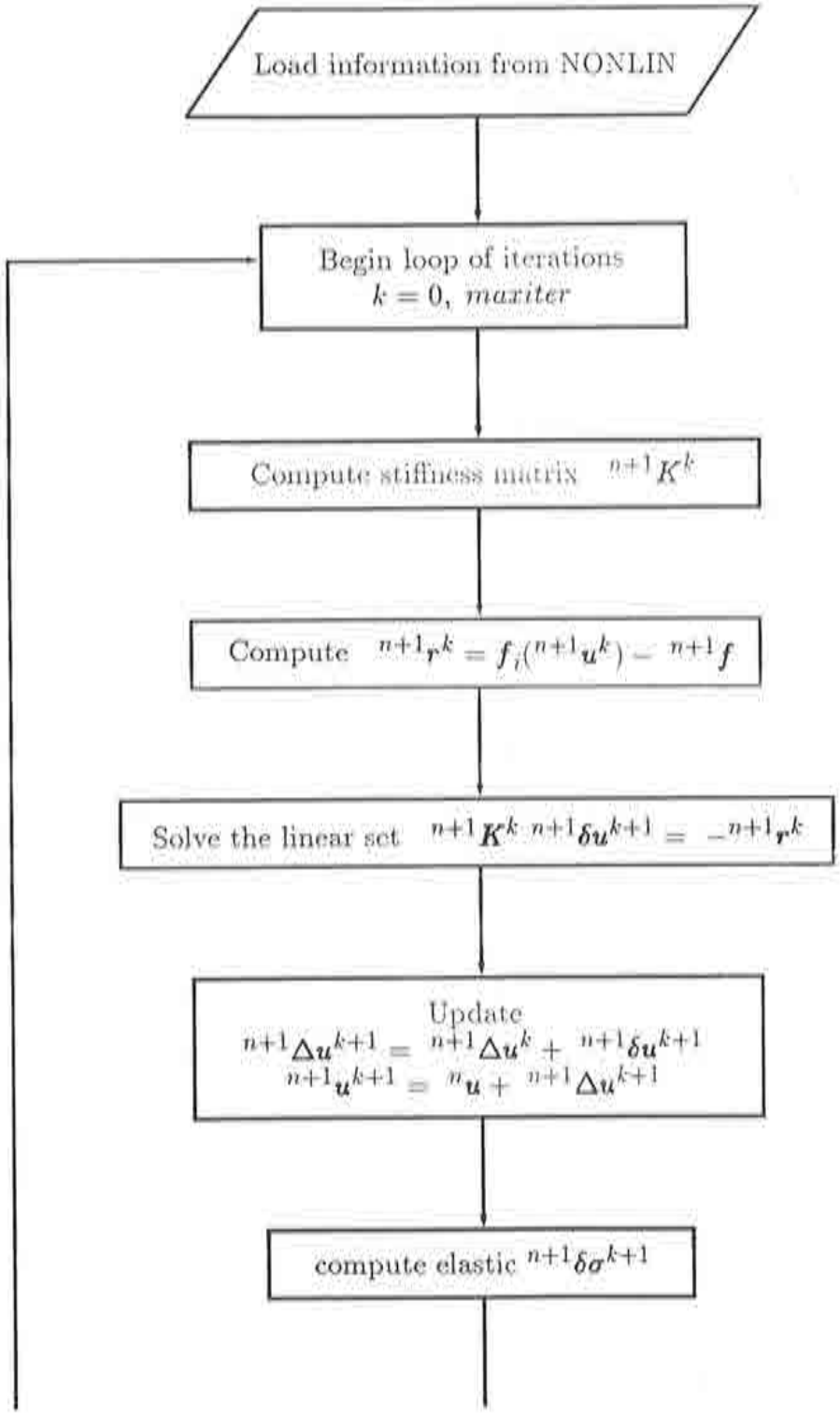


Figure 3.6: Flowchart for *procedure* INCREME.

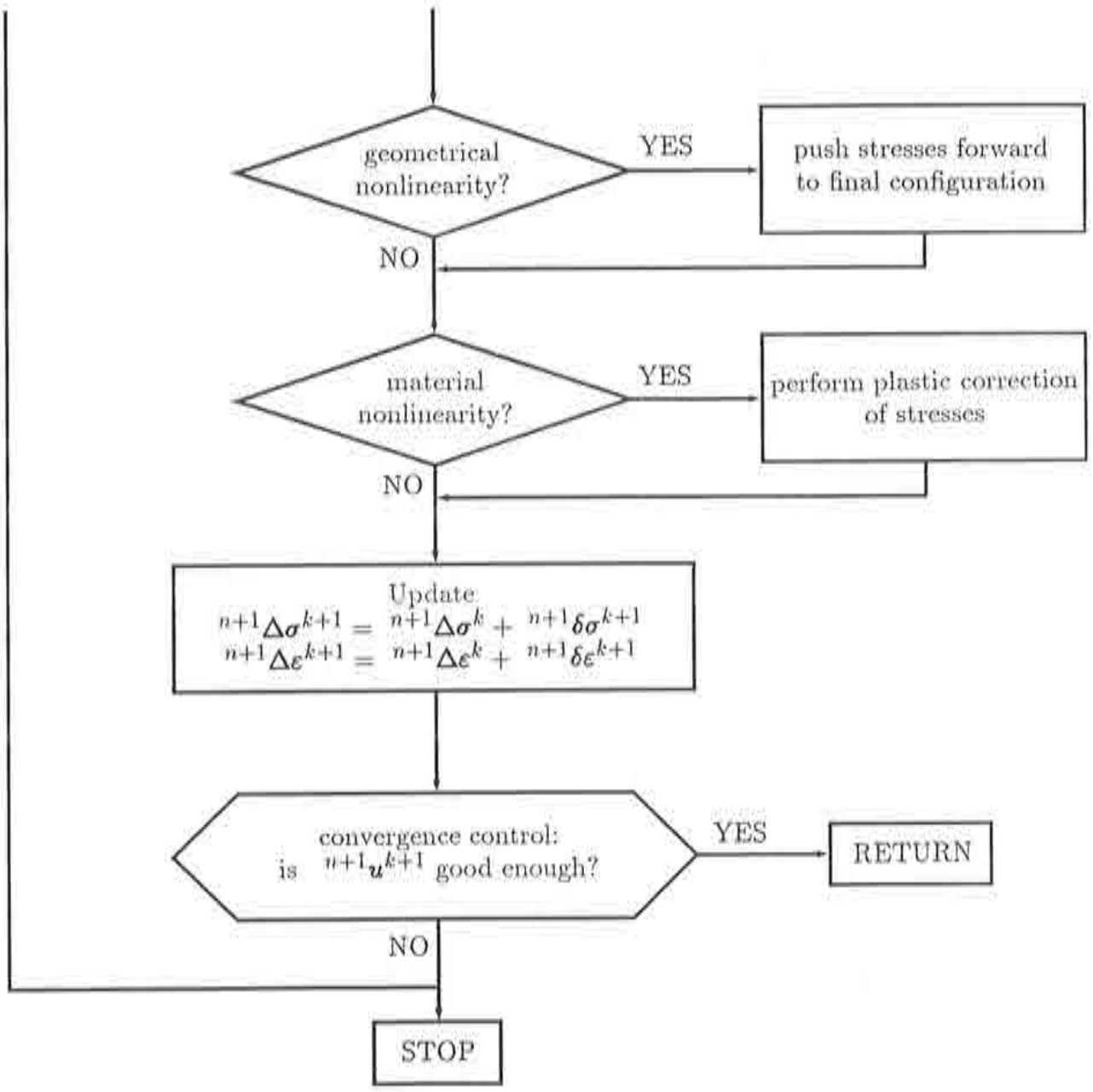


Figure 3.6: Flowchart for *procedure* INCREME (continued).

3.4 Full and modified Newton–Raphson methods

Full Newton–Raphson method

In Sections 3.1 and 3.2 we have presented the classical *incremental/iterative* approach to nonlinear problems. As a matter of fact, to illustrate that approach we have already used a specific *method*: the so-called *full Newton–Raphson method* (or *fNR*).

In this section, we want to develop a first set of methods (beginning with fNR and continuing with some variations on fNR) which allow us to solve a given nonlinear problem, always following the incremental/iterative scheme.

Let us rewrite the full Newton–Raphson algorithm. Suppose we are located in a certain load step $n+1$, where $({}^{n}\boldsymbol{u}, {}^{n}\boldsymbol{f})$ is a known equilibrium point and we want to obtain ${}^{n+1}\boldsymbol{u}$. Considering ${}^{n}\boldsymbol{u}$ as a starting point for this increment, we can rename it to ${}^{n+1}\boldsymbol{u}^{0}$, where the right superscript 0 indicates that the construction of point ${}^{n+1}\boldsymbol{u}^{0}$ is previous to iterations $1, 2, \ldots$. Consequently, the stiffness matrix ${}^{n}\boldsymbol{K}$ and the out-of-balance force vector ${}^{n}\boldsymbol{r}$ can also be rewritten respectively as ${}^{n+1}\boldsymbol{K}^{0}$ and ${}^{n+1}\boldsymbol{r}^{0}$.

After these substitutions, all terms in the iterative scheme presented in Section 3.2 turn up to have a $n+1$ left superscript. To lighten the overall notation, we drop left superscripts. Thus, in the coming equations, ${}^{n+1}\boldsymbol{K}^{0}$ is written as $\boldsymbol{K}^{0}$, ${}^{n+1}\boldsymbol{\Delta u}^{1}$ is denoted by $\boldsymbol{\Delta u}^{1}$, and so on.

Finally, the original list of instructions can be rearranged to produce the compact algorithm detailed in Table 3.3. Figure 3.7 illustrates how this method works for a one-dimensional problem.

prediction

1. Compute $\boldsymbol{K}^0$
2. Solve the linear set $\boldsymbol{K}^0 \, \Delta \boldsymbol{u}^1 = -\boldsymbol{r}^0$
3. Update $\boldsymbol{u}^1 = \boldsymbol{u}^0 + \Delta \boldsymbol{u}^1$
4. Evaluate $\boldsymbol{r}^1 = \boldsymbol{r}(\boldsymbol{u}^1)$
5. Convergence control: if point $\boldsymbol{u}^1$ is good enough, exit.

corrections

$k \geq 1$

6. Compute $\boldsymbol{K}^k$
7. Solve the linear set $\boldsymbol{K}^k \, \delta \boldsymbol{u}^{k+1} = -\boldsymbol{r}^k$
8. Update $\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \delta \boldsymbol{u}^{k+1}$
9. Evaluate $\boldsymbol{r}^{k+1} = \boldsymbol{r}(\boldsymbol{u}^{k+1})$
10. Convergence control: if point $\boldsymbol{u}^{k+1}$ is good enough, exit.
11. Assign $k = k + 1$ and go back to 6.

Table 3.3: Algorithm for the full Newton–Raphson method.

Remark 1. Under certain conditions, [10,11], the full Newton–Raphson method shows a quadratic rate of convergence, *i.e.*, a scalar γ can be found for which

$$\|\boldsymbol{u}^{k+1} - \boldsymbol{\alpha}\| \leq \gamma \, \|\boldsymbol{u}^k - \boldsymbol{\alpha}\|^2$$

where $\boldsymbol{\alpha}$ denotes the solution corresponding to the current increment. Thus, fNR results in a powerful technique.

Remark 2. At every iteration, we must compute and factorise one tangent stiffness matrix, solve one linear set of equations and evaluate one vector of residual forces. Consequently, the fNR is an expensive method.

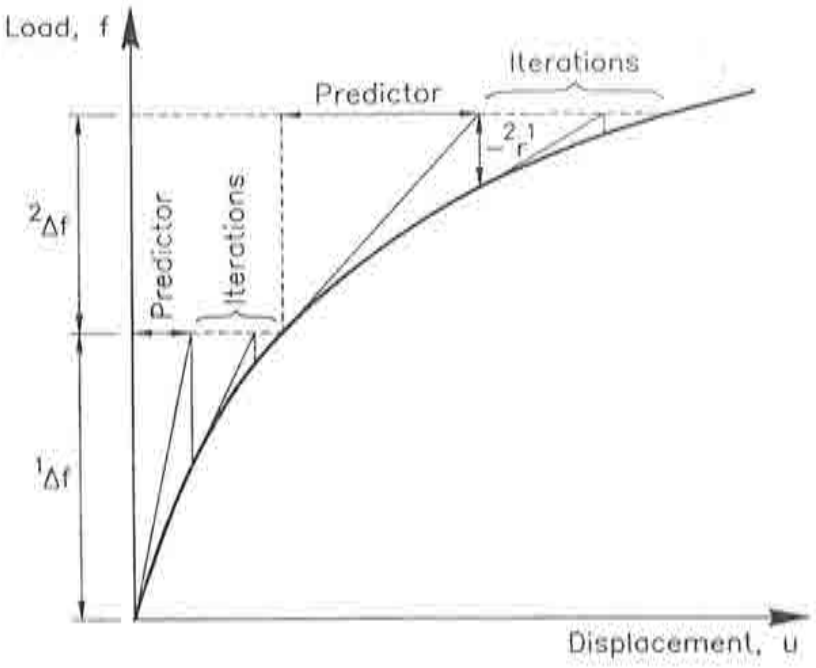


Figure 3.7: The full Newton–Raphson method.

Modifications to the full Newton–Raphson method

Once the basic full Newton–Raphson scheme is defined, a whole set of 'modified Newton–Raphson methods' can be derived from it. The idea of these methods is to keep the original pattern, but using not always up-to-date stiffness matrices.

But why should these modifications be made? Calculating a stiffness matrix $\boldsymbol{K}$ and factorising it to solve the corresponding linear set of equations at every iteration is a very expensive task. Therefore, using a previously calculated and factorised stiffness matrix will result in a much lower computational cost *per iteration*.

For instance, while in the fNR method a new tangent stiffness matrix is computed at every iteration, in the standard *modified Newton–Raphson method*

(or *mNR*) the stiffness matrix is updated just at the beginning of each load increment, see Figure 3.8. This means that less work is required because only one tangent matrix is computed and factorised per increment. But it also means that a higher amount of iterations must be expected before convergence is attained. Therefore, we cannot know beforehand if a mNR is going to be more or less expensive –globally– than a fNR given a certain nonlinear problem. The algorithm for the mNR method is listed in Table 3.4.

prediction

1. Compute $\boldsymbol{K}^0$
2. Solve the linear set $\boldsymbol{K}^0\ \Delta\boldsymbol{u}^1 = -\boldsymbol{r}^0$
3. Update $\boldsymbol{u}^1 = \boldsymbol{u}^0 + \Delta\boldsymbol{u}^1$
4. Evaluate $\boldsymbol{r}^1 = \boldsymbol{r}(\boldsymbol{u}^1)$
5. Convergence control: if point $\boldsymbol{u}^1$ is good enough, exit.

corrections

$k \geq 1$

6. Solve the linear set $\boldsymbol{K}^0\ \delta\boldsymbol{u}^{k+1} = -\boldsymbol{r}^k$
8. Update $\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \delta\boldsymbol{u}^{k+1}$
9. Evaluate $\boldsymbol{r}^{k+1} = \boldsymbol{r}(\boldsymbol{u}^{k+1})$
10. Convergence control: if point $\boldsymbol{u}^{k+1}$ is good enough, exit.
11. Assign $k = k + 1$ and go back to 6.

Table 3.4: Algorithm for the modified Newton–Raphson method.

Remark. This method has a linear rate of convergence, *i.e.*, a scalar γ can be found for which

$$\|\boldsymbol{u}^{k+1} - \boldsymbol{\alpha}\| \leq \gamma\ \|\boldsymbol{u}^k - \boldsymbol{\alpha}\|,$$

where $\boldsymbol{\alpha}$ denotes the solution corresponding to the current increment, [10,11].

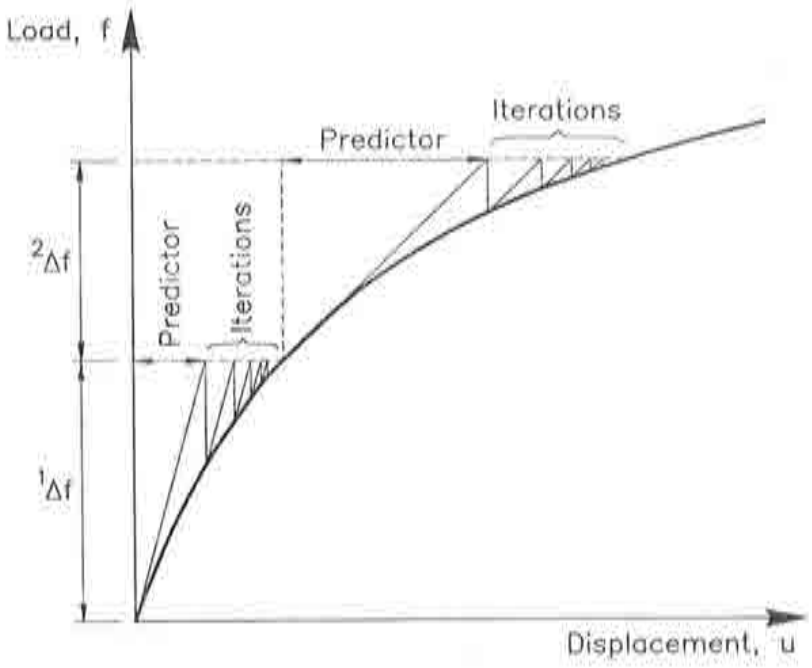


Figure 3.8: The modified Newton–Raphson method.

Of course, an infinite number of Newton–Raphson techniques can be produced by updating the tangent stiffness matrix at different times during the resolution of the problem.

Among these techniques, we feel that at least one more must be discussed here, and that is the so-called *initial stress method*. In this case, one tangent stiffness matrix is computed at the beginning of the *first* load step, namely ${}^0\boldsymbol{K}$, and it is always used up to the end of the problem. Therefore, just one tangent matrix is calculated and factorised in all. This means, as usual, that more iterations will probably be needed within each load step (that is, *if* convergence is attained, which becomes more improbable since the tangent stiffness matrix differs more radically from the employed matrix). This technique is illustrated in Figure 3.9.

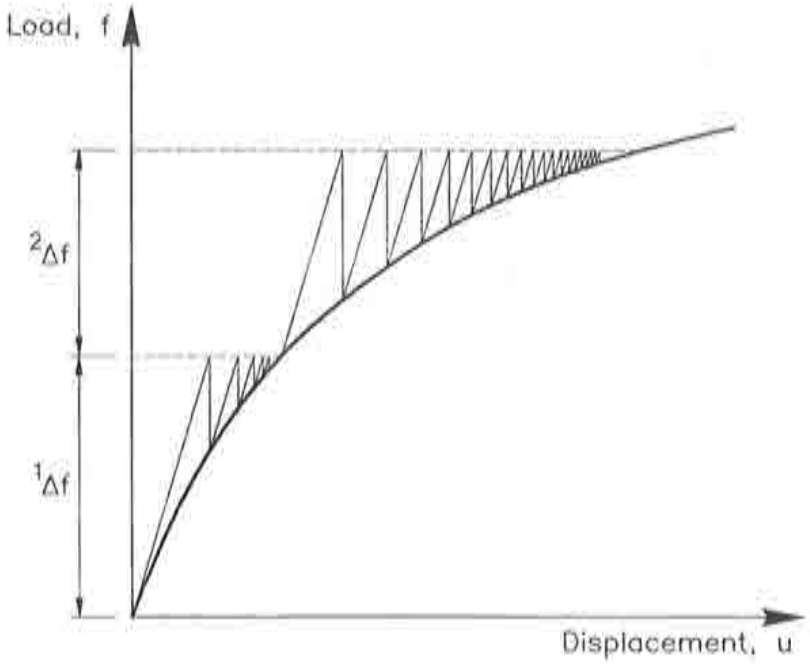


Figure 3.9: The Initial Stress method.

3.5 What Castem has that others don't: introducing Lagrange multipliers

In previous sections, the resolution of our nonlinear set of equations has been transformed into a sequence of linear sets which are solved successively. In fact, as it will be seen in further sections, this will also happen with every method studied here.

In this section, we deal with the problem of incorporating boundary conditions to solve those linear sets.

A standard technique which accomplishes this purpose consists of replacing those unknowns associated to boundary conditions with their prescribed values

so that, finally, a smaller set of linear equations must be solved, [15,20,30].

Castem2000 uses an alternative solution, based on the use of Lagrange multipliers.

The Lagrange-multiplier technique

This technique consists of augmenting the original vector of displacements $\boldsymbol{u}$ by two additional l-dimensional unknowns that we denote by $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ (Lagrange multipliers), [20,21,30]. The augmented vector $(\boldsymbol{u}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ is built so as to supply the stationary solution of the *elastic functional* ω given by

$$\omega = \frac{1}{2}\,\boldsymbol{u}^T \cdot \boldsymbol{K}\,\boldsymbol{u} - \boldsymbol{u}^T \cdot \boldsymbol{f}_e + \boldsymbol{\lambda}^T \cdot (\boldsymbol{A}_e\,\boldsymbol{u} - \boldsymbol{b}_e) + \boldsymbol{\lambda}^T \cdot (\boldsymbol{A}_e\,\boldsymbol{u} - \boldsymbol{b}_e) + \frac{1}{2}\,(\boldsymbol{\lambda} - \boldsymbol{\mu})^T \cdot (\boldsymbol{\lambda} - \boldsymbol{\mu}),$$

where $\boldsymbol{f}_e$ will now stand for the vector of known external forces (thus excluding reaction forces) and $\boldsymbol{A}_e\,\boldsymbol{u} = \boldsymbol{b}_e$ represent l linear boundary conditions.

This stationary solution can be obtained by using a minimisation procedure, which requires that

$$\frac{\partial w}{\partial \boldsymbol{u}} = \boldsymbol{0}\,, \quad \frac{\partial w}{\partial \boldsymbol{\lambda}} = \boldsymbol{0}\,, \quad \frac{\partial w}{\partial \boldsymbol{\mu}} = \boldsymbol{0}. \tag{3.8}$$

If we expand the three conditions listed in equation (3.8), we obtain that the following equations must be verified:

$$\boldsymbol{K}\,\boldsymbol{u} + \boldsymbol{A}_e^T\,(\boldsymbol{\lambda} + \boldsymbol{\mu}) - \boldsymbol{f}_e = \boldsymbol{0}, \tag{3.9}$$

$$\boldsymbol{A}_e^T\,\boldsymbol{u} + \boldsymbol{\lambda} - \boldsymbol{\mu} - \boldsymbol{b}_e = \boldsymbol{0}, \tag{3.10}$$

$$\boldsymbol{A}_e^T\,\boldsymbol{u} - \boldsymbol{\lambda} + \boldsymbol{\mu} - \boldsymbol{b}_e = \boldsymbol{0}. \tag{3.11}$$

From equations (3.10) and (3.11) we can see that the additional unknowns $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ must be *equal* vectors. Of course, this remark could have been made right from the start, as no *privileged* definition is given for one multiplier with respect

to the other (in other words, the definition of ω is symmetrical with respect to $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$).

As for equation (3.9), we know that the term $\boldsymbol{K}\,\boldsymbol{u}$ yields the internal nodal forces. Thus, using the given definition of $\boldsymbol{f}_e$, we have that the term

$$\boldsymbol{f}_r = \boldsymbol{A}_e^T\,(\boldsymbol{\lambda}+\boldsymbol{\mu}) = \boldsymbol{f}_e - \boldsymbol{K}\,\boldsymbol{u} = \boldsymbol{f}_e - \boldsymbol{f}_i$$

must account for the *reaction forces* associated to $\boldsymbol{u}$.

We can finally rewrite equations (3.9) to (3.11) under the shape of a new set of linear equations such as the following

$$\begin{pmatrix} \boldsymbol{K} & \boldsymbol{A}_e^T & \boldsymbol{A}_e^T \\ \boldsymbol{A}_e & \boldsymbol{I} & -\boldsymbol{I} \\ \boldsymbol{A}_e & -\boldsymbol{I} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{\lambda} \\ \boldsymbol{\mu} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_e \\ \boldsymbol{b}_e \\ \boldsymbol{b}_e \end{pmatrix} \tag{3.12}$$

where $\boldsymbol{I}$ stands for the l-dimensional identity matrix.

Hence, the Lagrange-multiplier technique consists in the resolution of an $(n+2l)$-dimensional set of equations (therefore a *larger* set of equations).

Once this set is solved, the reaction forces can be obtained simply by computing

$$\boldsymbol{f}_r = \boldsymbol{A}_e^T\,(\boldsymbol{\lambda}+\boldsymbol{\mu}) = 2\,\boldsymbol{A}_e^T\,\boldsymbol{\lambda} = 2\,\boldsymbol{A}_e^T\,\boldsymbol{\mu}, \tag{3.13}$$

where now $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are both known vectors.

But why should we be interested in transforming the original set of equations into a *larger* set?

The key idea is that imposing boundary conditions through the standard technique may turn up to be a cumbersome task. Moreover, when dealing with nonlinear constraints such as contact boundary conditions, or also when using combinations of different kinds of elements, etc., we would find out that they can be easily imposed through a Lagrange-multiplier technique, whereas the standard method would involve some clumsy operation sequence.

Another reason for using Lagrange multipliers is that objects are more *neatly* built: first, a stiffness matrix $\boldsymbol{K}$ and a boundary condition matrix $\boldsymbol{A}_c$ are computed; then, matrices $\boldsymbol{K}$ and $\boldsymbol{A}_c$ are put together to form the corresponding enlarged matrix $\boldsymbol{J}$; finally, matrix $\boldsymbol{J}$ is used to solve the necessary sets of equations. In this way, objects are used to produce new objects, but they are not modified, in any case, depending on the use that we intend to make of them.

Once the Lagrange-multiplier option is chosen, there is one more decision that needs to be made: *how many* multipliers do we want to use?

If *one* Lagrange multiplier $\boldsymbol{\lambda}$ is used, equation (3.12) will have to be replaced with the following, smaller set:

$$\begin{pmatrix} \boldsymbol{K} & \boldsymbol{A}_c^T \\ \boldsymbol{A}_c & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{\lambda} \end{pmatrix} \begin{pmatrix} \boldsymbol{f}_e \\ \boldsymbol{b}_c \end{pmatrix}, \tag{3.14}$$

where $\boldsymbol{0}$ stands for the l-dimensional zero matrix.

The advantage of solving (3.12) instead of (3.14) is that the resolution of the set that corresponds to one multiplier may involve *pivoting* of the matrix, whereas (3.12) will not.

Newton–Raphson methods in combination with Lagrange multipliers

To end this chapter, one last question needs to be answered:

What happens to Newton–Raphson methods when using Lagrange multipliers?

The only idea we must have in mind to answer that question is that the original set of equations has been transformed into a new, larger set (3.12).

So, what we are going to do is follow step-by-step one of the Newton–Raphson algorithms (for instance, the fNR algorithm listed in Table 3.3, Section 3.4) and see how the items that appear in it have to be read *.

Let us start then by the prediction phase, remembering that the current increment number is $n+1$, which means that no left superscript equals an $n+1$ left superscript.

1. First of all, we must compute the stiffness matrix $\boldsymbol{K}^0$. In our case, this stiffness matrix needs to be enlarged into the following matrix

$$\begin{pmatrix} \boldsymbol{K}^0 & \boldsymbol{A}_e^T & \boldsymbol{A}_e^T \\ \boldsymbol{A}_e & \boldsymbol{I} & -\boldsymbol{I} \\ \boldsymbol{A}_e & -\boldsymbol{I} & \boldsymbol{I} \end{pmatrix}$$

which we will denote by $\boldsymbol{J}^0$.

2. After the previous matrix has been built, we want to use it to solve the set of linear equations:

$$\boldsymbol{J}^0 \, \Delta \boldsymbol{U}^1 = -\boldsymbol{R}^0,$$

where $\Delta \boldsymbol{U}^1$ and $\boldsymbol{R}^0$ are the augmented versions of $\Delta \boldsymbol{u}^1$ and $\boldsymbol{r}^0$, respectively. For instance,

$$\Delta \boldsymbol{U}^1 = \begin{pmatrix} \Delta \boldsymbol{u}^1 \\ \Delta \lambda^1 \\ \Delta \boldsymbol{\mu}^1 \end{pmatrix}.$$

As for the independent vector of out-of-balance forces, what we will have to do is *split* the values of the boundary conditions $\boldsymbol{b}_e$ in the same way as we did for the external forces $\boldsymbol{f}_e$ in Section 3.1. Consequently, we will be able to write

$$^n\boldsymbol{b} = {}^{n-1}\boldsymbol{b} + {}^n\Delta\boldsymbol{b} \qquad n = 1, \ldots, N,$$

$$\boldsymbol{b}_e = {}^1\Delta\boldsymbol{b} + {}^2\Delta\boldsymbol{b} + \ldots + {}^{N-1}\Delta\boldsymbol{b} + {}^N\Delta\boldsymbol{b} = {}^N\boldsymbol{b}.$$

* Notice that the algorithm in Section 3.3 is a "blank" algorithm, in the sense that no method to treat boundary conditions has been specified. Therefore, an *interpretation* is needed for all linear sets.

In this context, the independent vector $\boldsymbol{r}^0$ will have to be enlarged into the following vector $\boldsymbol{R}^0$

$$\boldsymbol{R}^0 = \begin{pmatrix} {}^n\boldsymbol{f} - {}^{n+1}\boldsymbol{f} \\ {}^n\boldsymbol{b} - {}^{n+1}\boldsymbol{b} \\ {}^n\boldsymbol{b} - {}^{n+1}\boldsymbol{b} \end{pmatrix}.$$

3. The displacement update is performed simply by adding up the corresponding augmented vectors:

$$\begin{pmatrix} \boldsymbol{u}^1 \\ \boldsymbol{\lambda}^1 \\ \boldsymbol{\mu}^1 \end{pmatrix} = \begin{pmatrix} \boldsymbol{u}^0 \\ \boldsymbol{\lambda}^0 \\ \boldsymbol{\mu}^0 \end{pmatrix} + \begin{pmatrix} \Delta\boldsymbol{u}^1 \\ \Delta\boldsymbol{\lambda}^1 \\ \Delta\boldsymbol{\mu}^1 \end{pmatrix}.$$

4. After the prediction vector is computed, we must evaluate the vector of residual forces $\boldsymbol{r}^1$. Before introducing Lagrange multipliers, only out-of-balance *forces* were considered. Now, also out-of-balance *boundary conditions* will have to be taken into account. This means that we need to build a certain augmented vector $\boldsymbol{R}^1$ as

$$\boldsymbol{R}^1 = \begin{pmatrix} \boldsymbol{f}_i(\boldsymbol{u}^1) - {}^{n+1}\boldsymbol{f} + \boldsymbol{A}_e^T(\boldsymbol{\lambda}^1 + \boldsymbol{\mu}^1) \\ \boldsymbol{A}_e\,\boldsymbol{u}^1 - {}^{n+1}\boldsymbol{b} \\ \boldsymbol{A}_e\,\boldsymbol{u}^1 - {}^{n+1}\boldsymbol{b} \end{pmatrix}.$$

Naturally, what we have written for the prediction phase can be generalised for the corrections (points 6. to 11. in Table 3.3, Section 3.4).

At this stage, and before new methods are discussed, we want to review the notation that has been introduced.

From now on, we are going to designate by $\boldsymbol{J}^k$ $(k \geq 0)$ the following enlarged stiffness matrix:

$$\boldsymbol{J}^k = \begin{pmatrix} \boldsymbol{K}^k & \boldsymbol{A}_e^T & \boldsymbol{A}_e^T \\ \boldsymbol{A}_e & \boldsymbol{I} & -\boldsymbol{I} \\ \boldsymbol{A}_e & -\boldsymbol{I} & \boldsymbol{I} \end{pmatrix}, \tag{3.15}$$

where $\boldsymbol{K}^k$ is the tangent stiffness matrix $\boldsymbol{K}^k = \left.\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{u}}\right|_{\boldsymbol{u}^k}$ and $\boldsymbol{I}$ stands, as usual, for the l-dimensional identity matrix.

The augmented vector of unknowns will be denoted by $\boldsymbol{U}$, in such a way that

$$\boldsymbol{U}^k = \begin{pmatrix} \boldsymbol{u}^k \\ \boldsymbol{\lambda}^k \\ \boldsymbol{\mu}^k \end{pmatrix} \qquad \text{for} \quad \mathrm{k} \geq 0. \tag{3.16}$$

Consequently, we will be able to write things like

$$\Delta \boldsymbol{U}^1 = \begin{pmatrix} \Delta \boldsymbol{u}^1 \\ \Delta \boldsymbol{\lambda}^1 \\ \Delta \boldsymbol{\mu}^1 \end{pmatrix} \tag{3.17}$$

or also

$$\delta \boldsymbol{U}^{k+1} = \begin{pmatrix} \delta \boldsymbol{u}^{k+1} \\ \delta \boldsymbol{\lambda}^{k+1} \\ \delta \boldsymbol{\mu}^{k+1} \end{pmatrix} \tag{3.18}$$

etc.

Finally, we are going to introduce an enlarged vector $\boldsymbol{R}$ of residual forces in the following manner:

$$\boldsymbol{R}^0 = \begin{pmatrix} {}^n\boldsymbol{f} - {}^{n+1}\boldsymbol{f} \\ {}^n\boldsymbol{b} - {}^{n+1}\boldsymbol{b} \\ {}^n\boldsymbol{b} - {}^{n+1}\boldsymbol{b} \end{pmatrix}, \tag{3.19}$$

$$\boldsymbol{R}^k = \begin{pmatrix} \boldsymbol{f}_i(\boldsymbol{u}^k) - {}^{n+1}\boldsymbol{f} + \boldsymbol{A}_e^T(\boldsymbol{\lambda}^k + \boldsymbol{\mu}^k) \\ \boldsymbol{A}_e\ \boldsymbol{u}^k - {}^{n+1}\boldsymbol{b} \\ \boldsymbol{A}_e\ \boldsymbol{u}^k - {}^{n+1}\boldsymbol{b} \end{pmatrix}, \quad k > 0. \tag{3.20}$$

After this notation is introduced, the fNR algorithm can be rewritten to incorporate a Lagrange-multiplier treatment of boundary conditions. The resulting algorithm is listed in Table 3.5.

Furthermore, in a similar way to what we saw in Section 3.4,

- replacing matrix ${}^{n+1}\boldsymbol{J}^k$ with ${}^{n+1}\boldsymbol{J}^0$ yields the *modified Newton–Raphson method*.

and

- replacing ${}^{n+1}\boldsymbol{J}^k$ with ${}^0\boldsymbol{J}$ yields the *initial stress method*.

prediction

1. Compute $\boldsymbol{J}^0$
2. Solve the linear set $\boldsymbol{J}^0\ \Delta\boldsymbol{U}^1 = -\boldsymbol{R}^0$
3. Update $\boldsymbol{U}^1 = \boldsymbol{U}^0 + \Delta\boldsymbol{U}^1$
4. Evaluate $\boldsymbol{R}^1 = \boldsymbol{R}(\boldsymbol{U}^1)$
5. Convergence control: if point $\boldsymbol{U}^1$ is good enough, exit.

corrections

$k \geq 1$

6. Compute $\boldsymbol{J}^k$
7. Solve the linear set $\boldsymbol{J}^k\ \boldsymbol{\delta U}^{k+1} = -\boldsymbol{R}^k$
8. Update $\boldsymbol{U}^{k+1} = \boldsymbol{U}^k + \boldsymbol{\delta U}^{k+1}$
9. Evaluate $\boldsymbol{R}^{k+1} = \boldsymbol{R}(\boldsymbol{U}^{k+1})$
10. Convergence control: if point $\boldsymbol{U}^{k+1}$ is good enough, exit.
11. Assign $k = k+1$ and go back to 6.

Table 3.5: Full Newton–Raphson method in a Lagrange-multiplier context.

Finally, there is a feature of vector $\boldsymbol{R}^k$ $(k > 0)$ that we will need to have in mind later on, which is that *its last two components are null*.

To prove this, we are going to focus on the fNR method, although it will be clear from the proof that the process is not affected by the choice of the stiffness matrix $\boldsymbol{K}$ at all.

1) Let us start then by proving the proposition for the first value of k.

Indeed, for $k = 1$, the second and third components of $\boldsymbol{R}^k = \boldsymbol{R}^1$ read

$$\boldsymbol{A}_e\ \boldsymbol{u}^1 - {}^{n+1}\boldsymbol{b} = \boldsymbol{A}_e\ (\boldsymbol{u}^0 + \Delta\boldsymbol{u}^1) - {}^{n+1}\boldsymbol{b} = \boldsymbol{A}_e\ \boldsymbol{u}^0 + \boldsymbol{A}_e\ \Delta\boldsymbol{u}^1 - {}^{n+1}\boldsymbol{b}. \tag{3.21}$$

As $\boldsymbol{u}^0 = {}^{n+1}\boldsymbol{u}^0 = {}^{n}\boldsymbol{u}$ is the solution for the previous increment, we have that

$$\boldsymbol{A}_e\ \boldsymbol{u}^0 = {}^{n}\boldsymbol{b}. \tag{3.22}$$

As vector $\Delta\boldsymbol{U}^1$ has been computed so as to satisfy the linear set $\boldsymbol{J}^0\Delta\boldsymbol{U}^1 = -\boldsymbol{R}^0$, we can write

$$\boldsymbol{A}_e\ \Delta\boldsymbol{u}^1 = {}^{n+1}\boldsymbol{b} - {}^{n}\boldsymbol{b}. \tag{3.23}$$

If we now substitute equations (3.22) and (3.23) into equation (3.21), we obtain

$$\boldsymbol{A}_e\ \boldsymbol{u}^1 - {}^{n+1}\boldsymbol{b} = \boldsymbol{A}_e\ \boldsymbol{u}^0 + \boldsymbol{A}_e\ \Delta\boldsymbol{u}^1 - {}^{n+1}\boldsymbol{b} = {}^{n}\boldsymbol{b} + ({}^{n+1}\boldsymbol{b} - {}^{n}\boldsymbol{b}) - {}^{n+1}\boldsymbol{b} = \boldsymbol{0}. \tag{3.24}$$

Therefore, when computing $\boldsymbol{\delta U}^2$ as a solution for

$$\boldsymbol{J}^1\boldsymbol{\delta U}^2 = -\boldsymbol{R}^1,$$

we see that $\boldsymbol{\delta u}^2$ must verify the homogeneous equation

$$\boldsymbol{A}_e\ \boldsymbol{\delta u}^2 = \boldsymbol{0}, \tag{3.25}$$

in other words, we see that *the correction vector* $\boldsymbol{\delta u}^2$ *satisfies null boundary conditions*.

2) For $k = 2$, we have that

$$\boldsymbol{A}_e\ \boldsymbol{u}^2 - {}^{n+1}\boldsymbol{b} = \boldsymbol{A}_e\ (\boldsymbol{u}^1 + \boldsymbol{\delta u}^2) - {}^{n+1}\boldsymbol{b} = \boldsymbol{A}_e\ \boldsymbol{u}^1 + \boldsymbol{A}_e\ \boldsymbol{\delta u}^2 - {}^{n+1}\boldsymbol{b}. \tag{3.26}$$

Equations (3.24) and (3.25) can now be subtituted into equation (3.26) yielding

$$\boldsymbol{A}_e\ \boldsymbol{u}^2 - {}^{n+1}\boldsymbol{b} = \boldsymbol{A}_e\ \boldsymbol{u}^1 + \boldsymbol{A}_e\ \boldsymbol{\delta u}^2 - {}^{n+1}\boldsymbol{b} = \boldsymbol{0}. \tag{3.27}$$

Thus, the second and third components of $\boldsymbol{R}^2$ are also null, which means that $\boldsymbol{\delta u}^3$ also satisfies null boundary conditions.

If we repeat the same process for the next iterations, we can enunciate the two general conclusions of Table 3.6.

Conclusions

1. The vector of out-of-balance forces $\boldsymbol{R}^k$ can be written, for $k > 0$, as

$$\boldsymbol{R}^k = \begin{pmatrix} \boldsymbol{f}_i(\boldsymbol{u}^k) - {}^{n+1}\boldsymbol{f} + \boldsymbol{A}_e^T(\boldsymbol{\lambda}^k + \boldsymbol{\mu}^k) \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}, \quad k > 0. \tag{3.28}$$

2. Therefore, the correction vector $\boldsymbol{\delta u}^k$ $(k \geq 2)$ satisfies homogeneous boundary conditions:

$$\boldsymbol{A}_e\, \boldsymbol{\delta u}^k = \mathbf{0}, \quad k \geq 2 \tag{3.29}$$

Table 3.6: Remarks on NR variables in a Lagrange-multiplier context.

Chapter 4

Quasi–Newton methods

4.1 Motivation and theory

In Chapter 3, we have presented the classical Newton–Raphson methods. First, the *full Newton–Raphson method*, which provides second order convergence (*i.e.*, reduced number of iterations) in most of the cases where tangent stiffness matrices are available, but requires in return the computation and factorisation of a tangent matrix at the beginning of every iteration.

On the other hand, the *modified Newton–Raphson* and *initial stress* methods require a lower computational cost per iteration as far as stiffness matrices are concerned. However, their convergence rates are also lower, which means that more iterations will probably be needed, with the associated increase in the total cost; moreover, the process may even fail to converge.

Naturally, we would like to have a method with the convergence behaviour of fNR at a computational cost similar to the mNR technique. Quasi–Newton (QN) methods were designed for this purpose. These methods, which have their origin in the field of unconstrained optimization, are generalizations to n-dimensional

problems of the well-known "secant method" for solving roots of *one* nonlinear equation. In fact, the choice of a *secant* iteration matrix instead of a tangent stiffness matrix ensures the two following properties:

- the secant stiffness matrix *resembles* the tangent stiffness matrix
- but a strict *re-computation* of this secant stiffness matrix is not required at all.

All Quasi–Newton methods use secant stiffness matrices, but, conversely to one-dimensional problems, whereas the tangent stiffness matrix is unique, the secant matrix is not. Consequently, a whole number of different Quasi–Newton methods can be produced by employing *different* secant stiffness matrices.

Before describing some of those methods, we prefer, first, to formalize the unifying feature of all Quasi-Newton techniques: they all use secant stiffness matrices.

Assume we have just completed iteration k belonging of load step n, Figure 4.1.

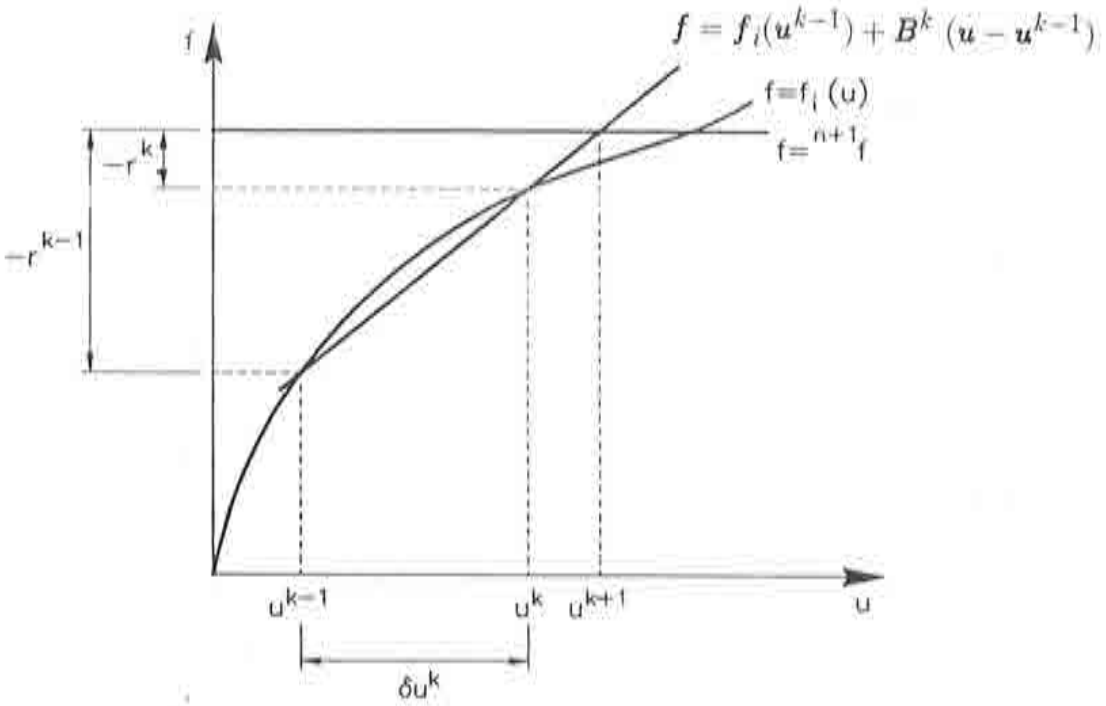


Figure 4.1: Performance of a generic Quasi–Newton method.

As usual, we assume that boundary conditions are introduced in some way into the nonlinear set of equations that we are trying to solve. Later on (see Section 4.3), we will write specific algorithms for the case where Lagrange multipliers are used to treat boundary conditions.

Points $\boldsymbol{u}^{k-1}$, $\boldsymbol{u}^k$ in Figure 4.1 are known from the previous iterations. Consequently, the vectors of residual forces $\boldsymbol{r}^{k-1}$ and $\boldsymbol{r}^k$ are also known.

Imagine for a moment that we are dealing with a one-dimensional problem. In this case, we will be able to compute the slope B^k of the secant line that passes through points $(u^{k-1}, f_i(u^{k-1}))$ and $(u^k, f_i(u^k))$, see Figure 4.1, directly as

$$B^k = \frac{r^k - r^{k-1}}{\delta u^k}. \tag{4.1}$$

In an n-dimensional problem ($n > 1$), and for a generic value of n, we rewrite equation (4.1) into

$$\boldsymbol{B}^k \, \boldsymbol{\delta u}^k = \boldsymbol{r}^k - \boldsymbol{r}^{k-1}. \tag{4.2}$$

Equation (4.2) is known as the *Quasi–Newton equation*, which characterises the *secant stiffness matrix* $\boldsymbol{B}^k$ for a general nonlinear problem. It is important to notice that in this set of equations, the unknown is the matrix, not the vector. That is, if a total of n equations are listed in (4.2), n^2 unknowns will be obtained. Therefore, as it was briefly indicated at the beginning of this section, the secant matrix that passes through two specified points *is not unique*, and there is a need to impose some additional equations on $\boldsymbol{B}^k$ ($n^2 - n$ equations, to be precise) in order to be able to determine it.

Defining

$$\boldsymbol{y}^{k-1} = \boldsymbol{r}^k - \boldsymbol{r}^{k-1} \tag{4.3}$$

$$\boldsymbol{s}^{k-1} = \boldsymbol{\delta u}^k = \boldsymbol{u}^k - \boldsymbol{u}^{k-1} \tag{4.4}$$

we will be able to write equation (4.2) in its most common form

$$\boldsymbol{B}^k \, \boldsymbol{s}^{k-1} = \boldsymbol{y}^{k-1}. \tag{4.5}$$

From now on we will continue to use the notation $[\boldsymbol{s}, \boldsymbol{y}]$, which is widely accepted in the Quasi–Newton context, rather than our original notation, $[\boldsymbol{u}, \boldsymbol{r}]$. After the methods have been explained, we will go back to the $[\boldsymbol{u}, \boldsymbol{r}]$ formulation.

Once a secant matrix $\boldsymbol{B}^k$ is computed, vector $\boldsymbol{u}^{k+1}$ will be obtained as shown in Figure 4.1, *i.e.*, by solving the linear set

$$\boldsymbol{B}^k \; \boldsymbol{s}^k = -\boldsymbol{r}^k \tag{4.6}$$

and updating subsequently

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \boldsymbol{s}^k.$$

A new secant matrix $\boldsymbol{B}^{k+1}$ that passes through points $(\boldsymbol{u}^k, \boldsymbol{f}_i(\boldsymbol{u}^k))$ and $(\boldsymbol{u}^{k+1}, \boldsymbol{f}_i(\boldsymbol{u}^{k+1}))$ can then be computed, yielding a new approximation $\boldsymbol{u}^{k+2}$, and so on.

This "direct" extension of the one-dimensional secant method into n-dimensional nonlinear problems induces the computation of the "generalized" slope, *i.e.* matrix $\boldsymbol{B}$, which is an approximation of the tangent matrix. This strategy defines the *Direct* Quasi-Newton family, in opposition to the *Inverse* Quasi-Newton techniques which are concerned with the approximation of the inverse of the tangent matrix, namely $\boldsymbol{H} = \boldsymbol{B}^{-1}$. In fact, the Quasi-Newton equation (4.5) can also be written as

$$\boldsymbol{H}^k \; \boldsymbol{y}^{k-1} = \boldsymbol{s}^{k-1}, \tag{4.7}$$

which, together with some extra conditions, enables the determination of matrix $\boldsymbol{H}^k$. Inverse Quasi-Newton are usually preferred, because equation (4.6) can be transformed into

$$\boldsymbol{s}^k = -\boldsymbol{H}^k \; \boldsymbol{r}^k, \tag{4.8}$$

so that there is no need for solving linear sets of equations.

Some Quasi–Newton methods –direct and inverse– are discussed in the following section.

4.2 Discussion of various Quasi-Newton methods. Algorithms for Inverse Broyden and BFGS

The Broyden method

Let us consider the direct form of the Quasi–Newton equation that was introduced in (4.5),

$$\boldsymbol{B}^k \; \boldsymbol{s}^{k-1} = \boldsymbol{y}^{k-1}.$$

As it was already explained in Section 4.1, if n is the number of equations in (4.5), then $n^2 - n$ additional equations on $\boldsymbol{B}^k$ are needed to determine its n^2 components.

Notice that condition (4.5) *defines completely the behaviour of* $\boldsymbol{B}^k$ *along the direction given by* $\boldsymbol{s}^{k-1}$. Moreover, it is the only new information about $\boldsymbol{B}^k$ with respect to $\boldsymbol{B}^{k-1}$. For this reason, Broyden suggested that matrices $\boldsymbol{B}^k$ and $\boldsymbol{B}^{k-1}$ should have the same behaviour in any direction except $\boldsymbol{s}^{k-1}$. To accomplish this, the Broyden method requires that the secant matrix $\boldsymbol{B}^k$ behave in the same way as $\boldsymbol{B}^{k-1}$ on the *orthogonal complement* of $\boldsymbol{s}^{k-1}$

$$\boldsymbol{B}^k \; \boldsymbol{z} = \boldsymbol{B}^{k-1} \; \boldsymbol{z} \qquad \text{for all } \boldsymbol{z} \text{ such that } \quad \boldsymbol{z}^T \; \boldsymbol{s}^{k-1} = 0. \tag{4.9}$$

It can be proved, [11,14], that (4.5) together with (4.9) determines one and only one matrix $\boldsymbol{B}^k$ in terms of $\boldsymbol{B}^{k-1}$, following the expression

$$\boldsymbol{B}^k = \boldsymbol{B}^{k-1} + \frac{(\boldsymbol{y}^{k-1} - \boldsymbol{B}^{k-1} \; \boldsymbol{s}^{k-1}) \; (\boldsymbol{s}^{k-1})^T}{(\boldsymbol{s}^{k-1})^T \; \boldsymbol{s}^{k-1}}. \tag{4.10}$$

It is interesting to remark at this point that equation (4.10) describes a simple update of $\boldsymbol{B}^k$ after $\boldsymbol{B}^{k-1}$.

Notice that defining vectors $\boldsymbol{v}^{k-1}$ and $\boldsymbol{w}^{k-1}$ as

$$\boldsymbol{v}^{k-1} = \frac{\boldsymbol{y}^{k-1} - \boldsymbol{B}^{k-1} \; \boldsymbol{s}^{k-1}}{(\boldsymbol{s}^{k-1})^T \; \boldsymbol{s}^{k-1}}$$

$$\boldsymbol{w}^{k-1} = \boldsymbol{s}^{k-1},$$

we can rewrite equation (4.10) as

$$\boldsymbol{B}^k = \boldsymbol{B}^{k-1} + \boldsymbol{v}^{k-1} \, (\boldsymbol{w}^{k-1})^T, \tag{4.11}$$

which is the common expression of the *rank-one update*, so termed because the modification $\boldsymbol{v}^{k-1} \, (\boldsymbol{w}^{k-1})^T$ is a rank-one matrix. Equation (4.11) must be initialized with a certain $\boldsymbol{B}^0$; some options are discussed later on.

The inverse version of the Broyden method

Once $\boldsymbol{B}^k$ is computed from equation (4.10), we need to solve the linear set (4.6)

$$\boldsymbol{B}^k \; \boldsymbol{s}^k = -\boldsymbol{r}^k$$

to obtain vector $\boldsymbol{s}^k$. This operation can be avoided if an expression for $\boldsymbol{H}^k$ rather than for $\boldsymbol{B}^k$ is obtained. By doing so, we can compute $\boldsymbol{s}^k$ directly following equation (4.8)

$$\boldsymbol{H}^k \; \boldsymbol{r}^k = -\boldsymbol{s}^k. \tag{4.12}$$

In order to obtain an expression for $\boldsymbol{H}^k$ from $\boldsymbol{H}^{k-1}$, the Sherman & Morrison lemma must be employed, [25].

Lemma (Sherman & Morrison)

Let $\boldsymbol{v}$,$\boldsymbol{w} \in \mathbb{R}^n$ and assume that $\boldsymbol{B}$ is a real n-dimensional nonsingular matrix.

Then, 1) $\boldsymbol{B} + \boldsymbol{v} \; \boldsymbol{w}^T$ is nonsingular if and only if

$$\sigma = 1 + \boldsymbol{w}^T \; \boldsymbol{B}^{-1} \; \boldsymbol{v} \neq 0.$$

2) If $\sigma \neq 0$, then the inverse of matrix $\boldsymbol{B} + \boldsymbol{v} \; \boldsymbol{w}^T$ can be computed as

$$(\boldsymbol{B} + \boldsymbol{v} \; \boldsymbol{w}^T)^{-1} = \boldsymbol{B}^{-1} - \frac{1}{\sigma} \; \boldsymbol{B}^{-1} \; \boldsymbol{v} \; \boldsymbol{w}^T \; \boldsymbol{B}^{-1}.$$

Proof: See [25].

From the lemma and since $(\boldsymbol{B}^{k-1})^{-1} = \boldsymbol{H}^{k-1}$, we can write $(\boldsymbol{B}^{k})^{-1} = \boldsymbol{H}^{k}$ as

$$\boldsymbol{H}^{k} = \boldsymbol{H}^{k-1} + \frac{(\boldsymbol{s}^{k-1} - \boldsymbol{H}^{k-1}\,\boldsymbol{y}^{k-1})\,(\boldsymbol{s}^{k-1})^{T}\,\boldsymbol{H}^{k-1}}{(\boldsymbol{s}^{k-1})^{T}\,(\boldsymbol{H}^{k-1}\,\boldsymbol{y}^{k-1})}, \tag{4.13}$$

which can also be expressed as

$$\boldsymbol{H}^{k} = \left[\boldsymbol{I} + \frac{(\boldsymbol{s}^{k-1} - \boldsymbol{H}^{k-1}\,\boldsymbol{y}^{k-1})}{(\boldsymbol{s}^{k-1})^{T}\,(\boldsymbol{H}^{k-1}\,\boldsymbol{y}^{k-1})}\,(\boldsymbol{s}^{k-1})^{T}\right]\,\boldsymbol{H}^{k-1}, \tag{4.14}$$

where $\boldsymbol{I}$ denotes the n-dimensional identity matrix. Equation (4.14), which must be initialized with a certain $\boldsymbol{H}^{0}$, is the fundamental relation in the Inverse Broyden method. Notice that, for this inverse QN method, the update formula for $\boldsymbol{H}^{k}$ is obtained by inverting explicitly the update formula for $\boldsymbol{B}^{k}$ rather than by using the inverse Quasi–Newton equation and imposing additional conditions on $\boldsymbol{H}^{k}$.

From an algorithmic viewpoint, instead of employing equation (4.14), it is more convenient, [12,27], to relate $\boldsymbol{H}^{k}$ to $\boldsymbol{H}^{0}$ through

$$\boldsymbol{H}^{k} = \left[\boldsymbol{I} + \boldsymbol{\omega}^{k}\,(\boldsymbol{s}^{k-1})^{T}\right]\,\left[\boldsymbol{I} + \boldsymbol{\omega}^{k-1}\,(\boldsymbol{s}^{k-2})^{T}\right]\cdots\left[\boldsymbol{I} + \boldsymbol{\omega}^{1}\,(\boldsymbol{s}^{0})^{T}\right]\boldsymbol{H}^{0} =$$

$$= \prod_{i=k}^{1}\left[\boldsymbol{I} + \boldsymbol{\omega}^{i}\,(\boldsymbol{s}^{i-1})^{T}\right]\boldsymbol{H}^{0}, \tag{4.15}$$

where the auxiliar vectors $\boldsymbol{\omega}^{i+1}$ are defined as

$$\boldsymbol{\omega}^{i+1} = \frac{(\boldsymbol{s}^{i} - \boldsymbol{H}^{i}\,\boldsymbol{y}^{i})}{(\boldsymbol{s}^{i})^{T}\,(\boldsymbol{H}^{i}\,\boldsymbol{y}^{i})}, \quad i = 0, \ldots, k-1. \tag{4.16}$$

Before discussing the algorithmic advantages of equation (4.15), it is convenient to deal with the initialization of the Quasi-Newton updates. As commented previously, it is necessary to initialize any Quasi-Newton method with a matrix $\boldsymbol{B}^{0}$ (direct QN) or $\boldsymbol{H}^{0}$ (inverse QN). The simplest choice is $\boldsymbol{B}^{0} = \boldsymbol{I}$ or $\boldsymbol{H}^{0} = \boldsymbol{I}$, where $\boldsymbol{I}$ is the n-dimensional identity matrix.

Nevertheless, if the computation of tangent stiffness matrices is available, it seems that a better choice could be

$$\boldsymbol{B}^{0} = \boldsymbol{K}^{0}, \tag{4.17}$$

that is, initializing the algorithm with a true tangent matrix. This choice of the initial secant matrix is expected to yield *better* secant matrices $\boldsymbol{B}^k$ than $\boldsymbol{B}^0 = \boldsymbol{I}$ (in the sense that these matrices $\boldsymbol{B}^k$ are going to look more like tangent stiffness matrices than the ones we would obtain after $\boldsymbol{B}^0 = \boldsymbol{I}$).

Assume then that $\boldsymbol{B}^0$ is the true tangent stiffness matrix at the beginning of the increment. If we want to use the inverse version of the Broyden method, equation (4.15), matrix $\boldsymbol{K}^0$ should be inverted in order to define $\boldsymbol{H}^0$. A more efficient strategy is to rewrite the computation of the prediction $\boldsymbol{s}^0$

$$\boldsymbol{s}^0 = \Delta \boldsymbol{u}^1 = -\boldsymbol{H}^0 \, \boldsymbol{r}^0$$

into the linear set

$$\boldsymbol{K}^0 \, \boldsymbol{s}^0 = \boldsymbol{K}^0 \, \Delta \boldsymbol{u}^1 = -\boldsymbol{r}^0.$$

At this point, the algorithmic convenience of equation (4.15) becomes clear: since it relates $\boldsymbol{H}^k$ to $\boldsymbol{H}^0$, the computation af a generic correction vector $\boldsymbol{s}^k$, equation (4.8), involves a matrix–vector product of the form

$$\boldsymbol{x} = -\boldsymbol{H}^0 \, \boldsymbol{r}^k,$$

which can be transformed into the linear set

$$\boldsymbol{K}^0 \, \boldsymbol{x} = -\boldsymbol{r}^k,$$

where *the same matrix*, $\boldsymbol{K}^0$, is employed at every iteration. This justifies the use of equation (4.15) instead of the basic equation (4.14).

The final algorithm for this version of the inverse Broyden method is presented in Table 4.1. In this algorithm we have recovered the original formulation $[\boldsymbol{u}, \boldsymbol{r}]$ through equations (4.3) and (4.4). We will be assuming, as usual, that $\delta \boldsymbol{u}^1 = \Delta \boldsymbol{u}^1$. Also, two auxiliary vectors $\boldsymbol{t}$ and $\boldsymbol{v}$ are defined.

1. Compute $\boldsymbol{K}^0$
2. Solve the linear set $\boldsymbol{K}^0\ \Delta\boldsymbol{u}^1 = -\boldsymbol{r}^0$
3. Update $\boldsymbol{u}^1 = \boldsymbol{u}^0 + \Delta\boldsymbol{u}^1$
4. Evaluate $\boldsymbol{r}^1 = \boldsymbol{r}(\boldsymbol{u}^1)$
5. Convergence control: if point $\boldsymbol{u}^1$ is good enough, exit.

   $k \geq 1$
6. Solve the linear set $\boldsymbol{K}^0\ \boldsymbol{v} = -\boldsymbol{r}^k$
7. $k = 1$ assign $\boldsymbol{t} = \boldsymbol{v}$

   $k > 1$ compute $\boldsymbol{t} = \prod_{i=k-1}^{1}(\boldsymbol{I} + \boldsymbol{\omega}^i\ (\boldsymbol{\delta u}^i)^T)\ \boldsymbol{v}$
8. Compute and store $\boldsymbol{\omega}^k = \frac{1}{(\boldsymbol{\delta u}^k)^T\ (\boldsymbol{\delta u}^k - \boldsymbol{t})}\ \boldsymbol{t}$
9. Compute and store $\boldsymbol{\delta u}^{k+1} = \boldsymbol{t} + ((\boldsymbol{\delta u}^k)^T\ \boldsymbol{t})\ \ \boldsymbol{\omega}^k$
10. Update $\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \boldsymbol{\delta u}^{k+1}$
11. Evaluate $\boldsymbol{r}^{k+1} = \boldsymbol{r}(\boldsymbol{u}^{k+1})$
12. Convergence control: if point $\boldsymbol{u}^{k+1}$ is good enough, exit.
13. Assign $k = k + 1$ and go back to 6.

Table 4.1: Algorithm for the Inverse Broyden method.

Remark 1. All linear sets are solved with matrix $\boldsymbol{K}^0$, see step 6. Thus, only one tangent stiffness matrix needs to be computed and factorised per increment.

Remark 2. Step 7 of the algorithm does *not* require the computation of matrix $\prod_{i=k-1}^{1}(\boldsymbol{I} + \boldsymbol{\omega}^i\ (\boldsymbol{\delta u}^i)^T)$, since the matrix–vector products of the form $(\boldsymbol{I} + \boldsymbol{a}\ \boldsymbol{b}^T)\boldsymbol{c}$ can be computed as $\boldsymbol{c} + (\boldsymbol{b}^T\boldsymbol{c})\boldsymbol{a}$. In fact, apart from the linear sets of steps 2. and 6., all the operations involved in Table 4.1 are vectorial operations.

Remark 3. For each iteration k, the computational cost consists approximately, [12,27], of

- evaluating one vector of out-of-balance forces and solving one linear set (the same as in mNR)
- recovering $2k$ vectors from storage ($\boldsymbol{\omega}^i$, $\boldsymbol{\delta u}^i$ for $i = 1, \ldots, k$).
- computing $2k$ scalar products.

Therefore, *the total cost of iteration k increases with k*. This means that just a few iterations per increment will be permitted for this method to be competitive.

Remark 4. Under certain conditions, [11], this method can be proved to have a *superlinear rate of convergence*, *i.e.*, a succession $\{\gamma_k\}$ such that $\lim_{k\to\infty} \gamma_k = 0$ can be found for which

$$\|\boldsymbol{u}^{k+1} - \boldsymbol{\alpha}\| \leq \gamma_k \, \|\boldsymbol{u}^k - \boldsymbol{\alpha}\|,$$

where $\boldsymbol{\alpha}$ denotes the solution corresponding to the current increment.

Symmetrical rank-one update

As we have already indicated, in general we will want to iterate with a secant matrix that is as similar to the tangent stiffness matrix as possible. In many applications, tangent matrices are symmetrical, as will be shown in the numerical examples. It seems thus interesting to begin with a symmetrical $\boldsymbol{B}^0$ and transfer this property at every iteration. Therefore, to impose *hereditary symmetry* as

$$\boldsymbol{B}^{k-1} \text{ symmetrical} \Longrightarrow \boldsymbol{B}^k \text{ symmetrical} \tag{4.18}$$

seems a reasonable choice.

It can be proved, [11], that the direct Quasi–Newton equation (4.5) together with (4.18) leads to the following update formula for $\boldsymbol{B}^k$:

$$\boldsymbol{B}^k = \boldsymbol{B}^{k-1} + \frac{(\boldsymbol{y}^{k-1} - \boldsymbol{B}^{k-1}\,\boldsymbol{s}^{k-1})\,(\boldsymbol{y}^{k-1} - \boldsymbol{B}^{k-1}\,\boldsymbol{s}^{k-1})^T}{(\boldsymbol{y}^{k-1} - \boldsymbol{B}^{k-1}\,\boldsymbol{s}^{k-1})^T\,\boldsymbol{s}^{k-1}}. \tag{4.19}$$

Equation (4.19) is known as the *symmetrical rank-one formula*. According to [11], this method shows a poor perfomance and is not frequently employed.

Rank-two updates

Direct rank-two updates

In many applications, the tangent stiffness matrices are not only symmetrical but also positive definite. Thus, it is a reasonable strategy to initialize a Quasi-Newton method with a symmetrical positive definite (SPD) $\boldsymbol{B}^0$ and transfer these properties at every iteration. That is, we want to satisfy the Quasi-Newton equation (4.5), the symmetry condition (4.18) and

$$\boldsymbol{B}^{k-1} \text{ positive definite} \Longrightarrow \boldsymbol{B}^k \text{ positive definite.} \tag{4.20}$$

The only rank-one update with hereditary symmetry is the one shown in equation (4.19), so there is a need for more complicated updates if condition (4.20) must be verified as well.

Rank-two updates, where $\boldsymbol{B}^k$ is obtained by adding to $\boldsymbol{B}^{k-1}$ *two* rank one matrices instead of one, fulfill this need. In particular, a family of rank-two updates is defined by

$$\begin{aligned}\boldsymbol{B}^k = \boldsymbol{B}^{k-1} + & \frac{(\boldsymbol{y}^{k-1} - \boldsymbol{B}^{k-1}\,\boldsymbol{s}^{k-1})\,(\boldsymbol{c}^{k-1})^T + \boldsymbol{c}^{k-1}\,(\boldsymbol{y}^{k-1} - \boldsymbol{B}^{k-1}\,\boldsymbol{s}^{k-1})^T}{(\boldsymbol{s}^{k-1})^T\,\boldsymbol{c}^{k-1}} \\ & - \frac{(\boldsymbol{y}^{k-1} - \boldsymbol{B}^{k-1}\,\boldsymbol{s}^{k-1})^T\,\boldsymbol{s}^{k-1}}{((\boldsymbol{c}^{k-1})^T\,\boldsymbol{s}^{k-1})^2}\,\boldsymbol{c}^{k-1}\,(\boldsymbol{c}^{k-1})^T,\end{aligned} \tag{4.21}$$

where $\boldsymbol{c}^{k-1}$ represents any vector belonging to $\mathbb{R}^n$. Equation (4.21) satisfies both the Quasi-Newton equation (4.5) and the hereditary symmetry condition (4.18), [11]. It has the interesting property that vector $\boldsymbol{c}^{k-1}$ has not been chosen yet (for instance, for $\boldsymbol{c}^{k-1} = \boldsymbol{y}^{k-1} - \boldsymbol{B}^{k-1}\,\boldsymbol{s}^{k-1}$ we recover the rank-one symmetrical update, (4.19)).

If it is required for matrix $\boldsymbol{B}^k$ in (4.21) to be positive definite if $\boldsymbol{B}^{k-1}$ is positive definite, that is, if conditions (4.5), (4.18) and (4.20) must be satisfied simultaneously, *a natural choice* for $\boldsymbol{c}^{k-1}$ is $\boldsymbol{y}^{k-1}$, [11].

Substituting

$$\boldsymbol{c}^{k-1} = \boldsymbol{y}^{k-1} \tag{4.22}$$

into equation (4.21) and rearranging the resulting expression of $\boldsymbol{B}^k$, the Davidon–Fletcher–Powell compact update formula is obtained, [14],

$$\boldsymbol{B}^k_{DFP} = \left[\boldsymbol{I} - \frac{\boldsymbol{y}^{k-1}\ (\boldsymbol{s}^{k-1})^T}{(\boldsymbol{s}^{k-1})^T\ \boldsymbol{y}^{k-1}}\right] \boldsymbol{B}^{k-1} \left[\boldsymbol{I} - \frac{\boldsymbol{s}^{k-1}\ (\boldsymbol{y}^{k-1})^T}{(\boldsymbol{s}^{k-1})^T\ \boldsymbol{y}^{k-1}}\right] + \frac{\boldsymbol{y}^{k-1}\ (\boldsymbol{y}^{k-1})^T}{(\boldsymbol{s}^{k-1})^T\ \boldsymbol{y}^{k-1}}, \tag{4.23}$$

which describes the *DFP method*.

Inverse rank-two updates

In an inverse Quasi-Newton context, it is possible to impose hereditary symmetry and positive definiteness on the *inverse* of matrix $\boldsymbol{B}^k$, *i.e.*,

$$\boldsymbol{H}^{k-1} \text{ symmetrical} \Longrightarrow \boldsymbol{H}^k \text{ symmetrical} \tag{4.24}$$

$$\boldsymbol{H}^{k-1} \text{ positive definite} \Longrightarrow \boldsymbol{H}^k \text{ positive definite} \tag{4.25}$$

After some algebra, [4], the Broyden–Fletcher–Goldfarb–Shanno update formula for $\boldsymbol{H}$ can be obtained

$$\boldsymbol{H}^k_{BFGS} = \left[\boldsymbol{I} - \frac{\boldsymbol{s}^{k-1}\ (\boldsymbol{y}^{k-1})^T}{(\boldsymbol{y}^{k-1})^T\ \boldsymbol{s}^{k-1}}\right] \boldsymbol{H}^{k-1} \left[\boldsymbol{I} - \frac{\boldsymbol{y}^{k-1}\ (\boldsymbol{s}^{k-1})^T}{(\boldsymbol{y}^{k-1})^T\ \boldsymbol{s}^{k-1}}\right] + \frac{\boldsymbol{s}^{k-1}\ (\boldsymbol{s}^{k-1})^T}{(\boldsymbol{y}^{k-1})^T\ \boldsymbol{s}^{k-1}}, \tag{4.26}$$

which describes the *BFGS method*.

It is interesting to remark, by comparing equations (4.23) and (4.26), that one can be obtained from the other simply by interchanging

$$\boldsymbol{s} \longleftrightarrow \boldsymbol{y}$$

$$\boldsymbol{B} \longleftrightarrow \boldsymbol{H}$$

In this sense, equations (4.23) and (4.26) are said to be *dual* or *complementary*.

The detailed algorithm for this method, [4,18,27], is listed in Table 4.2.

1. Compute $\boldsymbol{K}^0$
2. Solve the linear set $\boldsymbol{K}^0\ \Delta\boldsymbol{u}^1 = -\boldsymbol{r}^0$
3. Update $\boldsymbol{u}^1 = \boldsymbol{u}^0 + \Delta\boldsymbol{u}^1$
4. Evaluate $\boldsymbol{r}^1 = \boldsymbol{r}(\boldsymbol{u}^1)$
5. Convergence control: if point $\boldsymbol{u}^1$ is good enough, exit.

   $k \geq 1$
6. Compute and store $\omega^k = \frac{\delta\boldsymbol{u}^k}{(\delta\boldsymbol{u}^k)^T\ (\boldsymbol{r}^{k-1} - \boldsymbol{r}^k)}$
7. Compute and store $\boldsymbol{v}^k = \boldsymbol{r}^k - \left[1 + \sqrt{\frac{(\delta\boldsymbol{u}^k)^T\ (\boldsymbol{r}^{k-1} - \boldsymbol{r}^k)}{(\delta\boldsymbol{u}^k)^T \boldsymbol{r}^{k-1}}}\right] \boldsymbol{r}^{k-1}$
8. Compute $\boldsymbol{p} = \prod_{i=1}^{k} (\boldsymbol{I} + \boldsymbol{v}^i\ (\omega^i)^T)\ \boldsymbol{r}^k$
9. Solve the linear set $\boldsymbol{K}^0\ \boldsymbol{q} = -\boldsymbol{p}$
10. Compute and store $\delta\boldsymbol{u}^{k+1} = \prod_{i=k}^{1} (\boldsymbol{I} + \omega^i\ (\boldsymbol{v}^i)^T)\ \boldsymbol{q}$
11. Update $\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \delta\boldsymbol{u}^{k+1}$
12. Evaluate $\boldsymbol{r}^{k+1} = \boldsymbol{r}(\boldsymbol{u}^{k+1})$
13. Convergence control: if point $\boldsymbol{u}^{k+1}$ is good enough, exit.
14. Assign $k = k + 1$ and go back to 6.

Table 4.2: Algorithm for the BFGS method.

Remark 1. As explained in Remark 1 for the Inverse Broyden method, step 10 does *not* require the computation of matrix $\boldsymbol{I} + \boldsymbol{\omega}^i\ (\delta\boldsymbol{v}^i)^T$.

Remark 2. For each iteration k, we need to solve one linear set and evaluate one vector of out-of-balance forces, recover $2k$ vectors from storage and compute $4k$ scalar products, [4,27].

The cost of a BFGS iteration is therefore higher than that of a Broyden iteration. However, if tangent stiffness matrices are SPD, this difference is supposed to be compensated by the fact that BFGS iteration matrices, which are SPD, are better –that is, more similar to the true tangent matrix– than Broyden matrices (nor symmetrical neither positive definite).

Remark 3. The BFGS method, when accelerated with a certain line-search procedure, shows the same kind of superlinear convergence as the Broyden method, [11].

4.3 Getting Quasi–Newtons to work in Castem — total and partial approaches

If linear boundary conditions are treated via Lagrange multipliers, two options are possible when developing Quasi-Newton methods:

Option T (Total approach): use a secant approximation $\boldsymbol{B}^k$ of the Jacobian matrix $\boldsymbol{J}^k$:

$$\boldsymbol{B}^k \approx \boldsymbol{J}^k. \tag{4.27}$$

Option T describes the natural way in which Quasi–Newton methods approach nonlinear problems, as it approximates directly the whole Jacobian matrix.

Option P (Partial approach): compute a secant approximation $\hat{\boldsymbol{B}}^k$ to $\boldsymbol{K}^k$

$$\hat{\boldsymbol{B}}^k \approx \boldsymbol{K}^k, \tag{4.28}$$

and then employ *constant* matrices $\boldsymbol{A}_c$ and $\boldsymbol{I}$ to build $\boldsymbol{B}^k$ as

$$\boldsymbol{B}^k = \begin{pmatrix} \hat{\boldsymbol{B}}^k & \boldsymbol{A}_c^T & \boldsymbol{A}_c^T \\ \boldsymbol{A}_c & \boldsymbol{I} & -\boldsymbol{I} \\ \boldsymbol{A}_c & -\boldsymbol{I} & \boldsymbol{I} \end{pmatrix} \approx \boldsymbol{J}^k. \tag{4.29}$$

Option P takes profit of the fact that all submatrices in $\boldsymbol{J}$ are constant except for matrix $\boldsymbol{K}$; in other words, it concentrates all efforts on the only nonlinear part of

the Jacobian matrix $\boldsymbol{J}$. Of course, if constraints were not linear, option P would have to be discarded.

The following issue now arises: do options P and T lead to different or to the same methods? In other words, is the linearity of constraints automatically taken into account, in a such a way that the strategy (4.27) results in an approximation of the type (4.29)?

In the remaining part of this chapter, it will be shown that options T and P yield different methods for the rank-one updates (Broyden method) and the same method for the rank-two updates (DFP and BFGS). Moreover, when two different methods result for options P and T, the numerical tests show that the partial approach has a slightly better perfomance in terms of number of iterations and computing time.

To develop these two approaches, we need some of the notation that was introduced in Section 3.5, namely the augmented vector of unknowns

$$\boldsymbol{U}^k = \begin{pmatrix} \boldsymbol{u}^k \\ \boldsymbol{\lambda}^k \\ \boldsymbol{\mu}^k \end{pmatrix}, \tag{4.30}$$

and the augmented vector of residual forces:

$$\boldsymbol{R}^0 = \begin{pmatrix} {}^n\boldsymbol{f} - {}^{n+1}\boldsymbol{f} \\ {}^n\boldsymbol{b} - {}^{n+1}\boldsymbol{b} \\ {}^n\boldsymbol{b} - {}^{n+1}\boldsymbol{b} \end{pmatrix}, \tag{4.31}$$

$$\boldsymbol{R}^k = \begin{pmatrix} \boldsymbol{f}_i(\boldsymbol{u}^k) - {}^{n+1}\boldsymbol{f} + \boldsymbol{A}_e^T(\boldsymbol{\lambda}^k + \boldsymbol{\mu}^k) \\ \boldsymbol{A}_e\, \boldsymbol{u}^k - {}^{n+1}\boldsymbol{b} = \boldsymbol{0} \\ \boldsymbol{A}_e\, \boldsymbol{u}^k - {}^{n+1}\boldsymbol{b} = \boldsymbol{0} \end{pmatrix}, \quad k > 0. \tag{4.32}$$

4.3.1 Total and partial versions of the Inverse Broyden method

Option T (Total approach)

As we did in Section 4.2, we start by discussing the direct version of the Broyden method. After an update formula for $\boldsymbol{B}^k$ has been obtained, we will invert this expression to obtain a formula for $\boldsymbol{H}^k$.

Recall, first, the equations that define the direct Broyden method. For this option, since we are trying to approximate the Jacobian matrix $\boldsymbol{J}^k$, equation (3.15), we are dealing with augmented vectors. Hence, we need the following version of the Quasi–Newton equation:

$$\boldsymbol{B}^k\ \boldsymbol{\delta U}^k = \boldsymbol{R}^k - \boldsymbol{R}^{k-1} \tag{4.33}$$

and the Broyden condition given by

$$\boldsymbol{B}^k\ \boldsymbol{Z} = \boldsymbol{B}^{k-1}\ \boldsymbol{Z} \qquad \text{for all } \boldsymbol{Z} \text{ such that } \quad \boldsymbol{Z}^T\ \boldsymbol{\delta U}^k = 0. \tag{4.34}$$

It can be proved that equation (4.33) together with (4.34) determines $\boldsymbol{B}^k$ as

$$\boldsymbol{B}^k = \boldsymbol{B}^{k-1} + \frac{\boldsymbol{R}^k\ (\boldsymbol{\delta U}^k)^T}{(\boldsymbol{\delta U}^k)^T\ \boldsymbol{\delta U}^k} = \boldsymbol{B}^{k-1} + \frac{\boldsymbol{R}^k\ (\boldsymbol{\delta U}^k)^T}{\|\boldsymbol{\delta U}^k\|^2}. \tag{4.35}$$

Recalling the definition of vectors $\boldsymbol{R}^k$ and $\boldsymbol{\delta U}^k$, we can write matrix $\boldsymbol{R}^k\ (\boldsymbol{\delta U}^k)^T$ as

$$\begin{aligned}\boldsymbol{R}^k\ (\boldsymbol{\delta U}^k)^T &= \begin{pmatrix} \boldsymbol{f}_i(\boldsymbol{u}^k) - {}^{n+1}\boldsymbol{f} + \boldsymbol{A}_c^T(\boldsymbol{\lambda}^k + \boldsymbol{\mu}^k) \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\delta u}^k & \boldsymbol{\delta\lambda}^k & \boldsymbol{\delta\mu}^k \end{pmatrix} = \\ &= \begin{pmatrix} \boldsymbol{R}_1^k\ (\boldsymbol{\delta u}^k)^T & \boldsymbol{R}_1^k\ (\boldsymbol{\delta\lambda}^k)^T & \boldsymbol{R}_1^k\ (\boldsymbol{\delta\mu}^k)^T \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix}, \end{aligned} \tag{4.36}$$

where $\boldsymbol{R}_1^k$ denotes the first component of $\boldsymbol{R}^k$,

$$\boldsymbol{R}_1^k = \boldsymbol{f}_i(\boldsymbol{u}^k) - {}^{n+1}\boldsymbol{f} + \boldsymbol{A}_c^T(\boldsymbol{\lambda}^k + \boldsymbol{\mu}^k). \tag{4.37}$$

Substituting equation (4.36) into (4.35) yields

$$\boldsymbol{B}^k = \boldsymbol{B}^{k-1} + \begin{pmatrix} \frac{\boldsymbol{R}_1^k\ (\delta \boldsymbol{u}^k)^T}{\|\delta \boldsymbol{U}^k\|^2} & \frac{\boldsymbol{R}_1^k\ (\delta \boldsymbol{\lambda}^k)^T}{\|\delta \boldsymbol{U}^k\|^2} & \frac{\boldsymbol{R}_1^k\ (\delta \boldsymbol{\mu}^k)^T}{\|\delta \boldsymbol{U}^k\|^2} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix}. \tag{4.38}$$

To study the effect of the update (4.38), let us assume for a moment that $\boldsymbol{B}^{k-1}$ is a real Jacobian matrix ($\boldsymbol{B}^{k-1} = \boldsymbol{J}^{k-1}$). Then equation (4.38) yields

$$\boldsymbol{B}^k = \begin{pmatrix} \boldsymbol{K}^{k-1} + \frac{\boldsymbol{R}_1^k\ (\delta \boldsymbol{u}^k)^T}{\|\delta \boldsymbol{U}^k\|^2} & \boldsymbol{A}_e^T + \frac{\boldsymbol{R}_1^k\ (\delta \boldsymbol{\lambda}^k)^T}{\|\delta \boldsymbol{U}^k\|^2} & \boldsymbol{A}_e^T + \frac{\boldsymbol{R}_1^k\ (\delta \boldsymbol{\mu}^k)^T}{\|\delta \boldsymbol{U}^k\|^2} \\ \boldsymbol{A}_e & \boldsymbol{I} & -\boldsymbol{I} \\ \boldsymbol{A}_e & -\boldsymbol{I} & \boldsymbol{I} \end{pmatrix}. \tag{4.39}$$

Hence, Option T for the Broyden method does not lead to an approximation such as that in equation (4.29), since matrices $\boldsymbol{A}_e^T$ have been modified. From now on we are going to refer to this method as "Broyden-t".

To obtain the inverse version of the Broyden-t update, we need to invert equation (4.38). This can be done by Sherman&Morrison's lemma thanks to the fact that matrix $\boldsymbol{B}^k$ in (4.38) can be rewritten as

$$\boldsymbol{B}^k = \boldsymbol{B}^{k-1} + \boldsymbol{\alpha}^k\ (\boldsymbol{\beta}^k)^T \tag{4.40}$$

where

$$\boldsymbol{\alpha}^k = \frac{1}{\|\delta \boldsymbol{U}^k\|^2} \begin{pmatrix} \boldsymbol{R}_1^k \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix} = \frac{1}{\|\delta \boldsymbol{U}^k\|^2}\ \boldsymbol{R}^k, \tag{4.41}$$

$$\boldsymbol{\beta}^k = \begin{pmatrix} \delta \boldsymbol{u}^k \\ \delta \boldsymbol{\lambda}^k \\ \delta \boldsymbol{\mu}^k \end{pmatrix} = \delta \boldsymbol{U}^k. \tag{4.42}$$

Using the lemma to invert (4.40) and manipulating the resulting expression, we get

$$\boldsymbol{H}^k = \boldsymbol{H}^{k-1} + \frac{\delta \boldsymbol{U}^k - \boldsymbol{H}^{k-1}\ (\boldsymbol{R}^k - \boldsymbol{R}^{k-1})}{(\boldsymbol{\beta}^k)^T\ \boldsymbol{H}^{k-1}\ (\boldsymbol{R}^k - \boldsymbol{R}^{k-1})}\ (\boldsymbol{\beta}^k)^T\ \boldsymbol{H}^{k-1}. \tag{4.43}$$

The expression for $\boldsymbol{H}^k$ in (4.43) can be treated in a similar way as indicated in equations (4.13) to (4.16), where now we will have that

$$\omega^k = \frac{\delta \boldsymbol{U}^k - \boldsymbol{H}^{k-1}\,(\boldsymbol{R}^k - \boldsymbol{R}^{k-1})}{(\boldsymbol{\beta}^k)^T\,\boldsymbol{H}^{k-1}\,(\boldsymbol{R}^k - \boldsymbol{R}^{k-1})},$$

The algorithm that results is given in Table 4.3. For coherence with the rest of the notation, all augmented intermediate vectors are represented by capital letters. We have also introduced a new step where the auxiliary vector $\boldsymbol{\beta}^k$ is defined.

1. Compute $\boldsymbol{J}^0$
2. Solve the linear set $\boldsymbol{J}^0\,\Delta\boldsymbol{U}^1 = -\boldsymbol{R}^0$
3. Update $\boldsymbol{U}^1 = \boldsymbol{U}^0 + \Delta\boldsymbol{U}^1$
4. Evaluate $\boldsymbol{R}^1 = \boldsymbol{R}(\boldsymbol{U}^1)$
5. Convergence control: if point $\boldsymbol{U}^1$ is good enough, exit.

   $k \geq 1$
6. Solve the linear set $\boldsymbol{J}^0\,\boldsymbol{V} = -\boldsymbol{R}^k$
7. $k = 1$ assign $\boldsymbol{T} = \boldsymbol{V}$

   $k > 1$ compute $\boldsymbol{T} = \prod_{i=k-1}^{1}(\boldsymbol{I} + \boldsymbol{W}^i\,(\delta\boldsymbol{U}^i)^T)\,\boldsymbol{V}$
8. Recover from storage $\boldsymbol{\beta}^k = (\delta\boldsymbol{u}^k, \delta\boldsymbol{\lambda}^k, \delta\boldsymbol{\mu}^k) = \delta\boldsymbol{U}^k$
9. Compute and store $\boldsymbol{W}^k = \frac{1}{(\boldsymbol{\beta}^k)^T\,(\delta\boldsymbol{U}^k \cdot \boldsymbol{T})}\,\boldsymbol{T}$
10. Compute and store $\delta\boldsymbol{U}^{k+1} = \boldsymbol{T} + ((\boldsymbol{\beta}^k)^T\,\boldsymbol{T})\;\boldsymbol{W}^k$
11. Update $\boldsymbol{U}^{k+1} = \boldsymbol{U}^k + \delta\boldsymbol{U}^{k+1}$
12. Evaluate $\boldsymbol{R}^{k+1} = \boldsymbol{R}(\boldsymbol{U}^{k+1})$
13. Convergence control: if point $\boldsymbol{U}^{k+1}$ is good enough, exit.
14. Assign $k = k + 1$ and go back to 6.

Table 4.3: Algorithm for the Inverse Broyden-t method.

Option P (Partial approach)

In this case we intend to approximate matrix $\boldsymbol{K}^k$ with a certain matrix $\hat{\boldsymbol{B}}^k$. Therefore, we will need to impose the Broyden condition as

$$\hat{\boldsymbol{B}}^k \, \boldsymbol{z} = \hat{\boldsymbol{B}}^{k-1} \, \boldsymbol{z} \qquad \text{for all } \boldsymbol{z} \text{ such that } \quad \boldsymbol{z}^T \, \boldsymbol{\delta u}^k = 0. \tag{4.44}$$

If the Quasi–Newton equation (4.33) is complemented now with (4.44), the following expression for $\boldsymbol{B}^k$ is obtained:

$$\hat{\boldsymbol{B}}^k = \hat{\boldsymbol{B}}^{k-1} + \frac{\boldsymbol{R}_1^k \, (\boldsymbol{\delta u}^k)^T}{\|\boldsymbol{\delta u}^k\|^2}, \tag{4.45}$$

where $\boldsymbol{R}_1^k$ represents the vector defined in (4.37). We are going to refer to the resulting mehod as the "Broyden-p" method. After $\hat{\boldsymbol{B}}^k$ is obtained, equation (4.29) is employed to define the secant matrix $\boldsymbol{B}^k$ as

$$\boldsymbol{B}^k = \begin{pmatrix} \hat{\boldsymbol{B}}^k & \boldsymbol{A}_e^T & \boldsymbol{A}_e^T \\ \boldsymbol{A}_e & \boldsymbol{I} & -\boldsymbol{I} \\ \boldsymbol{A}_e & -\boldsymbol{I} & \boldsymbol{I} \end{pmatrix}.$$

Equation (4.45) is analogous to (4.35) since one can be obtained from the other by interchanging

$$\boldsymbol{\delta u} \longleftrightarrow \boldsymbol{\delta U}$$

$$\hat{\boldsymbol{B}} \longleftrightarrow \boldsymbol{B}.$$

In this case, the update formula for $\boldsymbol{B}^k$ reads

$$\boldsymbol{B}^k = \boldsymbol{B}^{k-1} + \begin{pmatrix} \frac{\boldsymbol{R}_1^k \, (\boldsymbol{\delta u}^k)^T}{\|\boldsymbol{\delta u}^k\|^2} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}, \tag{4.46}$$

that is,

$$\boldsymbol{B}^k = \begin{pmatrix} \hat{\boldsymbol{B}}^{k-1} + \frac{\boldsymbol{R}_1^k \, (\boldsymbol{\delta u}^k)^T}{\|\boldsymbol{\delta u}^k\|^2} & \boldsymbol{A}_e^T & \boldsymbol{A}_e^T \\ \boldsymbol{A}_e & \boldsymbol{I} & -\boldsymbol{I} \\ \boldsymbol{A}_e & -\boldsymbol{I} & \boldsymbol{I} \end{pmatrix},$$

where, of course, only submatrix $\hat{\boldsymbol{B}}^{k-1}$ has been modified.

Sherman&Morrison's lemma can now be applied to equation (4.46) using that

$$\boldsymbol{B}^k = \boldsymbol{B}^{k-1} + \boldsymbol{\alpha}^k \, (\boldsymbol{\beta}^k)^T, \tag{4.47}$$

where now

$$\boldsymbol{\alpha}^k = \frac{1}{\|\boldsymbol{\delta u}^k\|^2} \begin{pmatrix} \boldsymbol{R}_1^k \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix} = \frac{1}{\|\boldsymbol{\delta u}^k\|^2} \, \boldsymbol{R}^k, \tag{4.48}$$

$$\boldsymbol{\beta}^k = \begin{pmatrix} \boldsymbol{\delta u}^k \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}. \tag{4.49}$$

The inverse update formula that follows is

$$\boldsymbol{H}^k = \boldsymbol{H}^{k-1} + \frac{\boldsymbol{\delta U}^k - \boldsymbol{H}^{k-1} \, (\boldsymbol{R}^k - \boldsymbol{R}^{k-1})}{(\boldsymbol{\beta}^k)^T \, \boldsymbol{H}^{k-1} \, (\boldsymbol{R}^k - \boldsymbol{R}^{k-1})} \, (\boldsymbol{\beta}^k)^T \, \boldsymbol{H}^{k-1}. \tag{4.50}$$

Equation (4.50) is identical to (4.43) except for the definition of $\boldsymbol{\beta}^k$. This means that an algorithm for this inverse Broyden-p method can be produced simply by changing the definition of $\boldsymbol{\beta}^k$ in the Broyden-t algorithm.

Table 4.4 gives the step-by-step description of the Broyden-p method.

Remark. Since the definition of the auxiliar vector $\boldsymbol{\beta}^k$ in the two options of the Broyden method differ, the scalar products in steps 8-10 in Table 4.4 involve "shorter" vectors than in Table 4.3. This may have a relevant significance for very constrained problems.

1. Compute $\boldsymbol{J}^0$
2. Solve the linear set $\boldsymbol{J}^0\ \Delta\boldsymbol{U}^1 = -\boldsymbol{R}^0$
3. Update $\boldsymbol{U}^1 = \boldsymbol{U}^0 + \Delta\boldsymbol{U}^1$
4. Evaluate $\boldsymbol{R}^1 = \boldsymbol{R}(\boldsymbol{U}^1)$
5. Convergence control: if point $\boldsymbol{U}^1$ is good enough, exit.

   $k \geq 1$
6. Solve the linear set $\boldsymbol{J}^0\ \boldsymbol{V} = -\boldsymbol{R}^k$
7. $k = 1$ assign $\boldsymbol{T} = \boldsymbol{V}$

   $k > 1$ compute $\boldsymbol{T} = \prod_{i=k-1}^{1}(\boldsymbol{I} + \boldsymbol{W}^i\ (\delta\boldsymbol{U}^i)^T)\ \boldsymbol{V}$
8. Recover $\delta\boldsymbol{U}^k = (\delta\boldsymbol{u}^k, \delta\boldsymbol{\lambda}^k, \delta\boldsymbol{\mu}^k)$ from storage and build $\boldsymbol{\beta}^k = (\delta\boldsymbol{u}^k, \boldsymbol{0}, \boldsymbol{0})$
9. Compute and store $\boldsymbol{W}^k = \frac{1}{(\boldsymbol{\beta}^k)^T\ (\delta\boldsymbol{U}^k - \boldsymbol{T})}\ \boldsymbol{T}$
10. Compute and store $\delta\boldsymbol{U}^{k+1} = \boldsymbol{T} + ((\boldsymbol{\beta}^k)^T\ \boldsymbol{T})\ \boldsymbol{W}^k$
11. Update $\boldsymbol{U}^{k+1} = \boldsymbol{U}^k + \delta\boldsymbol{U}^{k+1}$
12. Evaluate $\boldsymbol{R}^{k+1} = \boldsymbol{R}(\boldsymbol{U}^{k+1})$
13. Convergence control: if point $\boldsymbol{U}^{k+1}$ is good enough, exit.
14. Assign $k = k + 1$ and go back to 6.

Table 4.4: Algorithm for the Inverse Broyden-p method.

4.3.2 Total/partial version of the BFGS method

Option T (Total approach)

DFP method

As previously seen, for this method the Quasi-Newton equation is complemented with hereditary symmetry and positive-definiteness of the secant approximation $\boldsymbol{B}^k$ to the Jacobian matrix $\boldsymbol{J}^k$:

$$\boldsymbol{B}^{k-1} \text{ symmetrical and positive definite} \Rightarrow \boldsymbol{B}^k \text{ symmetrical and positive definite.} \tag{4.51}$$

These requirements lead to the following expression for $\boldsymbol{B}^k$:

$$\boldsymbol{B}^k = \left[\boldsymbol{I} - \frac{\boldsymbol{c}^{k-1}\,(\boldsymbol{\delta u}^k)^T}{(\boldsymbol{\delta u}^k)^T\,\boldsymbol{c}^{k-1}}\right]\boldsymbol{B}^{k-1}\left[\boldsymbol{I} - \frac{\boldsymbol{\delta u}^k\,(\boldsymbol{c}^{k-1})^T}{(\boldsymbol{\delta u}^k)^T\,\boldsymbol{c}^{k-1}}\right] + \frac{\boldsymbol{c}^{k-1}\,(\boldsymbol{c}^{k-1})^T}{(\boldsymbol{\delta u}^k)^T\,\boldsymbol{c}^{k-1}}, \tag{4.52}$$

where

$$\boldsymbol{c}^{k-1} = \boldsymbol{R}^k - \boldsymbol{R}^{k-1} = \begin{pmatrix} \boldsymbol{R}_1^k - \boldsymbol{R}_1^{k-1} \\ \boldsymbol{0} \\ \boldsymbol{0} \end{pmatrix}. \tag{4.53}$$

Substituting equation (4.53) into (4.52) yields, after some manipulation, the following expression of $\boldsymbol{B}^k$ in terms of $\boldsymbol{B}^{k-1}$:

$$\boldsymbol{B}^k = \begin{pmatrix} \hat{\boldsymbol{B}}^{k-1} + \boldsymbol{M}^{k-1} & \boldsymbol{A}_e^T & \boldsymbol{A}_e^T \\ \boldsymbol{A}_e & \boldsymbol{I} & -\boldsymbol{I} \\ \boldsymbol{A}_e & -\boldsymbol{I} & \boldsymbol{I} \end{pmatrix}, \tag{4.54}$$

where $\boldsymbol{M}^{k-1}$ denotes a certain n-dimensional correction matrix. The important point about equation (4.54) is that *it describes already a Type P update of matrix* $\boldsymbol{B}^k$. That is, only the submatrix $\hat{\boldsymbol{B}}^{k-1}$ has been modified, although this was not an initial request.

Hence, when using a DFP update together with Lagrange multipliers, options T and P yield the same method.

BFGS method

In this method, we use the inverse form of the Quasi-Newton equation

$$\boldsymbol{H}^k\ (\boldsymbol{R}^k - \boldsymbol{R}^{k-1}) = \boldsymbol{\delta U}^k, \tag{4.55}$$

together with the requirement

$$\boldsymbol{H}^{k-1}\ \text{SPD} \Longrightarrow \boldsymbol{H}^k\ \text{SPD}. \tag{4.56}$$

Conditions (4.55) and (4.56) determine

$$\boldsymbol{H}^k = \left[\boldsymbol{I} - \frac{\boldsymbol{\delta u}^k\ (\boldsymbol{c}^{k-1})^T}{(\boldsymbol{c}^{k-1})^T\ \boldsymbol{\delta u}^k}\right] \boldsymbol{H}^{k-1} \left[\boldsymbol{I} - \frac{\boldsymbol{c}^{k-1}\ (\boldsymbol{\delta u}^k)^T}{(\boldsymbol{c}^{k-1})^T\ \boldsymbol{\delta u}^k}\right] + \frac{\boldsymbol{\delta u}^k\ (\boldsymbol{\delta u}^k)^T}{(\boldsymbol{c}^{k-1})^T\ \boldsymbol{\delta u}^k}, \tag{4.57}$$

where $\boldsymbol{c}^{k-1}$ is the same vector as in equation (4.53).

Since the BFGS method is an *inverse* Quasi-Newton method, a direct version is required to study how $\boldsymbol{B}^{k-1}$ is transformed into $\boldsymbol{B}^k$. This results in, [11],

$$\boldsymbol{B}^k = \boldsymbol{B}^{k-1} + \frac{\boldsymbol{c}^{k-1}\ (\boldsymbol{c}^{k-1})^T}{(\boldsymbol{\delta u}^k)^T\ \boldsymbol{c}^{k-1}} + \frac{\boldsymbol{R}^{k-1}(\boldsymbol{R}^{k-1})^T}{(\boldsymbol{\delta u}^k)^T\ \boldsymbol{R}^{k-1}}, \tag{4.58}$$

which can be rearranged into

$$\boldsymbol{B}^k = \begin{pmatrix} \hat{\boldsymbol{B}}^{k-1} + \boldsymbol{N}^{k-1} & \boldsymbol{A}_e^T & \boldsymbol{A}_e^T \\ \boldsymbol{A}_e & \boldsymbol{I} & -\boldsymbol{I} \\ \boldsymbol{A}_e & -\boldsymbol{I} & \boldsymbol{I} \end{pmatrix}, \tag{4.59}$$

with $\boldsymbol{N}^{k-1}$ standing for some n-dimensional correction matrix. Again, the important point about equation (4.59) is that only submatrix $\hat{\boldsymbol{B}}^{k-1}$ is modified, although this was not required a priori.

Therefore, option T leads automatically to an update of type P. This means that the associated algorithm can be obtained simply by substituting vectors $\boldsymbol{u}$ and $\boldsymbol{r}$ in Table 4.2 for their enlarged versions, $\boldsymbol{U}$, $\boldsymbol{R}$. This algorithm is listed in Table 4.5. Again, all augmented auxiliary vectors are represented by capital letters.

1. Compute $\boldsymbol{J}^0$
2. Solve the linear set $\boldsymbol{J}^0\ \Delta\boldsymbol{U}^1 = -\boldsymbol{R}^0$
3. Update $\boldsymbol{U}^1 = \boldsymbol{U}^0 + \Delta\boldsymbol{U}^1$
4. Evaluate $\boldsymbol{R}^1 = \boldsymbol{R}(\boldsymbol{U}^1)$
5. Convergence control: if point $\boldsymbol{U}^1$ is good enough, exit.

   $k \geq 1$

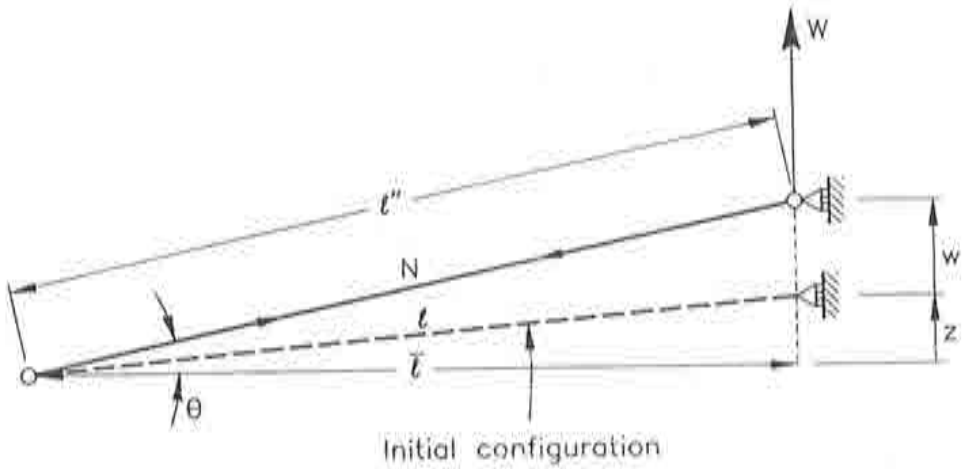
6. Compute and store $\boldsymbol{W}^k = \dfrac{\delta\boldsymbol{U}^k}{(\delta\boldsymbol{U}^k)^T\ (\boldsymbol{R}^{k-1} - \boldsymbol{R}^k)}$
7. Compute and store $\boldsymbol{V}^k = \boldsymbol{R}^k - \left[1 + \sqrt{\dfrac{(\delta\boldsymbol{U}^k)^T\ (\boldsymbol{R}^{k-1} - \boldsymbol{R}^k)}{(\delta\boldsymbol{U}^k)^T \boldsymbol{R}^{k-1}}}\ \right] \boldsymbol{R}^{k-1}$
8. Compute $\boldsymbol{P} = \prod_{i=1}^{k} (\boldsymbol{I} + \boldsymbol{V}^i\ (\boldsymbol{W}^i)^T)\ \boldsymbol{R}^k$
9. Solve the linear set $\boldsymbol{J}^0\ \boldsymbol{Q} = -\boldsymbol{P}$
10. Compute and store $\delta\boldsymbol{U}^{k+1} = \prod_{i=k}^{1} (\boldsymbol{I} + \boldsymbol{W}^i\ (\boldsymbol{V}^i)^T)\ \boldsymbol{Q}$
11. Update $\boldsymbol{U}^{k+1} = \boldsymbol{U}^k + \delta\boldsymbol{U}^{k+1}$
12. Evaluate $\boldsymbol{R}^{k+1} = \boldsymbol{R}(\boldsymbol{U}^{k+1})$
13. Convergence control: if point $\boldsymbol{U}^{k+1}$ is good enough, exit.
14. Assign $k = k + 1$ and go back to 6.

Table 4.5: Algorithm for the BFGS method, options T or P.

4.4 Numerical examples comparing total and partial versions of the Broyden method

Example TRUSS

As a first example, we consider the truss problem presented in Chapter 1. The geometric and material parameters for this test can be found in Figure 4.2.



z=5
ℓ=100
A=10
E=2.1 10^6
ν=0

Figure 4.2: Truss test.

This truss is analysed with a single element. An incremental/iterative analysis is performed with an incremental load $\Delta W = 60$. Figure 4.3 shows the nondimensional load/deflection curve after 8 load steps.

The total and partial versions of the Broyden method are compared in Figures 4.4 and 4.5 for load steps 1 and 8, for which the relative displacement error is plotted vs. the number of iterations. A very strict tolerance parameter $\varepsilon = 10^{-8}$ has been employed to discriminate the behaviour of the methods. However, the

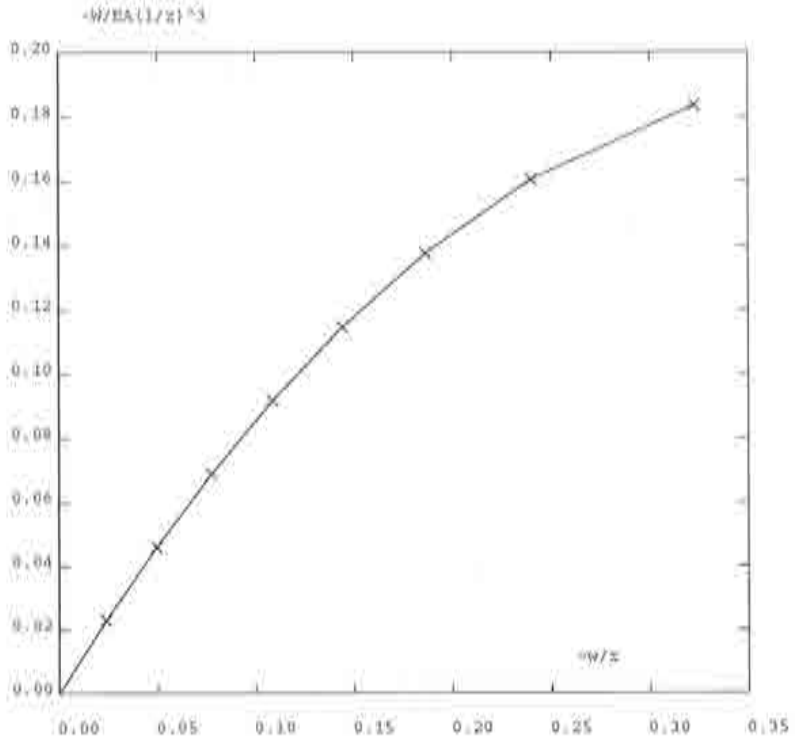


Figure 4.3: Truss test. Load vs. vertical displacement.

simplicity of the test results in a similar behaviour of the two options.

Table 4.6 gives the number of iterations and CPU time (in hundreths of second) for the whole analysis.

Method	Iterations	CPU time
qbroyden_p	44	1 239
qbroyden_t	45	1 246

Table 4.6: Total costs of the QN Broyden methods for the truss test

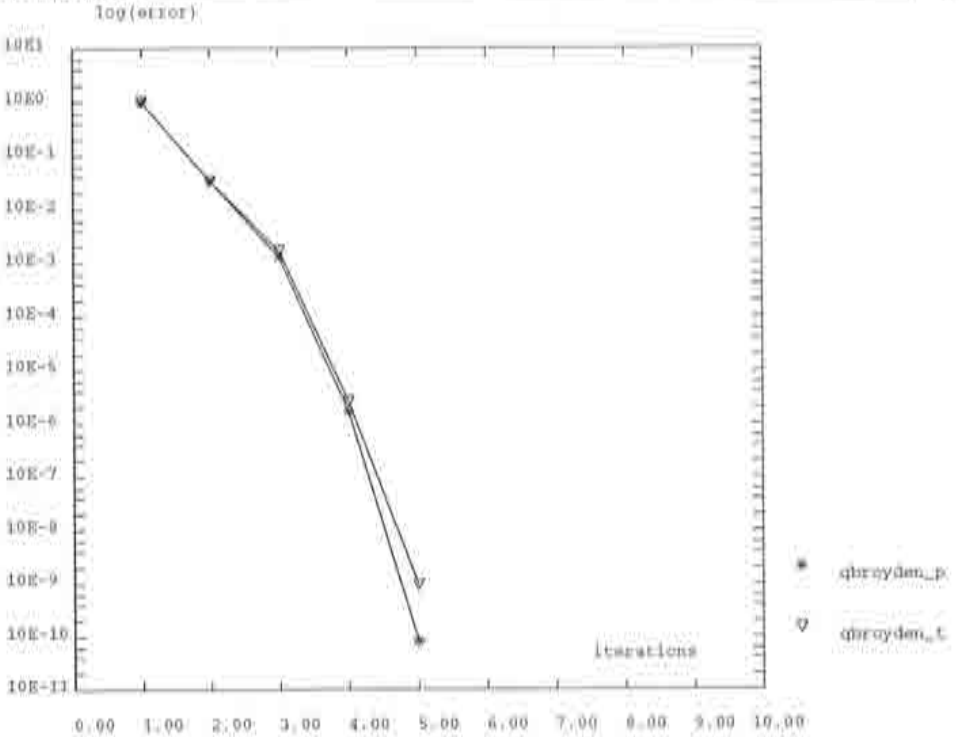


Figure 4.4: Truss test, load step 1. Comparison of the total and partial versions of the Broyden Quasi-Newton method.

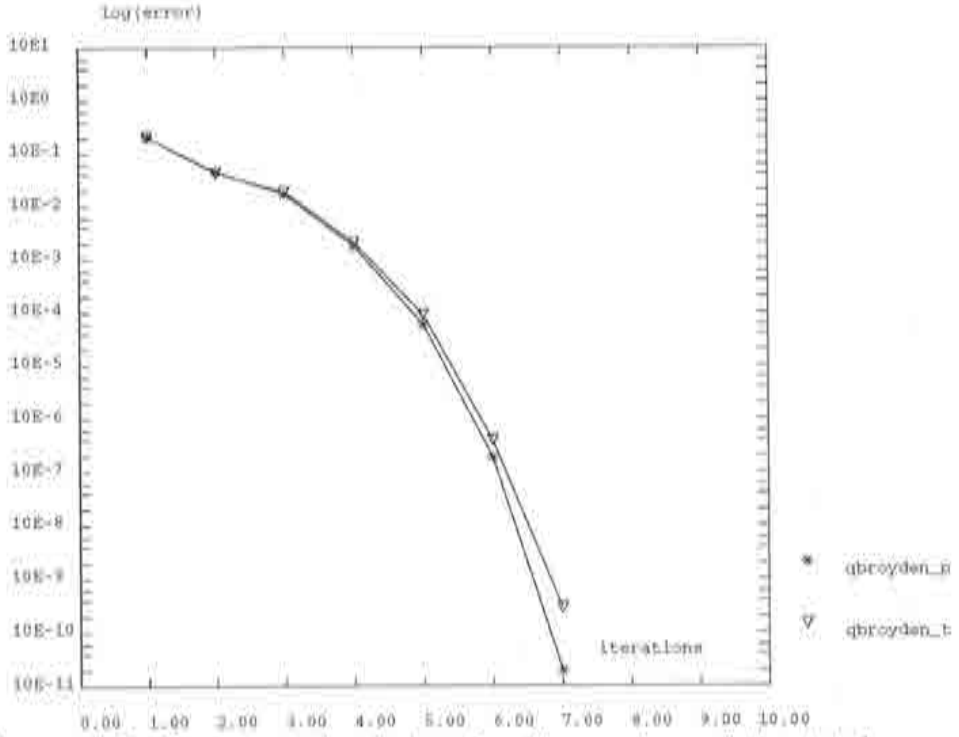


Figure 4.5: Truss test, load step 8. Comparison of the total and partial versions of the Broyden Quasi-Newton method.

Remarks:

- This test is one-dimensional. Thus, the load/deflection curve for this test is a typical curve like those used to illustrate the methods.
- Only a few iterations are required to achieve convergence in each load step, which means that this is a simple test.
- Both the figures and the cost table indicate that there is no substantial difference between the two versions of the QN Broyden method.

If we apply two more increments of load, we obtain the load/deflection curve of Figure 4.6.

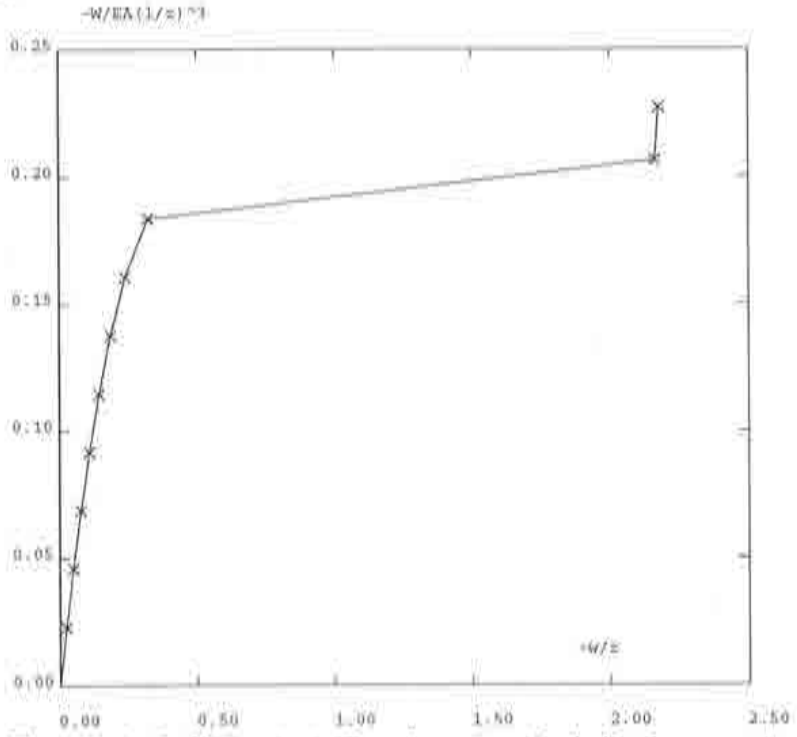


Figure 4.6: Truss test. Load vs. vertical displacement.

Figure 4.6 shows an unreallistically large jump of the deflection in load step 9. This incorrect result is due to a limitation of the strategy of applying successive increments of load. In Chapter 8, we will go back to this test to see that more sophisticated strategies are required to capture the real response of the structure.

Example CYLINDER

This test consists of a small-strain analysis of a perfectly plastic material. A hollow cylinder is subject to an inner pressure p, [7], see Figure 4.7.

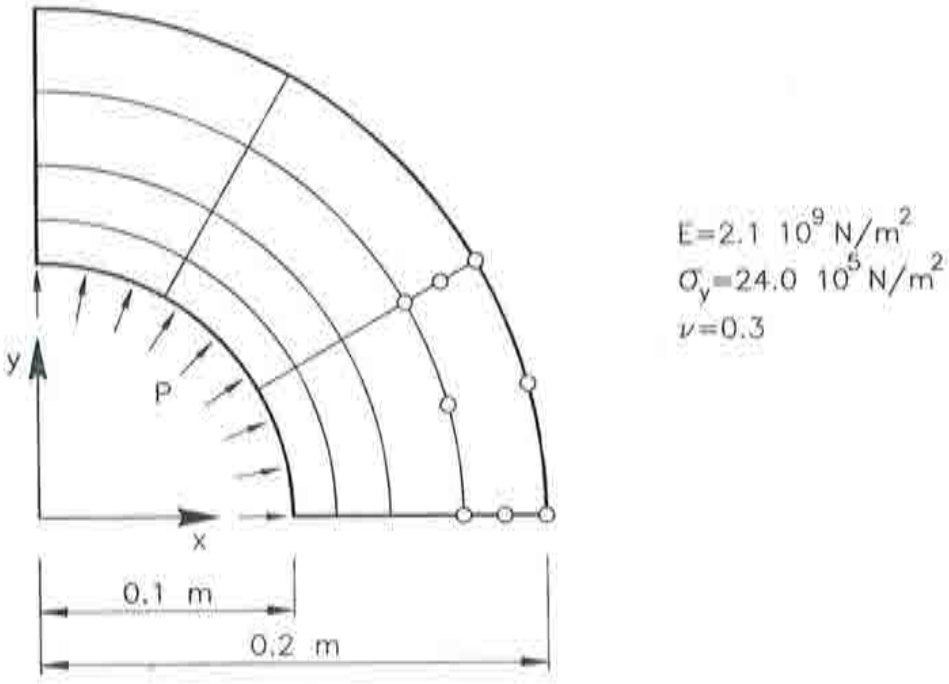


Figure 4.7: Cylinder test.

The cylinder is analysed with bidimensional 8-noded plane strain elements. Figure 4.8 shows the pressure vs. radial displacement response of the solid.

The Broyden-t and Broyden-p methods are compared in Figures 4.9 and 4.10. A tolerance $\varepsilon = 10^{-6}$ has been used. In Figure 4.9, associated to load step 5, the two methods show a very similar behaviour. Figure 4.10 corresponds to a more nonlinear load step 12. This results in a sharp difference between the two methods.

Table 4.7 gives the total amount of iterations and CPU time associated to each option. Only 11 increments are accounted for since the Broyden-t method failed to converge in load step 12.

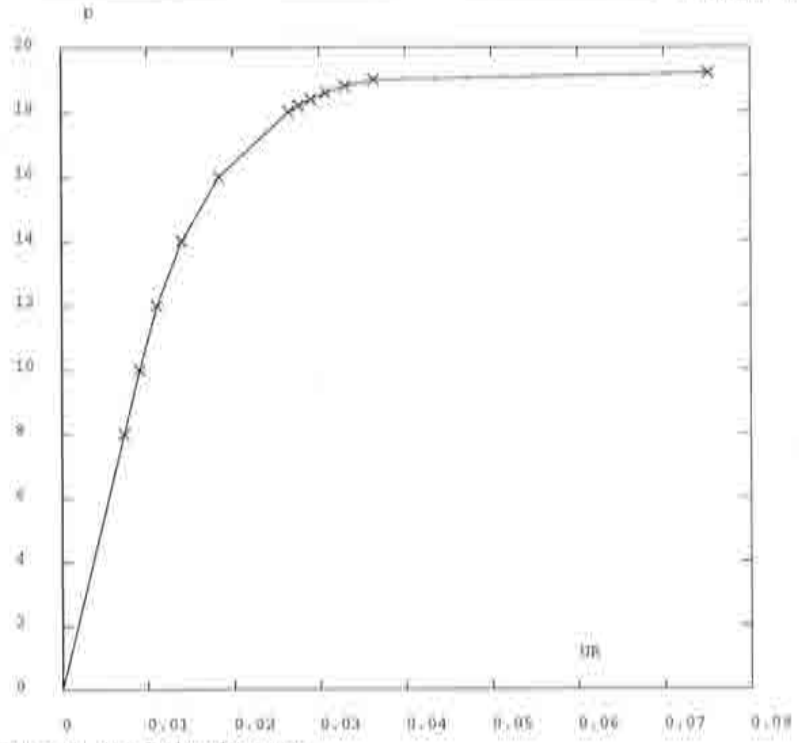


Figure 4.8: Cylinder test.
Internal pressure vs. radial displacement.

Method	Iterations	CPU time
qbroyden_p	59	1 961
qbroyden_t	60	1 958

Table 4.7: Total costs of the QN Broyden methods for the cylinder test

Remarks:

• For the first 11 increments, the two methods, Broyden-t and Broyden-p, show approximately the same behaviour.

• However, for load step 12, the partial version converges in 14 iterations, whereas the total version fails to converge.

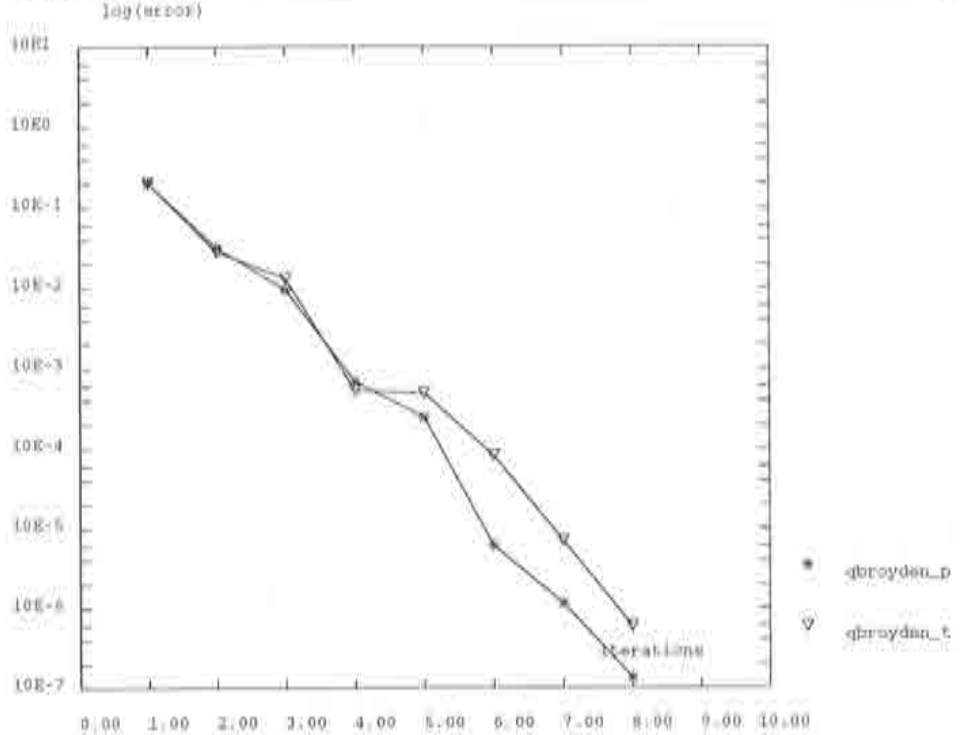


Figure 4.9: Cylinder test, load step 5. Comparison of the total and partial versions of the Broyden Quasi-Newton method.

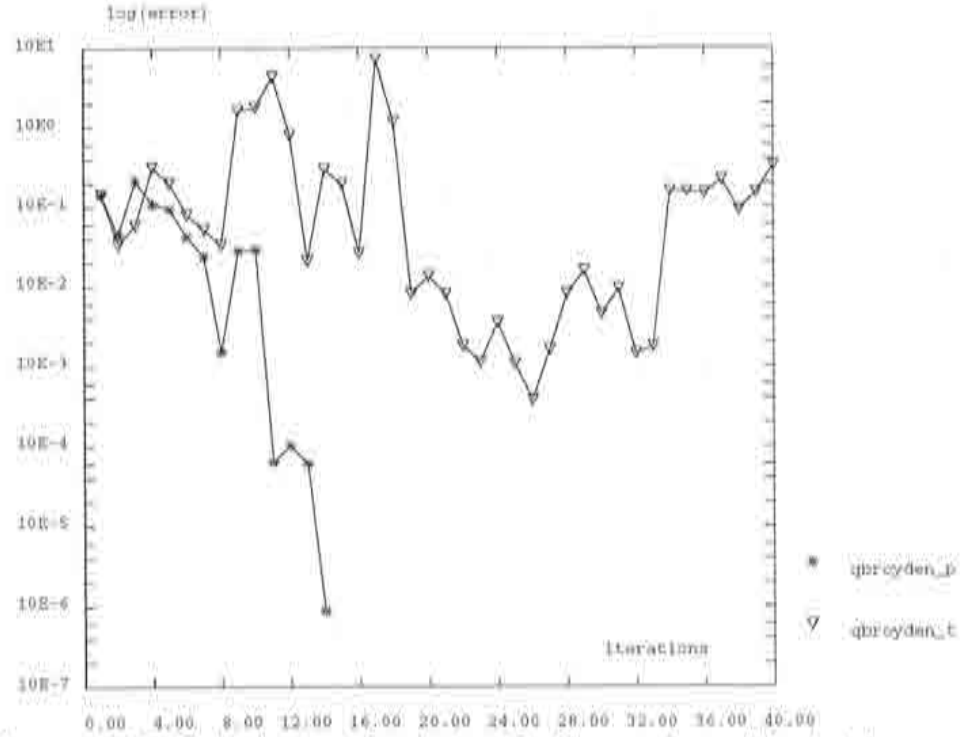


Figure 4.10: Cylinder test, load step 12. Comparison of the total and partial versions of the Broyden Quasi-Newton method.

Example SHELL

In this case, the spherical shell of Figure 4.11 is studied, [7,19,30]. The shell is clamped at the border and loaded at the centre ($r = 0$) with a force P.

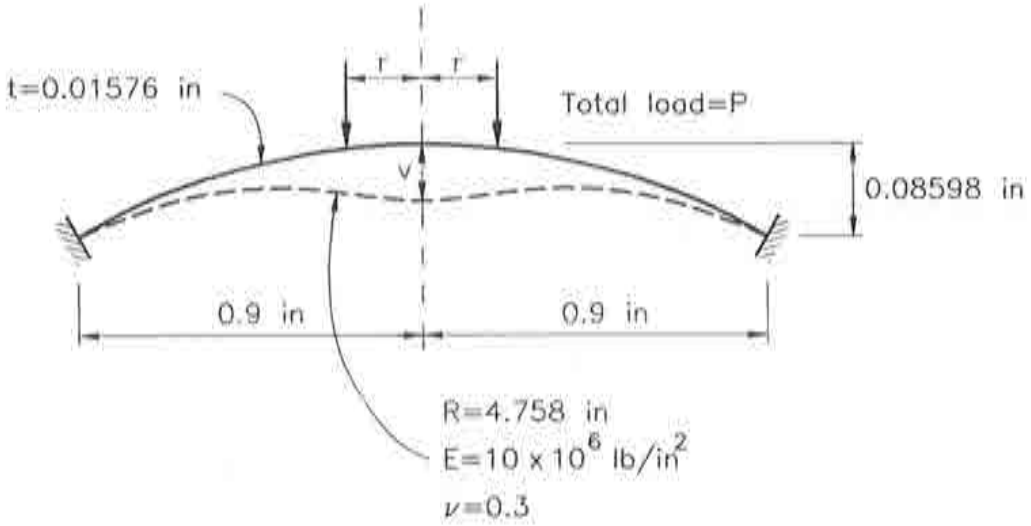


Figure 4.11: Shell test.

An elastic large-strain analysis is performed with 6 axisymmetrical elements. A total load P=80 lb is applied in 80 steps. Figure 4.12 shows the load/deflection curve associated to this test. In Figures 4.13 and 4.14, corresponding to increments 26 and 41, the two versions of the Broyden method are compared. A tolerance $\varepsilon = 10^{-6}$ has been used.

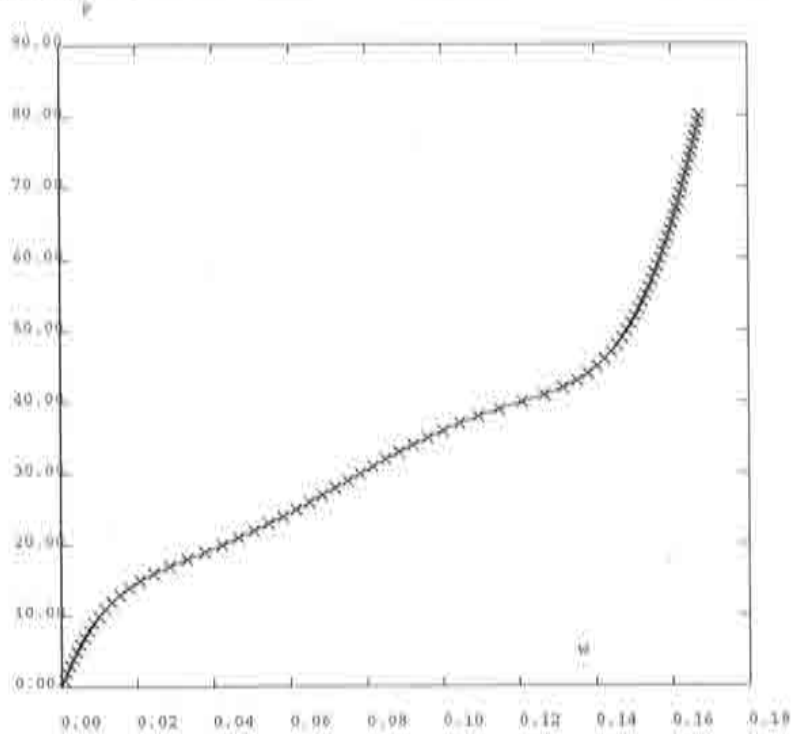


Figure 4.12: Shell test. Load vs. maximum vertical deflection.

Table 4.8 gives the total amount of iterations and CPU time associated to each option.

Method	Iterations	CPU time
qbroyden_p	504	16 241
qbroyden_t	544	17 230

Table 4.8: Total costs of the QN Broyden methods for the shell test

Remarks:

- The shell test is more complex than the truss or cylinder tests, thus requiring a higher amount of iterations.
- The partial version of the QN Broyden method shows a slightly better behaviour than the total version.

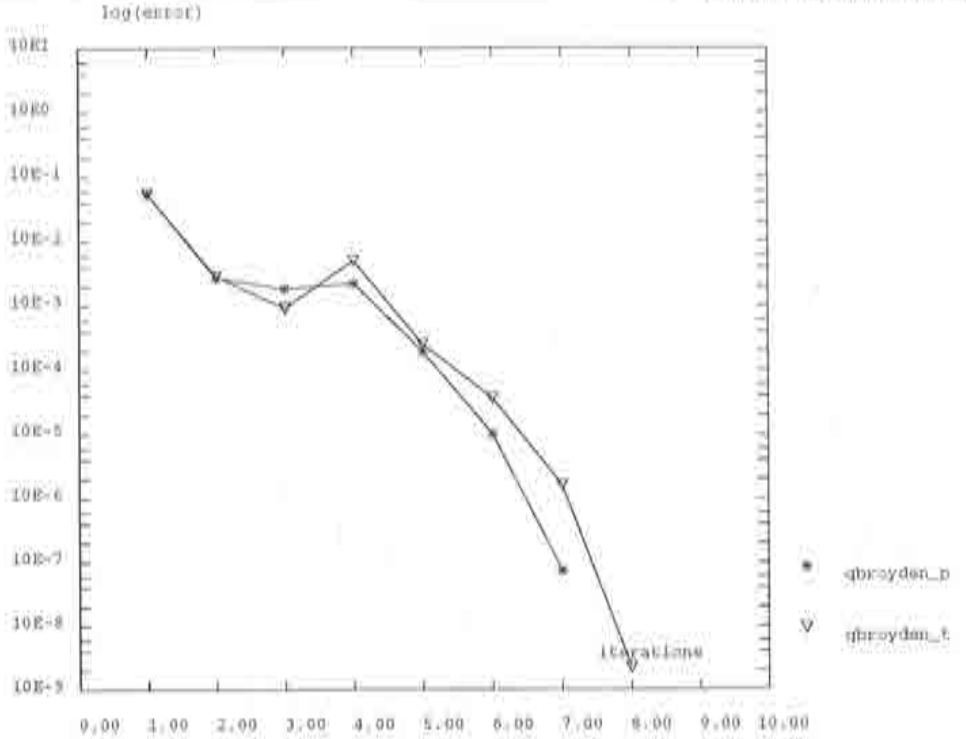


Figure 4.13: Shell test, load step 26. Comparison of the total and partial versions of the Broyden Quasi-Newton method.

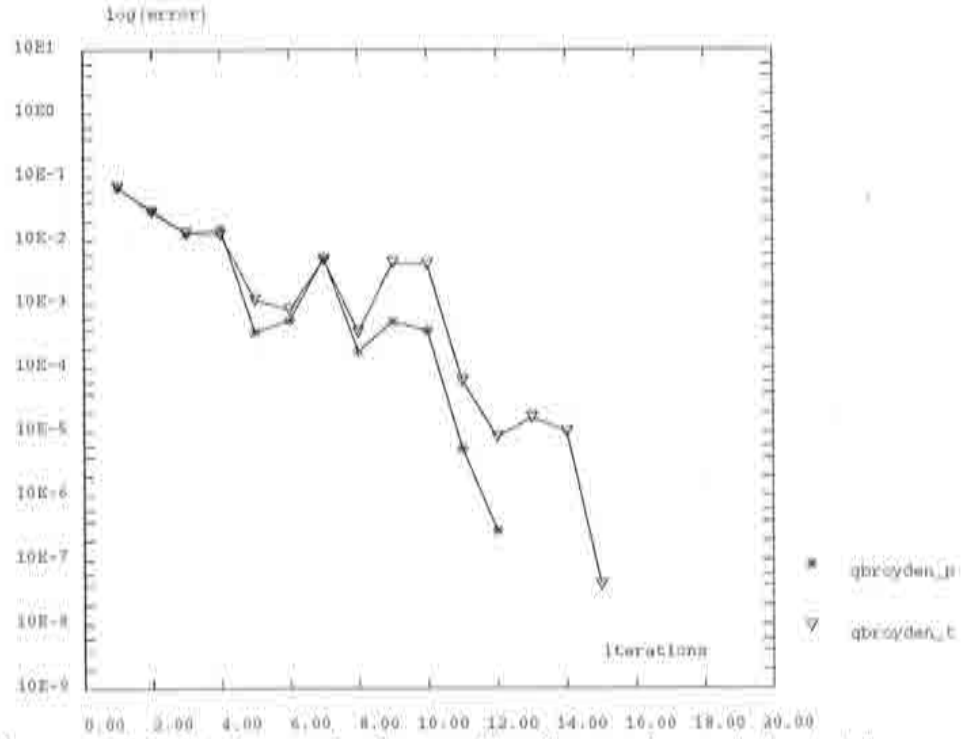


Figure 4.14: Shell test, load step 41. Comparison of the total and partial versions of the Broyden Quasi-Newton method.

Example NECKING

The necking problem is a well-known benchmark test in nonlinear solid mechanics, [26], where both material and geometric nonlinearity are present . A circular bar, with a radius of 6.413 mm and 53.334 mm length, see Figure 4.15, is subject to uniaxial tension. Necking is induced by a slight geometric imperfection (1% radius reduction) in the central part of the bar. The elastoplastic constitutive law can be found in [26]. Figure 4.15 also shows the deformed shape after a 14 mm pull.

A mesh of 50 8-noded axisymmetrical elements is employed to perform an incremental/iterative analysis with 200 load steps, with a tolerance $\varepsilon = 10^{-4}$. The performance of the total and partial versions of the Broyden method are compared in Figures 4.16, 4.17, 4.18 and 4.19, corresponding respectively to two initial steps (2 and 3), an intermediate one (87) and an advanced one (197). Table 4.9 shows the required number of iterations and CPU time.

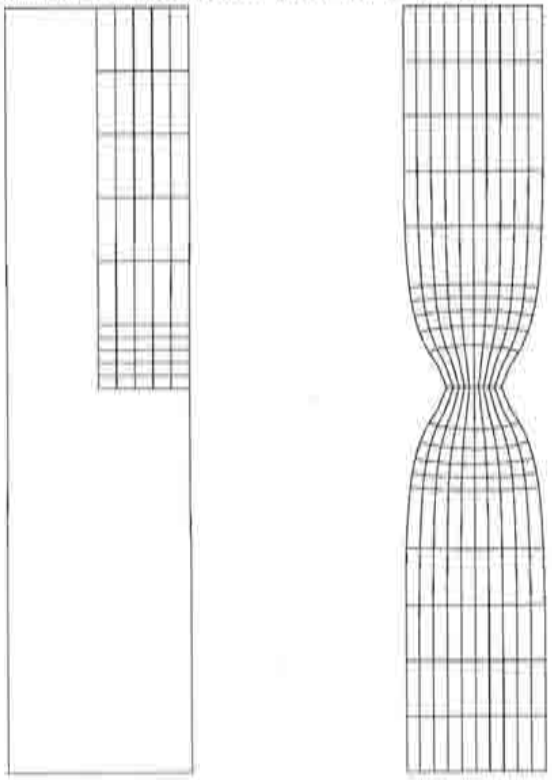


Figure 4.15: Necking test. Initial and deformed meshes.

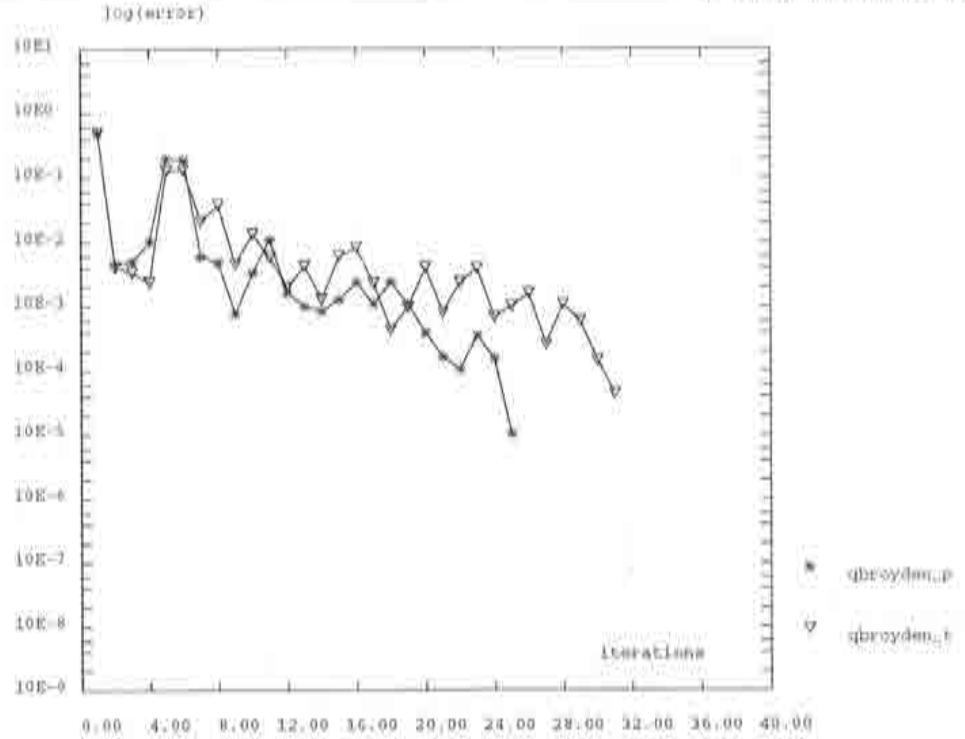


Figure 4.16: Necking test, load step 2. Comparison of the total and partial versions of the Broyden Quasi-Newton method.

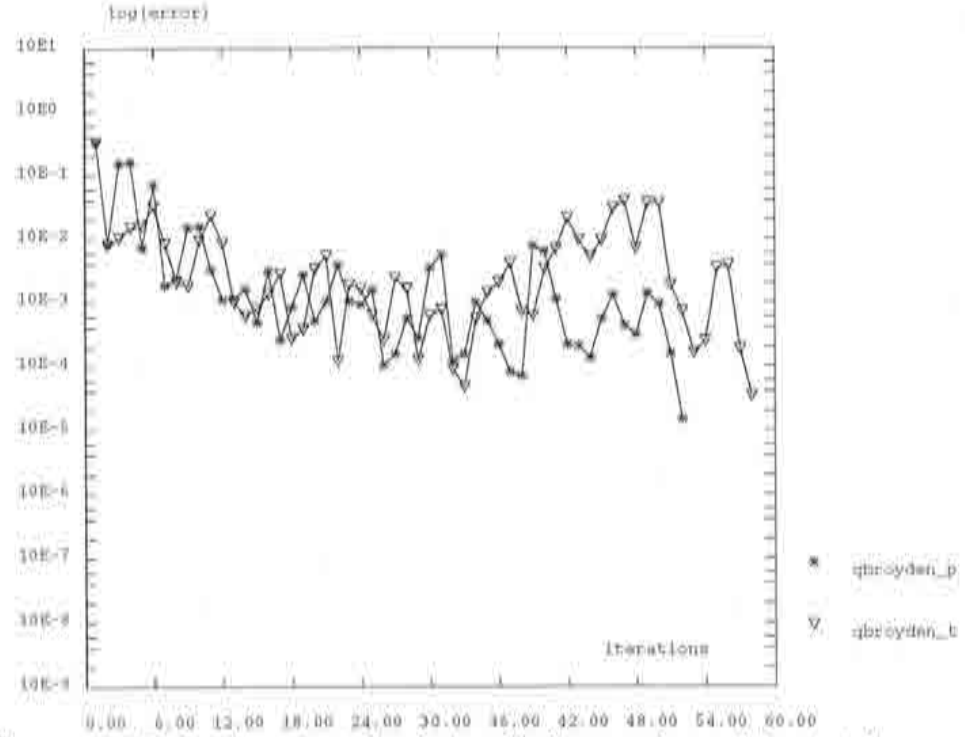


Figure 4.17: Necking test, load step 3. Comparison of the total and partial versions of the Broyden Quasi-Newton method.

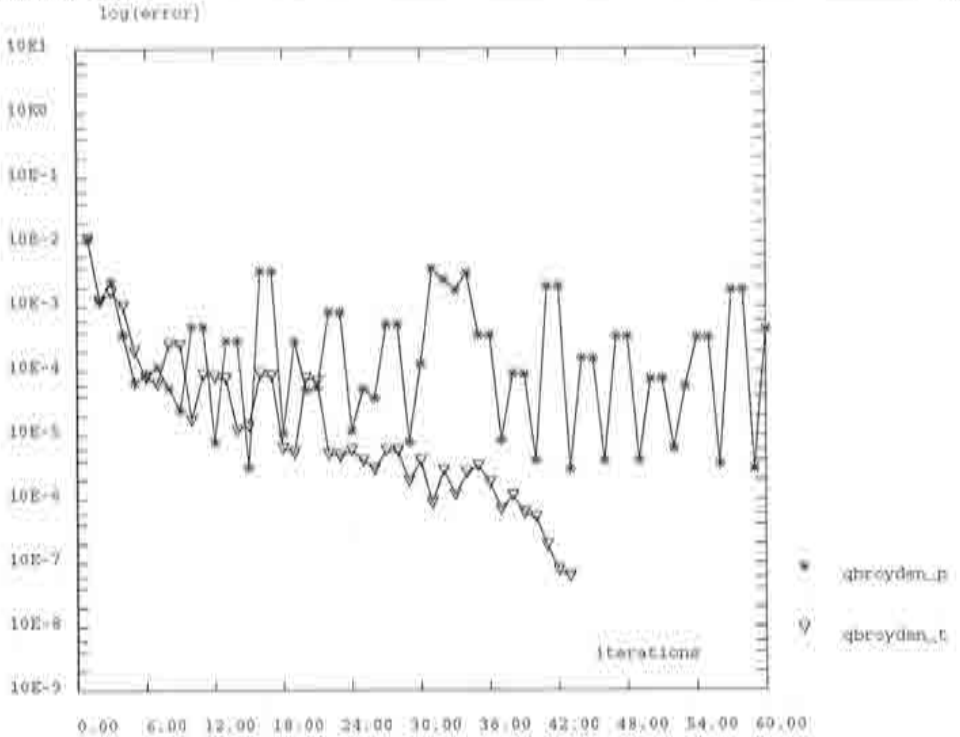


Figure 4.18: Necking test, load step 88. Comparison of the total and partial versions of the Broyden Quasi-Newton method.

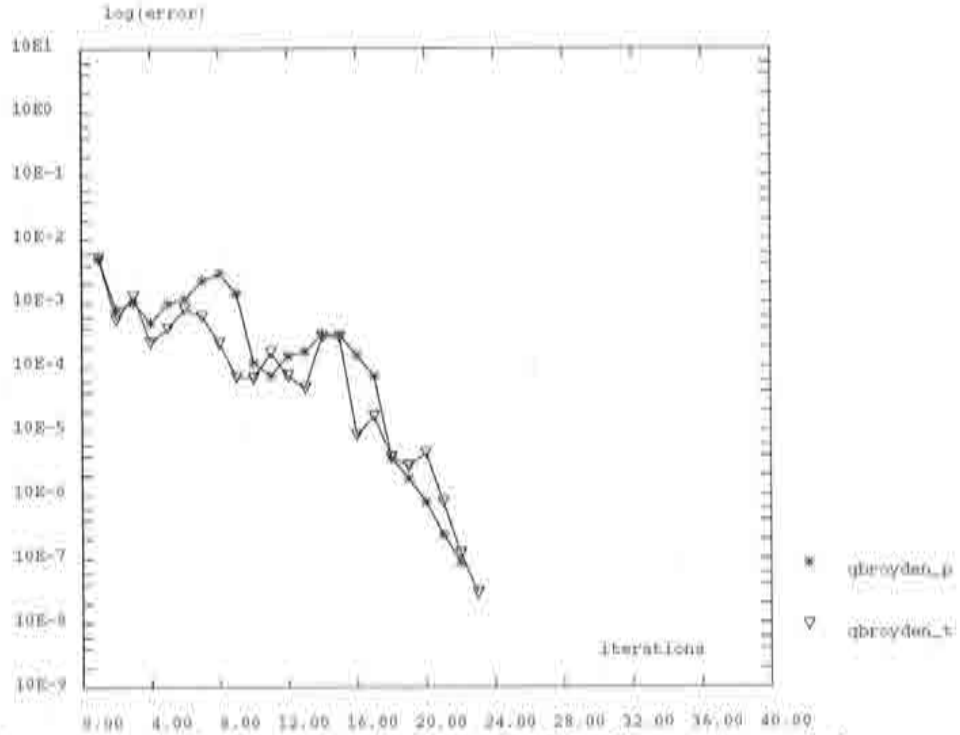


Figure 4.19: Necking test, load step 197. Comparison of the total and partial versions of the Broyden Quasi-Newton method.

Method	Iterations	CPU time
qbroyden_p	1 914	137 826
qbroyden_t	1 809	123 613

Table 4.9: Total costs of the QN Broyden methods for the necking test.

Remarks:

• In this case, the total version of the Broyden method shows a slight advantage over the partial version.

Example THERMIC

In the mechanical tests already presented, tangent stiffness matrices are SPD. The Broyden method is not especially designed to deal with this type of problems, since it does not satisfy hereditary symmetry and positive definiteness.

To assess the behaviour of the Broyden method in a non-symmetrical problem, a simple thermic test is performed, [27], where the diffusion of heat in a nonlinear material is studied. Nonlinearity is associated to a non-constant, temperature-dependent conductivity K of the form

$$K(T) = 1 + 2T^m, \tag{4.60}$$

where T is the temperature. This nonlinear constitutive law results in non-symmetrical Jacobian matrices, [27].

The problem domain, see Figure 4.20, is a sector of a hollow disc with a rectangular hole. Figure 4.20 also shows the boundary conditions: prescribed temperature in the rectangle and the inner arc, a convection condition in the outer arc, and periodicity in the left and right sides. The use of Lagrange multipliers to handle linear restrictions has enabled a simple and straightforward treatment of these periodic boundary conditions.

An iterative analysis with a single step has been carried out setting parameter m in equation (4.60) to 2. The resulting temperature field is plotted in Figure 4.21.

The two options for the Broyden method are compared in Figure 4.22. For this test, the total version shows an oscillatory behaviour during the first iterations. These oscillations are subsequently corrected and then the two methods show approximately parallel convergence curves. Table 4.10 shows the computational cost for the two Broyden methods.

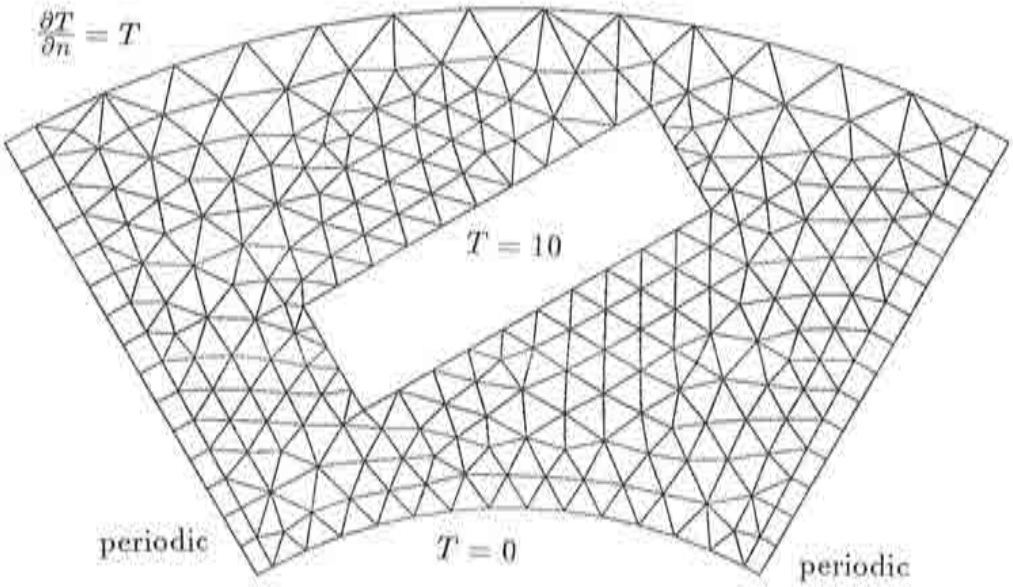


Figure 4.20: Thermic test. Finite element mesh and boundary conditions.

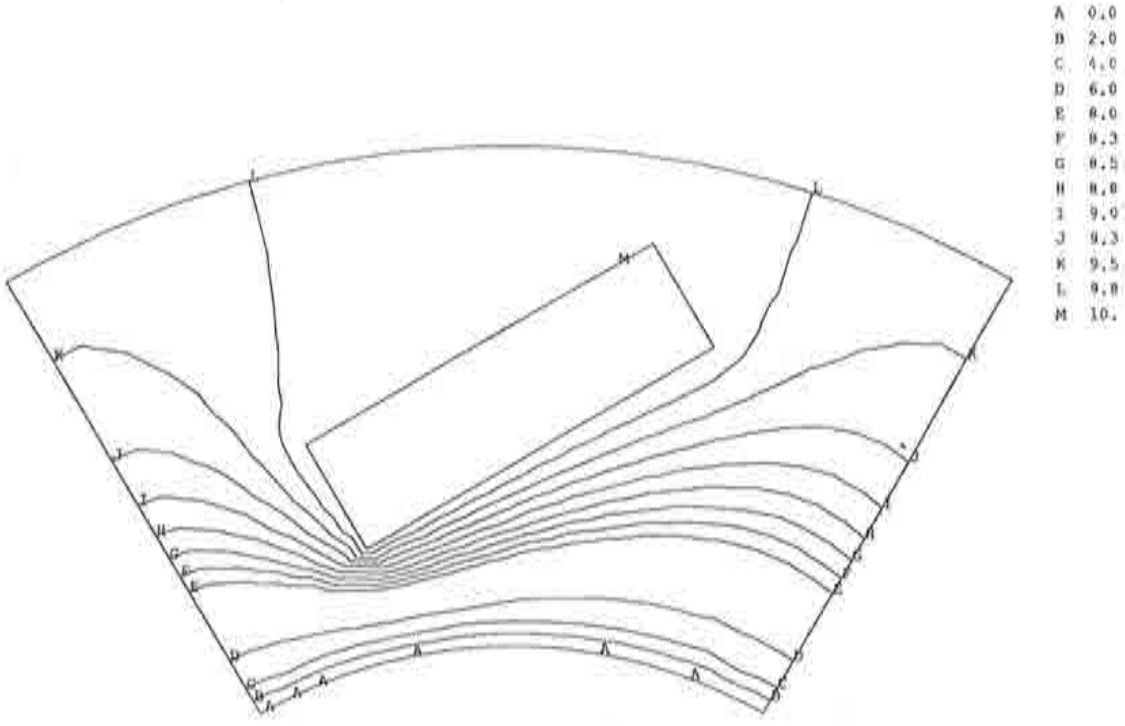


Figure 4.21: Thermic test. Temperature field.

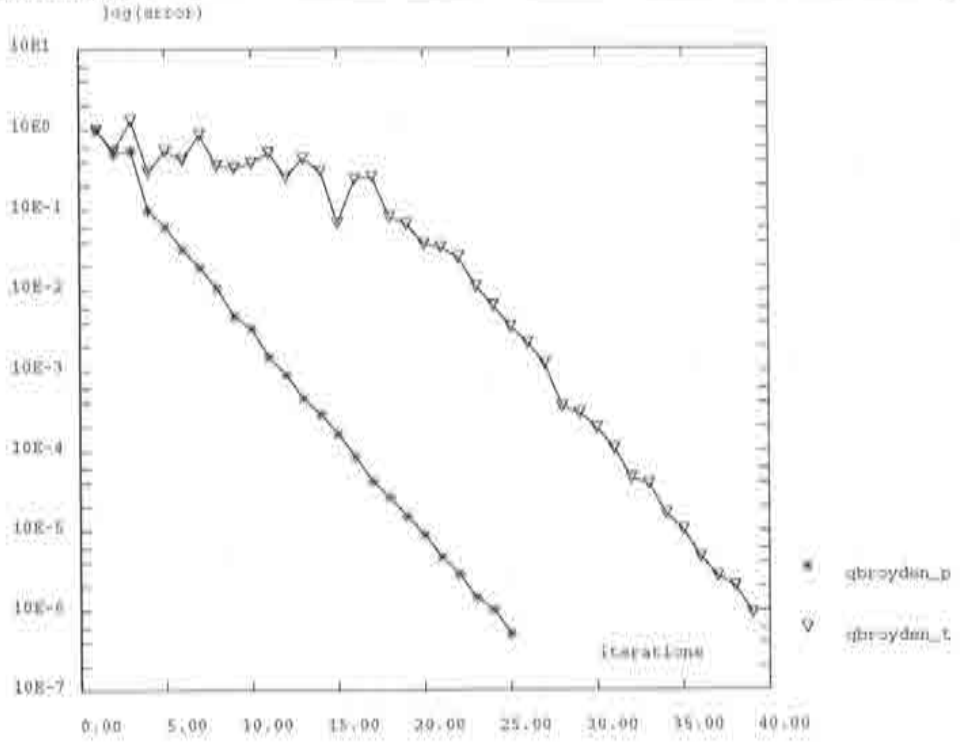


Figure 4.22: Thermic test. Comparison of the total and partial versions of the Broyden Quasi-Newton method.

Method	Iterations	CPU time
qbroyden_p	25	654
qbroyden_t	39	1 373

Table 4.10: Total costs of the QN Broyden methods for the thermic test.

Chapter 5

Secant–Newton methods

5.1 Secant-related acceleration techniques or making Quasi–Newtons run faster

As we saw in Section 4.2, a generic iteration k of a Quasi–Newton method involves the evaluation of one vector of residual forces and the resolution of one linear set of equations (as in mNR), plus the recovery of $2k$ vectors from storage and the computation of $2k$(Broyden)/$4k$(BFGS) scalar products, [4,12,27].

The increasing cost of the iterations is indeed a handicap of the Quasi–Newton methods in the sense that, if convergence is not reached within a small number of iterations, the total cost of the method may turn up to be prohibitive. However, the use of secant matrices has been shown to supply a very good alternative to tangent matrices.

Thus the so-called *Secant-related acceleration techniques* or simply *Secant–Newton methods* (SN), [10], appeared as a modification of the Quasi–Newton scheme with a *constant computational cost per iteration*. The keyword 'acceleration' indicates that SN iterations are computed faster, since the first

iteration of the QN and SN versions cost the same but then the QN iterations have an increasing cost whereas the cost of the SN iterations is kept constant.

In all direct/inverse Quasi–Newton methods, matrix $\boldsymbol{B}^k/\boldsymbol{H}^k$ can be obtained as a function of $\boldsymbol{B}^{k-1}/\boldsymbol{H}^{k-1}$ and other parameters, equations (4.10), (4.13), (4.19), (4.23), (4.26). In the corresponding Secant–Newton version, matrix $\boldsymbol{B}^k/\boldsymbol{H}^k$ is built *as if the previous secant matrix had never been updated*. In other words, $\boldsymbol{B}^{k-1}/\boldsymbol{H}^{k-1}$ is replaced with $\boldsymbol{B}^0/\boldsymbol{H}^0$ in the QN update formula that gives the value for matrix $\boldsymbol{B}^k/\boldsymbol{H}^k$. This is equivalent to a Quasi–Newton without memory where only the last correction is recalled.

In the following section, the Inverse Broyden and BFGS methods in their Secant–Newton form will also be presented.

5.2 Algorithms for Secant Inverse Broyden and BFGSS methods

Secant Inverse Broyden method

Recalling the original update formula for the Quasi–Newton Inverse Broyden method, equation (4.13), the Secant–Newton version of this method is obtained after exchanging $\boldsymbol{H}^{k-1}$ for $\boldsymbol{H}^0$ in the previous equation, which yields

$$\boldsymbol{H}^k = \boldsymbol{H}^0 + \frac{(\boldsymbol{s}^{k-1} - \boldsymbol{H}^0\,\boldsymbol{y}^{k-1})\,(\boldsymbol{s}^{k-1})^T\,\boldsymbol{H}^0}{(\boldsymbol{s}^{k-1})^T\,(\boldsymbol{H}^0\,\boldsymbol{y}^{k-1})}. \tag{5.1}$$

Thus, vector $\boldsymbol{s}^k$ is obtained by postmultiplying (5.1) by $(-\boldsymbol{r}^k)$,

$$\boldsymbol{s}^k = \boldsymbol{H}^k\,(-\boldsymbol{r}^k) = \left[\boldsymbol{H}^0 + \frac{(\boldsymbol{s}^{k-1} - \boldsymbol{H}^0\,\boldsymbol{y}^{k-1})\,(\boldsymbol{s}^{k-1})^T\,\boldsymbol{H}^0}{(\boldsymbol{s}^{k-1})^T\,(\boldsymbol{H}^0\,\boldsymbol{y}^{k-1})}\right](-\boldsymbol{r}^k) =$$

$$= \boldsymbol{H}^0\,(-\boldsymbol{r}^k) + \frac{(\boldsymbol{s}^{k-1} - \boldsymbol{H}^0\,\boldsymbol{y}^{k-1})\,(\boldsymbol{s}^{k-1})^T\,\boldsymbol{H}^0}{(\boldsymbol{s}^{k-1})^T\,(\boldsymbol{H}^0\,\boldsymbol{y}^{k-1})}\,(\boldsymbol{s}^{k-1})^T\,\boldsymbol{H}^0\,(-\boldsymbol{r}^k). \tag{5.2}$$

We can now introduce

$$\tilde{\boldsymbol{s}}^k = \boldsymbol{H}^0\,(-\boldsymbol{r}^k), \tag{5.3}$$

which is in fact the typical mNR correction vector.

Using equation (5.3) and recalling the definition given in (4.3) for $\boldsymbol{y}^{k-1}$ we have that

$$\boldsymbol{H}^0\ \boldsymbol{y}^{k-1} = \boldsymbol{H}^0\ (\boldsymbol{r}^k - \boldsymbol{r}^{k-1}) = \boldsymbol{H}^0\ \boldsymbol{r}^k - \boldsymbol{H}^0\ \boldsymbol{r}^{k-1} = \bar{\boldsymbol{s}}^{k-1} - \bar{\boldsymbol{s}}^k. \tag{5.4}$$

Replacing now equations (5.3) and (5.4) into (5.2), we obtain the following expression for $\boldsymbol{s}^k$, [27]:

$$\boldsymbol{s}^k = \bar{\boldsymbol{s}}^k + \frac{\boldsymbol{s}^{k-1} - (\bar{\boldsymbol{s}}^{k-1} - \bar{\boldsymbol{s}}^k)}{(\boldsymbol{s}^{k-1})^T\ (\bar{\boldsymbol{s}}^{k-1} - \bar{\boldsymbol{s}}^k)}\ (\boldsymbol{s}^{k-1})^T\ \bar{\boldsymbol{s}}^k, \tag{5.5}$$

which can be rearranged to yield

$$\boldsymbol{s}^k = (1+\rho)\ \bar{\boldsymbol{s}}^k + \rho\ \boldsymbol{s}^{k-1} - \rho\ \bar{\boldsymbol{s}}^{k-1}, \tag{5.6}$$

where

$$\rho = \frac{(\boldsymbol{s}^{k-1})^T\ \bar{\boldsymbol{s}}^k}{\tau} \tag{5.7}$$

and

$$\tau = (\boldsymbol{s}^{k-1})^T\ (\bar{\boldsymbol{s}}^{k-1} - \bar{\boldsymbol{s}}^k). \tag{5.8}$$

From equation (5.6) it is clear that the correction vector $\boldsymbol{s}^k$ will be obtained by performing a *linear combination* of vectors $\bar{\boldsymbol{s}}^k$, $\boldsymbol{s}^{k-1}$, $\bar{\boldsymbol{s}}^{k-1}$, which correspond respectively, equation (5.3), to the previous Broyden correction vector, the previous mNR correction vector and the current mNR correction vector. Notice that only two vectors ($\boldsymbol{s}^{k-1}$, $\bar{\boldsymbol{s}}^{k-1}$) will need to be transferred from one iteration to the next and that just two scalar products are performed to obtain τ and ρ.

The complete algorithm for this method is listed in Table 5.1. As in the QN version, if the first secant matrix $\boldsymbol{B}^0$ is chosen as the tangent stiffness matrix $\boldsymbol{K}^0$, the computation of the inverse matrix $\boldsymbol{H}^0$ should be avoided. This is accomplished by solving linear sets with matrix $\boldsymbol{K}^0$ rather than performing direct products with matrix $\boldsymbol{H}^0 = (\boldsymbol{K}^0)^{-1}$.

Apart from this, the original context $[\boldsymbol{u}, \boldsymbol{r}]$ is recovered in Table 5.1. For that purpose, we have introduced the notation

$$\delta\bar{\boldsymbol{u}}^k = \bar{\boldsymbol{s}}^{k-1}. \tag{5.9}$$

1. Compute $\boldsymbol{K}^0$
2. Solve the linear set $\boldsymbol{K}^0\ \Delta\boldsymbol{u}^1 = -\boldsymbol{r}^0$
3. Assign $\delta\tilde{\boldsymbol{u}}^1 = \Delta\boldsymbol{u}^1$
4. Update $\boldsymbol{u}^1 = \boldsymbol{u}^0 + \Delta\boldsymbol{u}^1$
5. Evaluate $\boldsymbol{r}^1 = \boldsymbol{r}(\boldsymbol{u}^1)$
6. Convergence control: if point $\boldsymbol{u}^1$ is good enough, exit.

   $k \geq 1$
7. Solve the linear set $\boldsymbol{K}^0\ \delta\tilde{\boldsymbol{u}}^{k+1} = -\boldsymbol{r}^k$
8. Compute $\tau = (\delta\boldsymbol{u}^k)^T\ (\delta\tilde{\boldsymbol{u}}^k - \delta\tilde{\boldsymbol{u}}^{k+1})$
9. Compute $\rho = \dfrac{(\delta\boldsymbol{u}^k)^T\ \delta\tilde{\boldsymbol{u}}^{k+1}}{\tau}$
10. Compute $\delta\boldsymbol{u}^{k+1} = (1+\rho)\ \delta\tilde{\boldsymbol{u}}^{k+1} + \rho\ \delta\boldsymbol{u}^k - \rho\ \delta\tilde{\boldsymbol{u}}^k$
11. Update $\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \delta\boldsymbol{u}^{k+1}$
12. Evaluate $\boldsymbol{r}^{k+1} = \boldsymbol{r}(\boldsymbol{u}^{k+1})$
13. Convergence control: if point $\boldsymbol{u}^{k+1}$ is good enough, exit.
14. Assign $k = k+1$ and go back to 7.

Table 5.1: Algorithm for the Secant Inverse Broyden method.

Remark 1. For each iteration k, we need to evaluate one vector of out-of-balance forces and solve one linear set of equations (the same as in mNR), recover two vectors from storage and compute two scalar products. Therefore, the cost of a Secant Inverse Broyden iteration is constant, as it was intended from the beginning.

Remark 2. No rate-of-convergence properties are known for the Secant-Newton methods.

Remark 3. If k_{QN} iterations are needed for the QN version of the Broyden method, the total cost per increment (measured in terms of scalar products) is, approximately,

$$c_{QN} = \sum_{i=1}^{k_{QN}} i = 2\,\frac{k_{QN}+1}{2},$$

whereas, if k_{SN} iterations are required for the SN version, the total cost per increment is $c_{SN} = 2\,k_{SN}$. Thus, the SN version is more efficient if, approximately,

$$k_{SN} \leq \frac{1}{2}\,(k_{QN}+1)\,k_{QN}.$$

Remark 4. If ρ is equal to zero in equation (5.6), the following value for the correction vector is obtained

$$\boldsymbol{s}^k = \tilde{\boldsymbol{s}}^k = \boldsymbol{H}^0\,(-\boldsymbol{r}^k) = (\boldsymbol{K}^0)^{-1}\,(-\boldsymbol{r}^k),$$

which corresponds to the modified Newton–Raphson method. In this sense, the SN Inverse Broyden method can also be regarded as a *refinement of a mNR* in which a weighted linear combination of the mNR correction vector $\tilde{\boldsymbol{s}}^k$ together with vectors $\boldsymbol{s}^{k-1}$ and $\tilde{\boldsymbol{s}}^{k-1}$ is performed.

Secant BFGS method

The Secant BFGS (or BFGSS) method is obtained by replacing matrix $\boldsymbol{H}^{k-1}$ with $\boldsymbol{H}^0$ in the QN BFGS update formula, [10],

$$\boldsymbol{H}^k_{BFGSS} = \left[\boldsymbol{I} - \frac{\boldsymbol{s}^{k-1}\,(\boldsymbol{y}^{k-1})^T}{(\boldsymbol{y}^{k-1})^T\,\boldsymbol{s}^{k-1}}\right]\boldsymbol{H}^0\left[\boldsymbol{I} - \frac{\boldsymbol{y}^{k-1}\,(\boldsymbol{s}^{k-1})^T}{(\boldsymbol{y}^{k-1})^T\,\boldsymbol{s}^{k-1}}\right] + \frac{\boldsymbol{s}^{k-1}\,(\boldsymbol{s}^{k-1})^T}{(\boldsymbol{y}^{k-1})^T\,\boldsymbol{s}^{k-1}}. \quad (5.10)$$

Using equation (5.10), the correction vector $\boldsymbol{s}^k$ can be computed as

$$\boldsymbol{s}^k = \boldsymbol{H}^k\,(-\boldsymbol{r}^k) = \left[\boldsymbol{I} - \frac{\boldsymbol{s}^{k-1}\,(\boldsymbol{y}^{k-1})^T}{(\boldsymbol{y}^{k-1})^T\,\boldsymbol{s}^{k-1}}\right]\boldsymbol{H}^0\left[\boldsymbol{I} - \frac{\boldsymbol{y}^{k-1}\,(\boldsymbol{s}^{k-1})^T}{(\boldsymbol{y}^{k-1})^T\,\boldsymbol{s}^{k-1}}\right](-\boldsymbol{r}^k) +$$

$$+ \frac{\boldsymbol{s}^{k-1}\,(\boldsymbol{s}^{k-1})^T}{(\boldsymbol{y}^{k-1})^T\,\boldsymbol{s}^{k-1}}\,(-\boldsymbol{r}^k). \quad (5.11)$$

Equation (5.11) can be manipulated in a similar way as in the Secant Broyden method. The final result of this manipulation is, using the previous definition of $\tilde{\boldsymbol{s}}^k$, equation (5.3),

$$\boldsymbol{s}^k = (1+C)\,\tilde{\boldsymbol{s}}^k + (-C)\,\tilde{\boldsymbol{s}}^{k-1} + \Big(C - (1+C)B + CA\Big)\,\boldsymbol{s}^{k-1}, \tag{5.12}$$

with A, B, C being the three following scalar values:

$$A = \frac{(\boldsymbol{y}^{k-1})^T\,\tilde{\boldsymbol{s}}^{k-1}}{(\boldsymbol{s}^{k-1})^T\,\boldsymbol{y}^{k-1}} \tag{5.13}$$

$$B = \frac{(\boldsymbol{y}^{k-1})^T\,\tilde{\boldsymbol{s}}^{k}}{(\boldsymbol{s}^{k-1})^T\,\boldsymbol{y}^{k-1}} \tag{5.14}$$

$$C = \frac{(\boldsymbol{y}^{k-1})^T\,\tilde{\boldsymbol{r}}^{k}}{(\boldsymbol{s}^{k-1})^T\,\boldsymbol{y}^{k-1}}. \tag{5.15}$$

From equations (5.11) to (5.15) it can be seen that a BFGSS update involves the storage of two vectors and the computation of four scalar products.

Table 5.2 contains the step-by-step algorithm of this method.

Remark 1. For each iteration k, we need to evaluate one vector of out-of-balance forces (the same as in mNR), recover two vectors form storage and compute four scalar products (τ_1, τ_2, τ_3, τ_4).

Remark 2. The Secant BFGS method can also be interpreted as a refinement of the modified Newton–Raphson method. The mNR correction can be recovered by setting the values of A, B, C in (5.12) to zero.

Remark 3. As commented previously, no rate-of-convergence properties are known for the Secant-Newton methods.

Equation (5.12) can be further simplified into the linear combination of only two or one correction vectors. The resulting methods, developed by Crisfield, [10], are known respectively as SN2, SN1.

1. Compute $\boldsymbol{K}^0$
2. Solve the linear set $\boldsymbol{K}^0\ \boldsymbol{\Delta u}^1 = -\boldsymbol{r}^0$
3. Assign $\boldsymbol{\delta \tilde{u}}^1 = \boldsymbol{\Delta u}^1$
4. Update $\boldsymbol{u}^1 = \boldsymbol{u}^0 + \boldsymbol{\Delta u}^1$
5. Evaluate $\boldsymbol{r}^1 = \boldsymbol{r}(\boldsymbol{u}^1)$
6. Convergence control: if point $\boldsymbol{u}^1$ is good enough, exit.

   $k \geq 1$
7. Solve the linear set $\boldsymbol{K}^0\ \boldsymbol{\delta \tilde{u}}^{k+1} = -\boldsymbol{r}^k$
8. Compute

$$\tau_1 = (\boldsymbol{r}^{k-1} - \boldsymbol{r}^k)^T\ \boldsymbol{\delta \bar{u}}^k$$

$$\tau_2 = (\boldsymbol{r}^{k-1} - \boldsymbol{r}^k)^T\ \boldsymbol{\delta \tilde{u}}^{k+1}$$

$$\tau_3 = (\boldsymbol{\delta u}^k)^T\ \boldsymbol{r}^k$$

$$\tau_4 = (\boldsymbol{\delta u}^k)^T\ (\boldsymbol{r}^{k-1} - \boldsymbol{r}^k)$$

9. Compute

$$A = \frac{\tau_1}{\tau_4}$$

$$B = \frac{\tau_2}{\tau_4}$$

$$C = \frac{\tau_3}{\tau_4}$$

10. Compute

$$\boldsymbol{\delta u}^{k+1} = (1 + C)\ \boldsymbol{\delta \tilde{u}}^{k+1} + (-C)\ \boldsymbol{\delta \tilde{u}}^k + (C - (1 + C)B + CA)\ \boldsymbol{\delta u}^k$$

11. Update $\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \boldsymbol{\delta u}^{k+1}$
12. Evaluate $\boldsymbol{r}^{k+1} = \boldsymbol{r}(\boldsymbol{u}^{k+1})$
13. Convergence control: if point $\boldsymbol{u}^{k+1}$ is good enough, exit.
14. Assign $k = k + 1$ and go back to 7.

Table 5.2: Algorithm for the BFGSS method.

5.3 Again two choices when working with Lagrange multipliers

As presented in Section 4.3, two different choices for the secant matrix $\boldsymbol{B}^k$ can be made when treating boundary conditions via Lagrange multipliers. These two choices (Option T –Total– and Option P –Partial–) lead to different methods in the case of the Broyden technique, but they describe exactly the same update in the case of the BFGS method.

Similar conclusions will be drawn for the Secant–Newton version of the studied QN method; that is, two different Secant Broyden are devised while only one BFGSS is produced.

5.3.1 Total and partial versions of the Secant Inverse Broyden method

Option T (Total approach)

To obtain the secant version of the Broyden-t method, matrix $\boldsymbol{H}^{k-1}$ in equation (4.43) must be replaced with $\boldsymbol{H}^0$. The resulting expression,

$$\boldsymbol{H}^k = \boldsymbol{H}^0 + \frac{\boldsymbol{\delta U}^k - \boldsymbol{H}^0\,(\boldsymbol{R}^k - \boldsymbol{R}^{k-1})}{(\boldsymbol{\beta}^k)^T\,\boldsymbol{H}^0\,(\boldsymbol{R}^k - \boldsymbol{R}^{k-1})}\,(\boldsymbol{\beta}^k)^T\,\boldsymbol{H}^0, \tag{5.16}$$

can be manipulated as in equations (5.3) to (5.6).

Recalling that $(\boldsymbol{\beta}^k)^T = (\boldsymbol{\delta u}^k \quad \boldsymbol{\delta\lambda}^k \quad \boldsymbol{\delta\mu}^k)$ and defining $\boldsymbol{\delta\tilde{U}}^k = \boldsymbol{H}^0\,(-\boldsymbol{R}^k)$, we obtain the algorithm that is listed in Table 5.3. This algorithm corresponds exactly to the *enlarged* version of the one in Table 5.1. A new step containing the definition of vector $\boldsymbol{\beta}^k$ has also been introduced.

1. Compute $\boldsymbol{J}^0$
2. Solve the linear set $\boldsymbol{J}^0\ \Delta\boldsymbol{U}^1 = -\boldsymbol{R}^0$
3. Assign $\delta\tilde{\boldsymbol{U}}^1 = \Delta\boldsymbol{U}^1$
4. Update $\boldsymbol{U}^1 = \boldsymbol{U}^0 + \Delta\boldsymbol{U}^1$
5. Evaluate $\boldsymbol{R}^1 = \boldsymbol{R}(\boldsymbol{U}^1)$
6. Convergence control: if point $\boldsymbol{U}^1$ is good enough, exit.

   $k \geq 1$
7. Solve the linear set $\boldsymbol{J}^0\ \delta\tilde{\boldsymbol{U}}^{k+1} = -\boldsymbol{R}^k$
8. Recover from storage $\boldsymbol{\beta}^k = (\delta\boldsymbol{u}^k, \delta\boldsymbol{\lambda}^k, \delta\boldsymbol{\mu}^k) = \delta\boldsymbol{U}^k$
9. Compute $\tau = (\boldsymbol{\beta}^k)^T\ (\delta\tilde{\boldsymbol{U}}^k - \delta\tilde{\boldsymbol{U}}^{k+1})$
10. Compute $\rho = \dfrac{(\boldsymbol{\beta}^k)^T\ \delta\tilde{\boldsymbol{U}}^{k+1}}{\tau}$
11. Compute $\delta\boldsymbol{U}^{k+1} = (1+\rho)\ \delta\tilde{\boldsymbol{U}}^{k+1} + \rho\ \delta\boldsymbol{U}^k - \rho\ \delta\tilde{\boldsymbol{U}}^k$
12. Update $\boldsymbol{U}^{k+1} = \boldsymbol{U}^k + \delta\boldsymbol{U}^{k+1}$
13. Evaluate $\boldsymbol{R}^{k+1} = \boldsymbol{R}(\boldsymbol{U}^{k+1})$
14. Convergence control: if point $\boldsymbol{U}^{k+1}$ is good enough, exit.
15. Assign $k = k+1$ and go back to 7.

Table 5.3: Algorithm for the Secant Inverse Broyden-t method.

Option P (Partial approach)

To obtain the general update formula for this method, we can either substitute $\boldsymbol{H}^{k-1}$ for $\boldsymbol{H}^0$ in (4.50) or directly take the algorithm in Table 5.3 and replace the definition of $\boldsymbol{\beta}^k$ with

$$\boldsymbol{\beta}^k = (\delta\boldsymbol{u}^k, \boldsymbol{0}, \boldsymbol{0})$$

The final algorithm that is obtained is the one presented in Table 5.4.

1. Compute $\boldsymbol{J}^0$
2. Solve the linear set $\boldsymbol{J}^0\ \boldsymbol{\Delta U}^1 = -\boldsymbol{R}^0$
3. Assign $\boldsymbol{\delta \tilde{U}}^1 = \boldsymbol{\Delta U}^1$
4. Update $\boldsymbol{U}^1 = \boldsymbol{U}^0 + \boldsymbol{\Delta U}^1$
5. Evaluate $\boldsymbol{R}^1 = \boldsymbol{R}(\boldsymbol{U}^1)$
6. Convergence control: if point $\boldsymbol{U}^1$ is good enough, exit.

   $k \geq 1$
7. Solve the linear set $\boldsymbol{J}^0\ \boldsymbol{\delta \tilde{U}}^{k+1} = -\boldsymbol{R}^k$
8. Recover $\boldsymbol{\delta U}^k = (\boldsymbol{\delta u}^k, \boldsymbol{\delta \lambda}^k, \boldsymbol{\delta \mu}^k)$ from storage and build $\boldsymbol{\beta}^k = (\boldsymbol{\delta u}^k, \mathbf{0}, \mathbf{0})$
9. Compute $\tau = (\boldsymbol{\beta}^k)^T\ (\boldsymbol{\delta \tilde{U}}^k - \boldsymbol{\delta \tilde{U}}^{k+1})$
10. Compute $\rho = \dfrac{(\boldsymbol{\beta}^k)^T\ \boldsymbol{\delta \tilde{U}}^{k+1}}{\tau}$
11. Compute $\boldsymbol{\delta U}^{k+1} = (1+\rho)\ \boldsymbol{\delta \tilde{U}}^{k+1} + \rho\ \boldsymbol{\delta U}^k - \rho\ \boldsymbol{\delta \tilde{U}}^k$
12. Update $\boldsymbol{U}^{k+1} = \boldsymbol{U}^k + \boldsymbol{\delta U}^{k+1}$
13. Evaluate $\boldsymbol{R}^{k+1} = \boldsymbol{R}(\boldsymbol{U}^{k+1})$
14. Convergence control: if point $\boldsymbol{U}^{k+1}$ is good enough, exit.
15. Assign $k = k+1$ and go back to 7.

Table 5.4: Algorithm for the Secant Inverse Broyden-p method.

5.3.2 Total/partial version of the BFGSS method

In Section 4.3.2, the partial and total approach to the BFGS method resulted in the same algorithm. Thus it seems reasonable to expect only one secant version of the BFGS method. As formulations T and P are equivalent, the final algorithm for the BFGSS method will be obtained by *enlarging* all vectors and matrices in the BFGSS algorithm presented in Table 5.2.

The enlarged BFGSS algorithm is listed in Table 5.5.

Remark . Similarly to the case of the Broyden method, if k_{QN} and k_{SN} iterations are required respectively for the QN and SN versions of the BFGS method, the SN version is more efficient if, approximately,

$$k_{\mathrm{SN}} \leq \frac{1}{2}\,(k_{\mathrm{QN}} + 1)\,k_{\mathrm{QN}}.$$

1. Compute $\boldsymbol{J}^0$
2. Solve the linear set $\boldsymbol{J}^0\ \Delta\boldsymbol{U}^1 = -\boldsymbol{R}^0$
3. Assign $\delta\tilde{\boldsymbol{U}}^1 = \Delta\boldsymbol{U}^1$
4. Update $\boldsymbol{U}^1 = \boldsymbol{U}^0 + \Delta\boldsymbol{U}^1$
5. Evaluate $\boldsymbol{R}^1 = \boldsymbol{R}(\boldsymbol{U}^1)$
6. Convergence control: if point $\boldsymbol{U}^1$ is good enough, exit.

   $k \geq 1$
7. Solve the linear set $\boldsymbol{J}^0\ \delta\tilde{\boldsymbol{U}}^{k+1} = -\boldsymbol{R}^k$
8. Compute

$$\tau_1 = (\boldsymbol{R}^{k-1} - \boldsymbol{R}^k)^T\ \delta\tilde{\boldsymbol{U}}^k$$

$$\tau_2 = (\boldsymbol{R}^{k-1} - \boldsymbol{R}^k)^T\ \delta\tilde{\boldsymbol{U}}^{k+1}$$

$$\tau_3 = (\delta\boldsymbol{U}^k)^T\ \boldsymbol{R}^k$$

$$\tau_4 = (\delta\boldsymbol{U}^k)^T\ (\boldsymbol{R}^{k-1} - \boldsymbol{R}^k)$$

9. Compute $A = \frac{\tau_1}{\tau_4}, \quad B = \frac{\tau_2}{\tau_4}, \quad C = \frac{\tau_3}{\tau_4}.$
10. Compute

$$\delta\boldsymbol{U}^{k+1} = (1+C)\ \delta\tilde{\boldsymbol{U}}^{k+1} + (-C)\ \delta\tilde{\boldsymbol{U}}^k + (C - (1+C)B + CA)\ \delta\boldsymbol{U}^k$$

11. Update $\boldsymbol{U}^{k+1} = \boldsymbol{U}^k + \delta\boldsymbol{U}^{k+1}$
12. Evaluate $\boldsymbol{R}^{k+1} = \boldsymbol{R}(\boldsymbol{U}^{k+1})$
13. Convergence control: if point $\boldsymbol{U}^{k+1}$ is good enough, exit.
14. Assign $k = k+1$ and go back to 7.

Table 5.5: Algorithm for the BFGSS method.

5.4 Numerical examples comparing total and partial versions of the Broyden method

In this section, we are going to present some examples comparing the behaviour of the total and partial versions of the SN Broyden method. The comparison between Quasi–Newton and Secant–Newton methods is made in Chapter 6.

Example TRUSS

As a first example, we consider again the test described in Figure 4.2. The analysis is performed exactly under the same conditions as in Section 4.4.

Figures 5.1 and 5.2 represent the behaviour of the two versions of the SN Broyden method for load steps 1 and 8.

Table 5.6 gives the total amount of iterations and CPU time required for the whole analysis.

Method	Iterations	CPU time
sbroyden_p	44	1 176
sbroyden_t	45	1 209

Table 5.6: Total costs of the SN Broyden methods for the truss test.

Remarks:

- The truss test is too simple for the two versions of the SN Broyden method to show a significant difference.

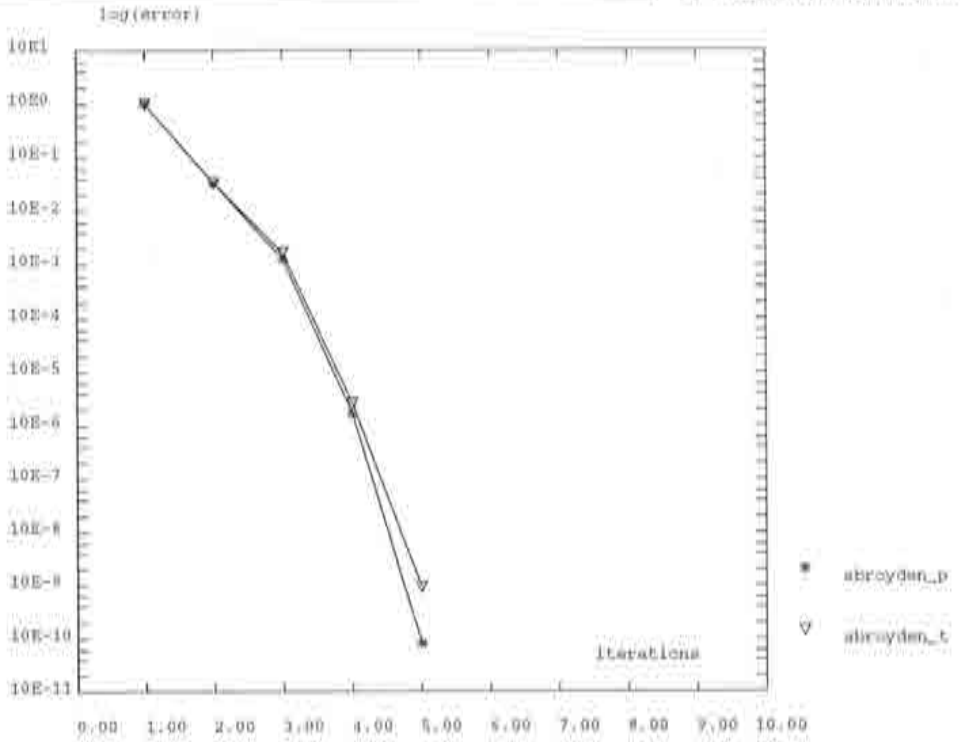


Figure 5.1: Truss test, load step 1. Comparison of the total and partial versions of the Broyden Secant-Newton method.

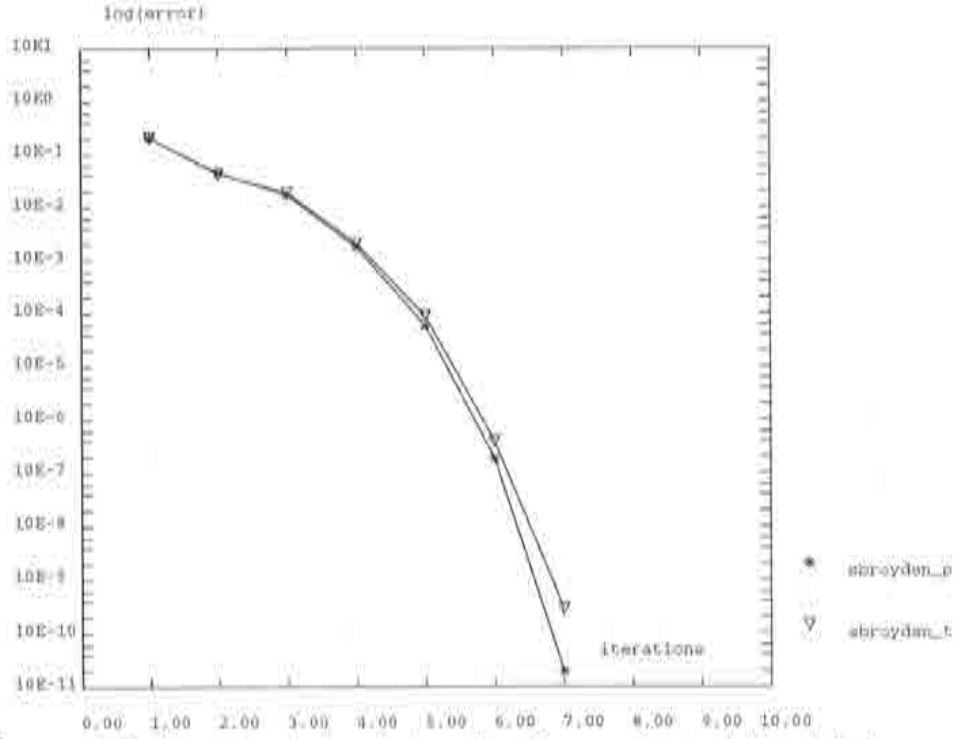


Figure 5.2: Truss test, load step 8. Comparison of the total and partial versions of the Broyden Secant-Newton method.

Example CYLINDER

Let us consider now the cylinder test introduced in Figure 4.7. This cylinder has been analysed in the same conditions described in Section 4.4 to compare the two versions of the SN Broyden method. Figures 5.3, 5.4 and 5.5 show the behaviour of the two options for load steps 5, 11 and 12.

Table 5.7 contains the number of iterations and CPU time associated to this analysis.

Method	Iterations	CPU time
sbroyden_p	73	2 392
sbroyden_t	88	2 656

Table 5.7: Total costs of the SN Broyden methods for the cylinder test.

Remarks:

- The partial version of the SN Broyden method is more clearly better than the total version for this test.
- Contrary to what happened for the Quasi–Newton Broyden-t method, the Secant–Newton Broyden-t method achieves convergence for load step 12.

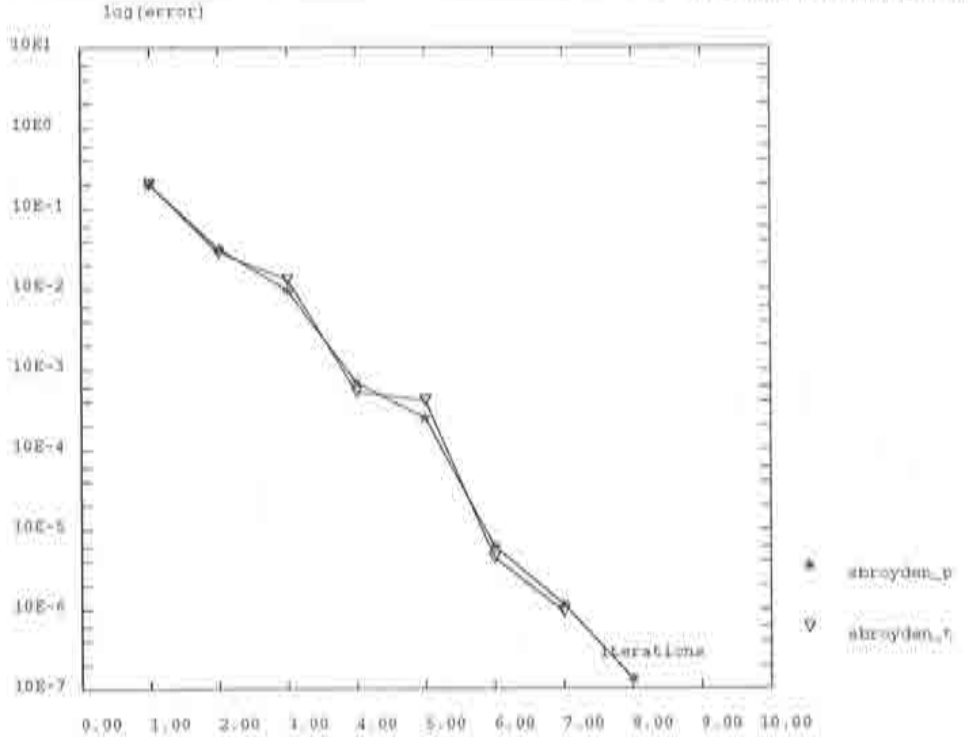


Figure 5.3: Cylinder test, load step 5. Comparison of the total and partial versions of the Broyden Secant-Newton method.

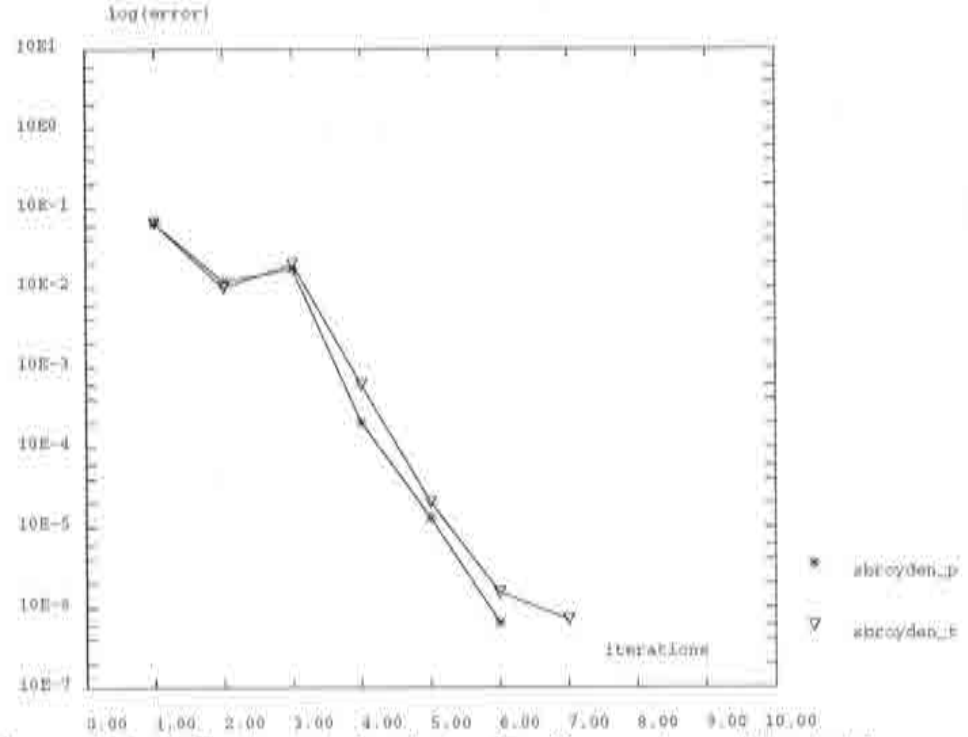


Figure 5.4: Cylinder test, load step 11. Comparison of the total and partial versions of the Broyden Secant-Newton method.

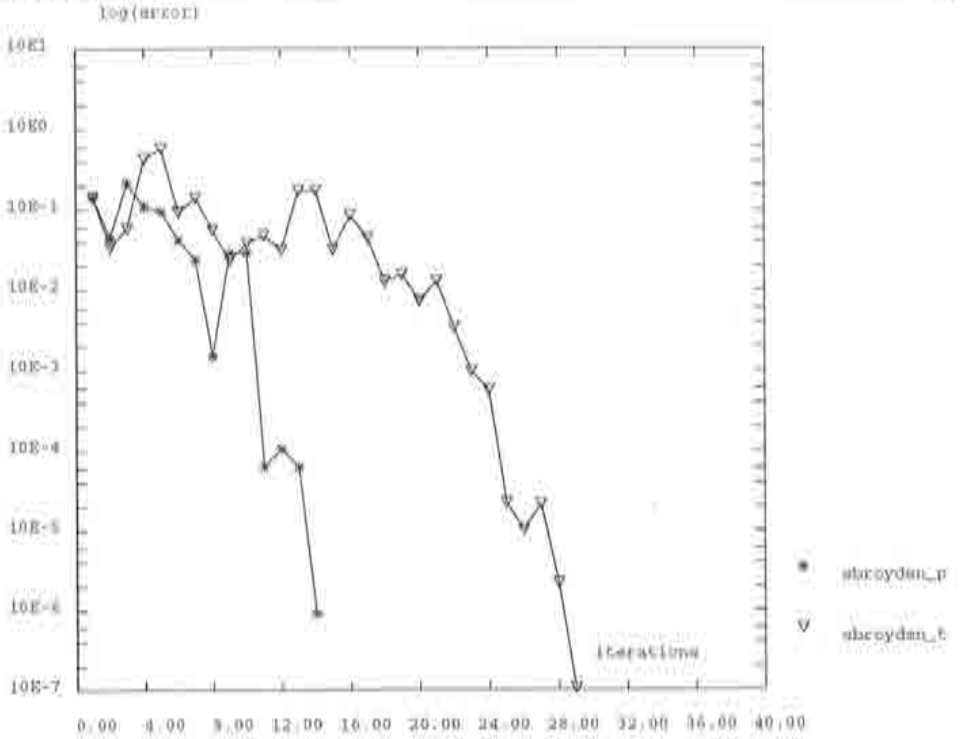


Figure 5.5: Cylinder test, load step 12. Comparison of the total and partial versions of the Broyden Secant-Newton method.

Example SHELL

The shell example of Figure 4.11 is studied now. Figures 5.6 and 5.7, associated to increments 26 and 41, are presented to illustrate the behaviour of the SN Broyden-t and Broyden-p methods. These figures are associated to the same sort of analysis performed in Section 4.4.

In Table 5.8, the total iterations and CPU times are presented.

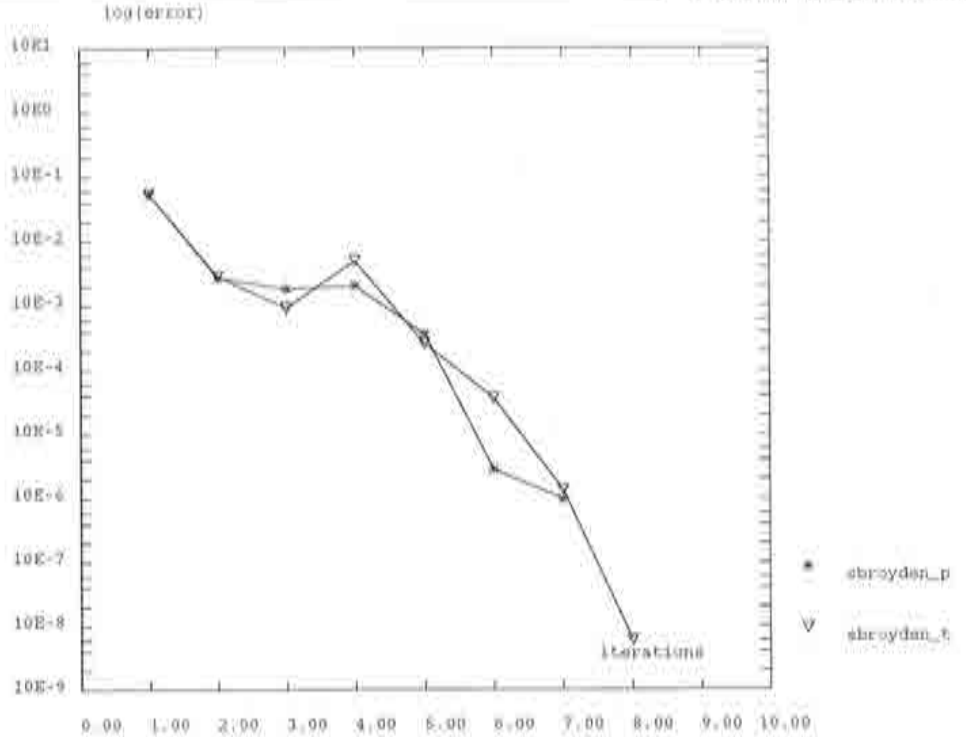


Figure 5.6: Shell test, load step 26. Comparison of the total and partial versions of the Broyden Secant-Newton method.

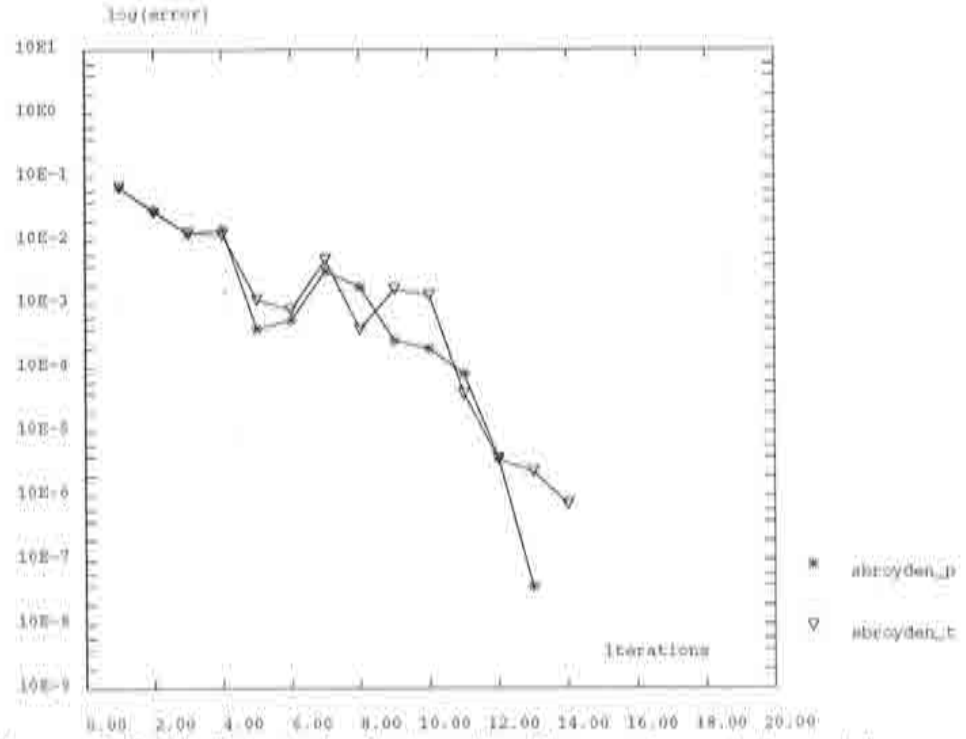


Figure 5.7: Shell test, load step 41. Comparison of the total and partial versions of the Broyden Secant-Newton method.

Method	Iterations	CPU time
sbroyden_p	524	15 789
sbroyden_t	542	16 298

Table 5.8: Total costs of the SN Broyden methods for the shell test.

Remarks:

• For the shell test, which is more complex than the truss or cylinder tests, the partial version of the SN Broyden method is just slightly better than the total version.

Example NECKING

We recall now the necking test presented in Figure 4.15. When solved with the two SN Broyden procedures under the same conditions as in Section 4.4, the results shown in Figures 5.8 , 5.9 and 5.10 are obtained. These three figures show the behaviour of the two versions for some of the initial steps, 2 and 3, and for the last converged step, 87.

Table 5.9 gives the total amount of iterations and CPU times.

Method	Iterations	CPU time
sbroyden_p	540	32 928
sbroyden_t	600	36 436

Table 5.9: Total costs of the SN Broyden methods for the necking test.

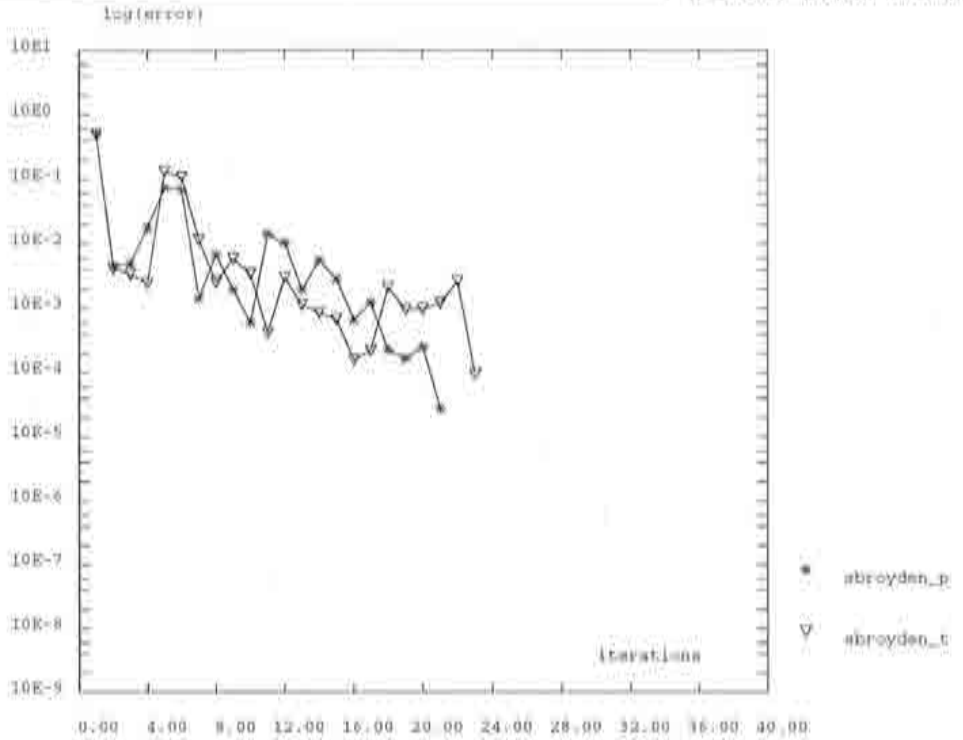


Figure 5.8: Necking test, load step 2. Comparison of the total and partial versions of the Broyden Secant-Newton method.

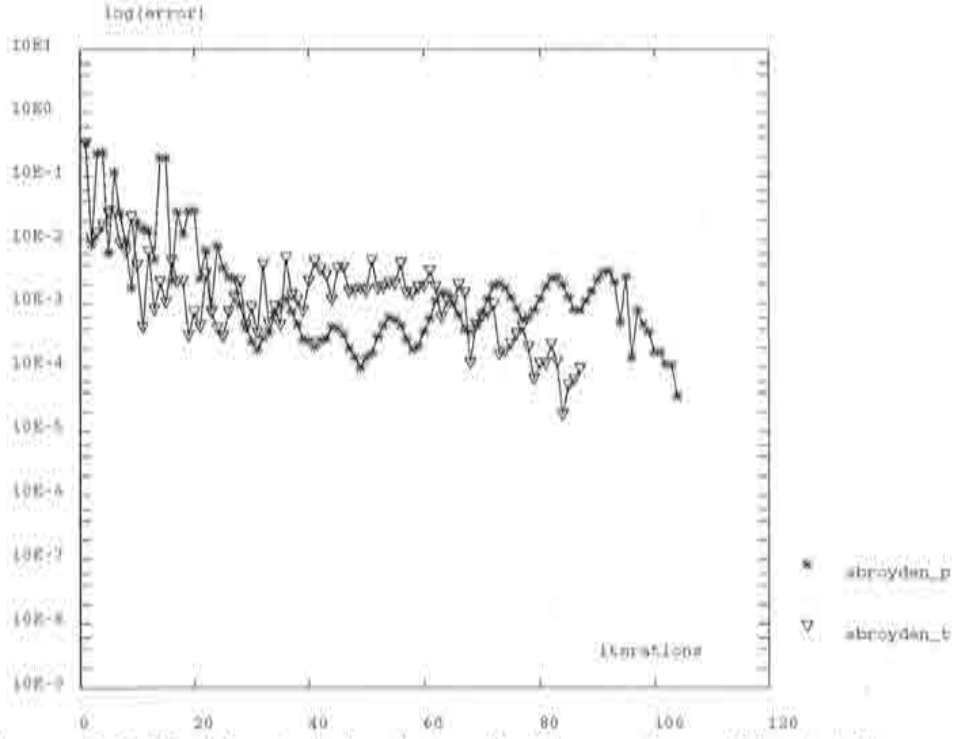


Figure 5.9: Necking test, load step 3. Comparison of the total and partial versions of the Broyden Secant-Newton method.

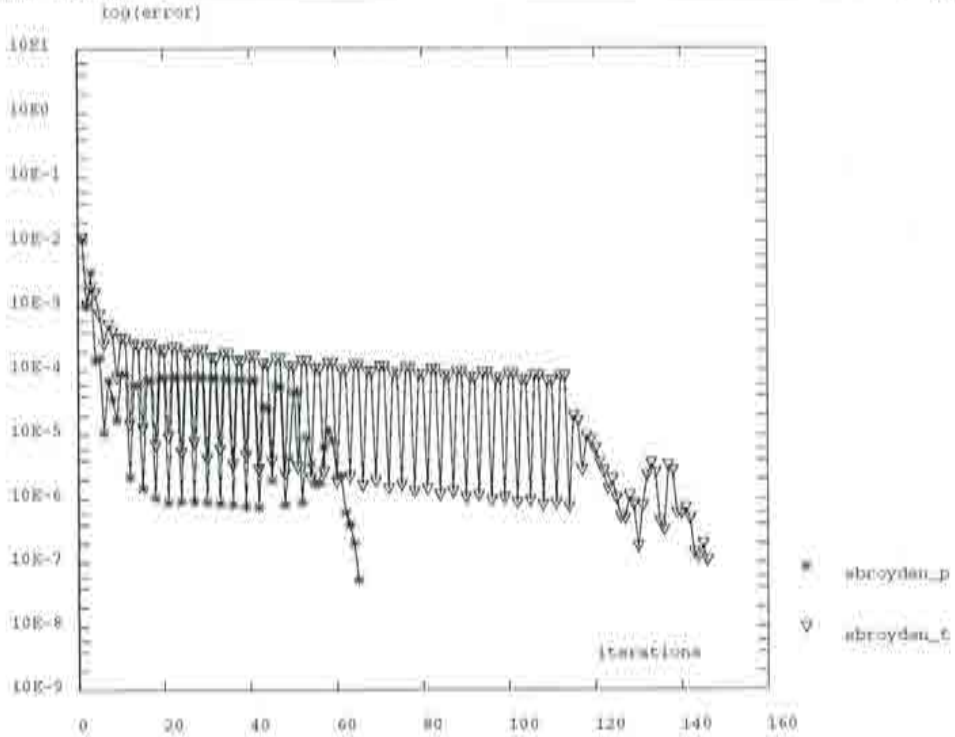


Figure 5.10: Necking test, load step 87. Comparison of the total and partial versions of the Broyden Secant-Newton method.

Remarks:

- Both versions of the Secant Broyden method diverge for the 87th of the planned 200 load steps. Thus, this test is too nonlinear for Secant methods.
- Up until load step 87, the partial version is just slightly better than the total option.

Example THERMIC

The nonlinear heat diffusion test presented in Figure 4.20 is reproduced with the Broyden Secant-Newton methods, to study their performance for non-symmetrical problems.

The value $m = 2$ employed in the constitutive equation (4.60) to compare the Broyden Quasi-Newton methods results in a too nonlinear problem and the SN Broyden do not converge. To allow the comparison between the two SN Broyden methods, the degree of nonlinearity is decreased by setting $m = 1.5$.

The total and partial versions of the Broyden Secant-Newton method are compared in Figure 5.11 and Table 5.10.

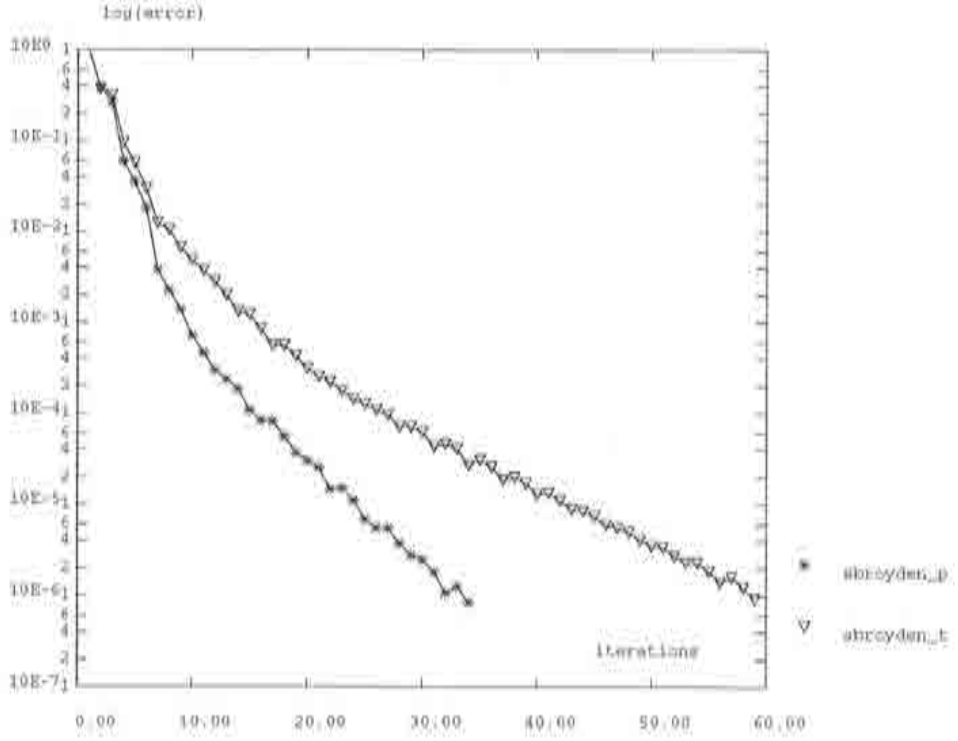


Figure 5.11: Thermic test. Comparison of the total and partial versions of the Broyden Secant-Newton method.

Method	Iterations	CPU time
sbroyden_p	34	588
sbroyden_t	59	1 051

Table 5.10: Total costs of the SN Broyden methods for the thermic test.

Remarks:

• For this test, the partial versions of both the QN and the SN Broyden method show a significant advantage over their total counterparts.

Chapter 6

Numerical examples involving Newton–Raphson, Quasi–Newton and Secant–Newton methods

In this chapter, various numerical examples will be shown which compare NR, QN and SN methods.

In the first part of the chapter, a comparison between Newton–Raphson methods is presented.

In the second part, Newton–Raphson methods are contrasted with Quasi–Newton and Secant–Newton methods.

Newton–Raphson methods

Example TRUSS

To begin with, we recall the truss test that was presented in Section 4.4, see Figure 4.2. In this case, the problem is solved using the full Newton–Raphson (fNR), modified Newton–Raphson (mNR) and initial stress (k0) methods.

Figures 6.1, 6.2 and 6.3 represent the behaviour of those methods for load steps 2, 5 and 8.

Table 6.1 gives the total amount of iterations and CPU time required for the whole problem (8 load steps).

Method	Iterations	CPU time
fNR	37	933
mNR	90	2 034
k0	200	4 235

Table 6.1: Total costs of the NR methods for the truss test.

Remarks:

- In Figures 6.1 to 6.3, the typical behaviour of the Newton–Raphson methods can be observed: quadratic convergence for the fNR and linear convergence for mNR and initial stress methods. Between the two modifications of the fNR, the mNR shows faster convergence than the k0 method.
- The modifications to the fNR method require an important computational cost with respect to the fNR.
- As the load level increases, the simplified methods mNR and k0 show increasingly bad behaviours. This is due to the fact that the problem grows more nonlinear as more load is applied.

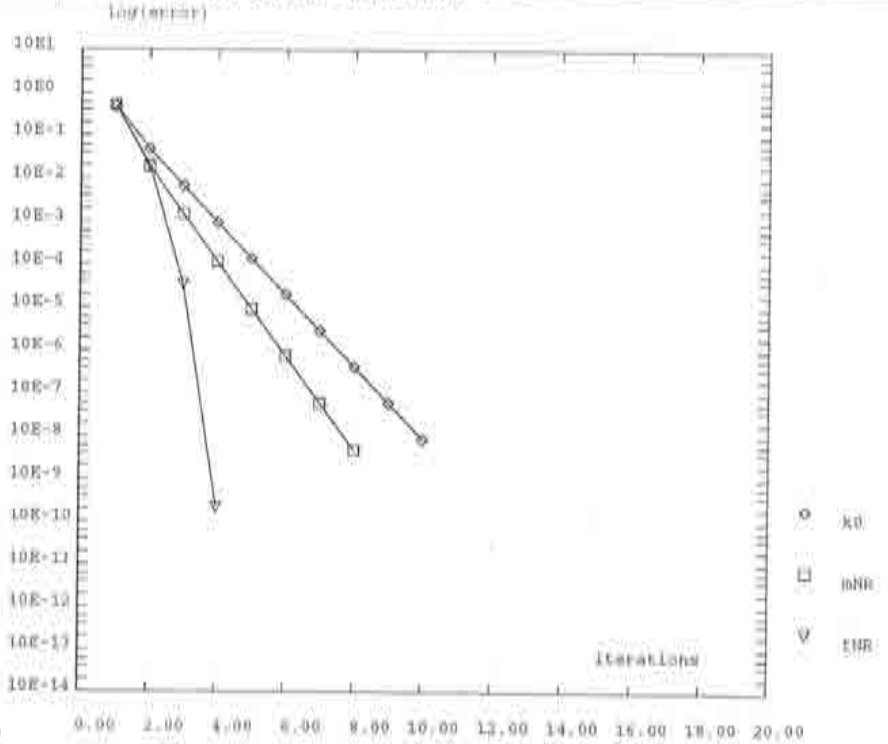


Figure 6.1: Truss test, load step 2. Comparison of full Newton-Raphson, modified Newton-Raphson and initial stress methods.

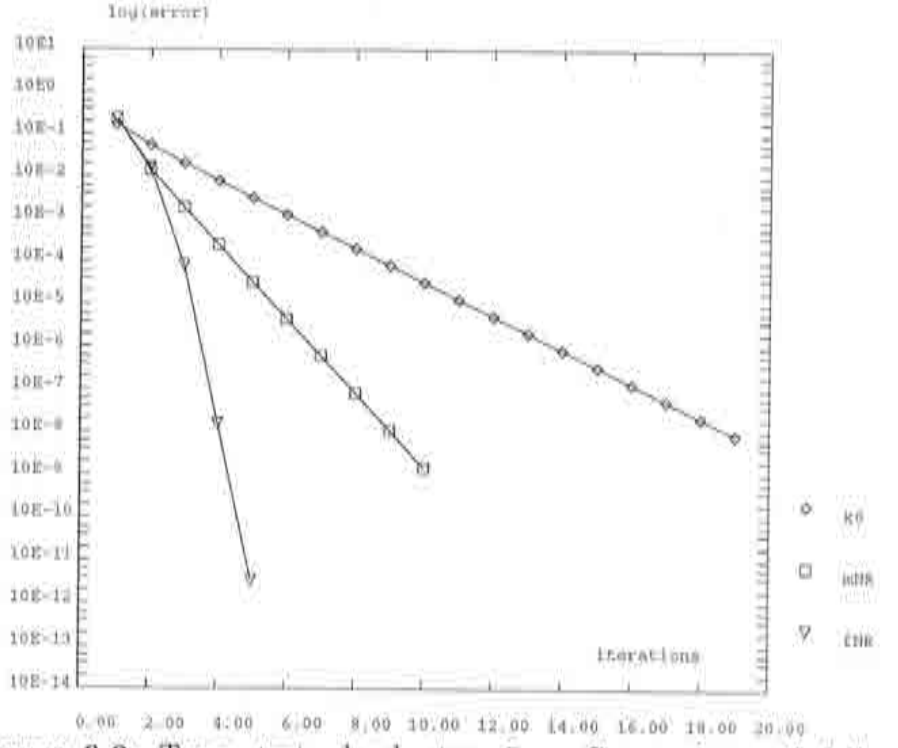


Figure 6.2: Truss test, load step 5. Comparison of full Newton-Raphson, modified Newton-Raphson and initial stress methods.

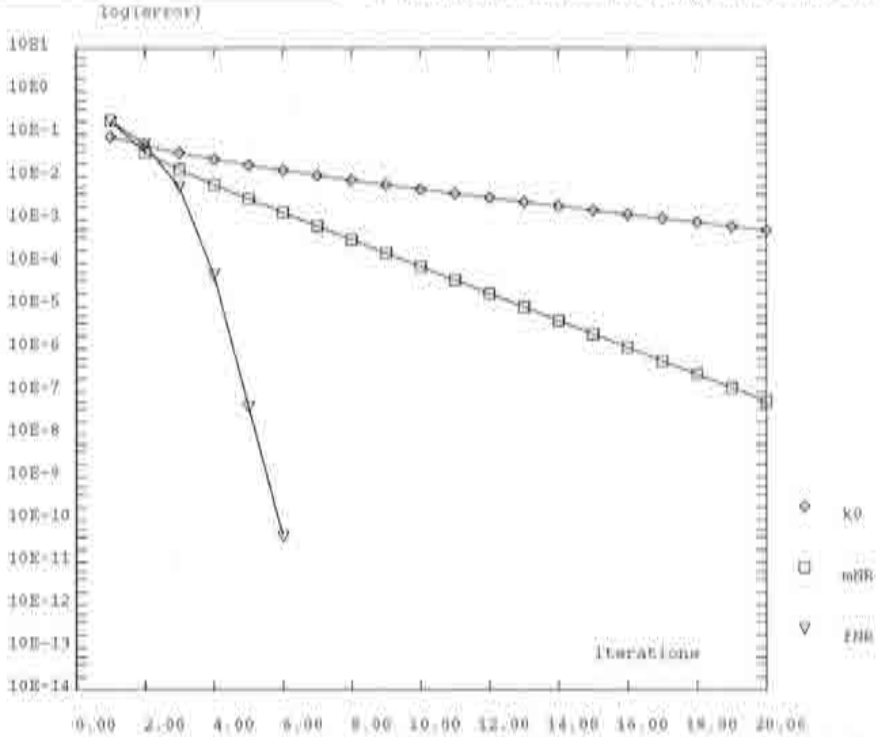


Figure 6.3: Truss test, load step 8. Comparison of full Newton-Raphson, modified Newton-Raphson and initial stress methods.

Example CYLINDER

We now analyse the cylinder case that was introduced in Chapter 4, see Figure 4.7. In Figures 6.4 to 6.7, the three Newton–Raphson methods are compared for load steps 4, 8, 11 and 12.

The total amount of iterations and CPU time are given in Table 6.2.

Method	Iterations	CPU time
fNR	55	1 779
mNR	651	16 023
k0	—	—

Table 6.2: Total costs of the NR methods for the cylinder test.

Remarks:

- For this example, the three Newton–Raphson methods show the expected behaviour.
- As it can be seen from the figures and definitely from the table, the modifications of the full Newton–Raphson method can turn up to have a prohibitive cost.
- The almost horizontal line that represents the k0 method for load step 12, Figure 6.7, indicates that this method fails to converge for this last increment.

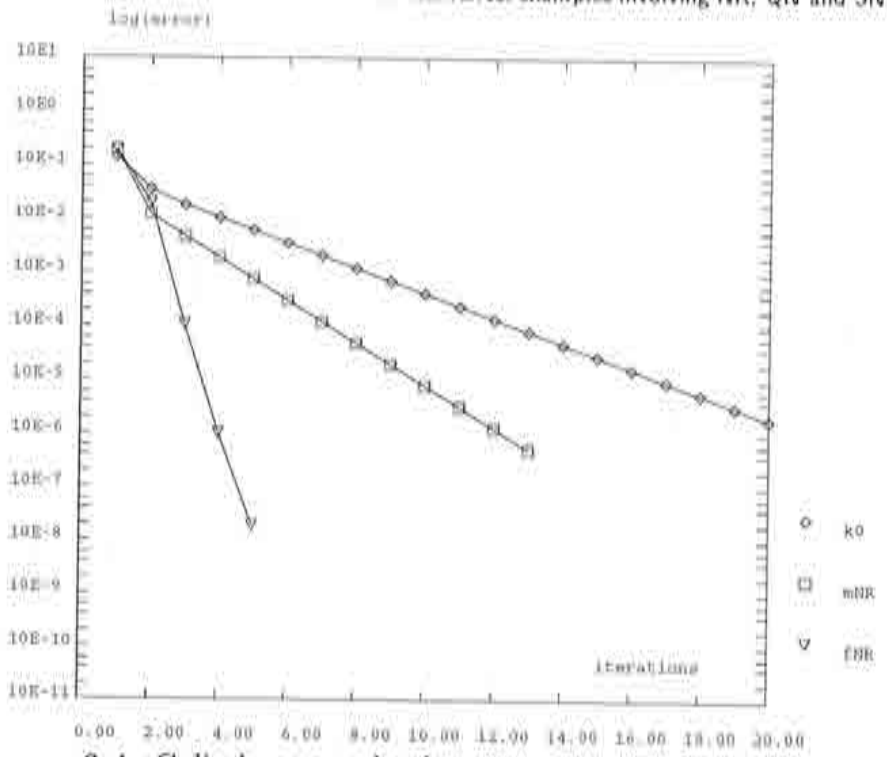


Figure 6.4: Cylinder test, load step 4. Comparison of full Newton-Raphson, modified Newton-Raphson and initial stress methods.

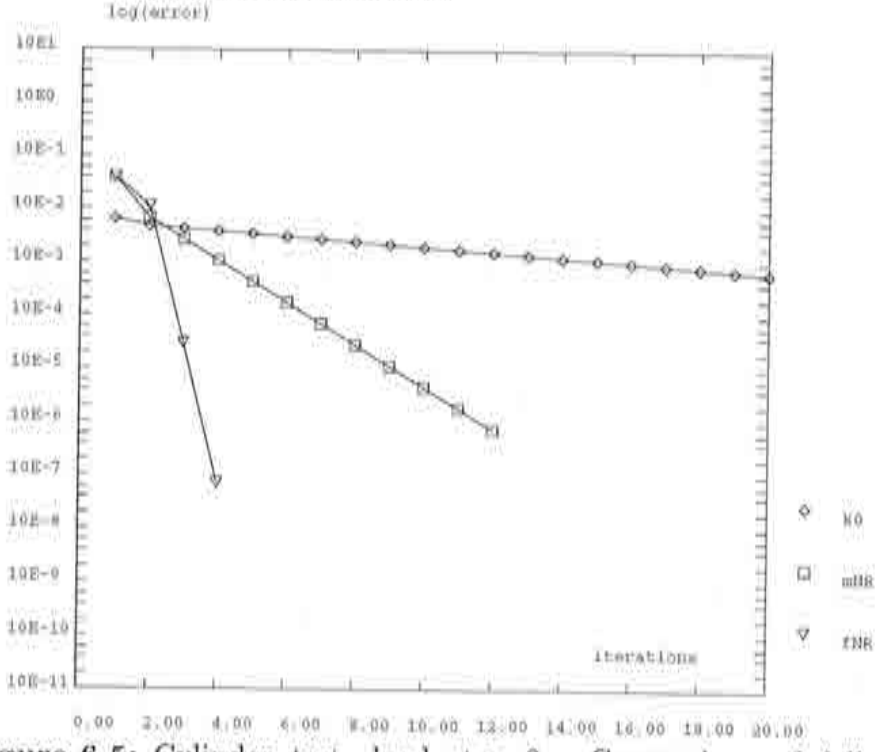


Figure 6.5: Cylinder test, load step 8. Comparison of full Newton-Raphson, modified Newton-Raphson and initial stress methods.

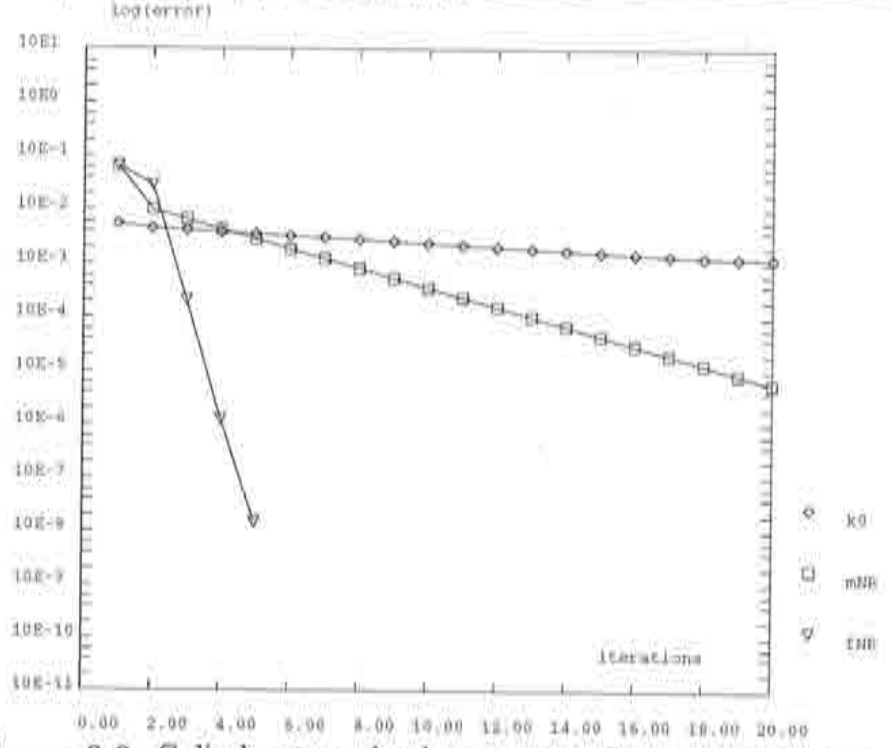


Figure 6.6: Cylinder test, load step 11. Comparison of full Newton-Raphson, modified Newton-Raphson and initial stress methods.

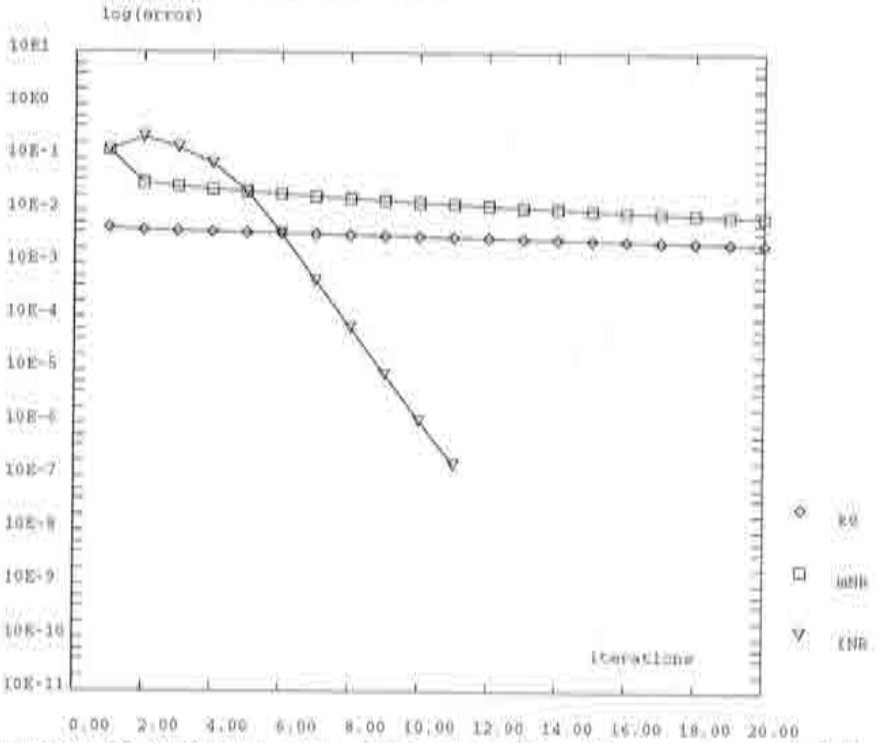


Figure 6.7: Cylinder test, load step 12. Comparison of full Newton-Raphson, modified Newton-Raphson and initial stress methods.

Newton–Raphson, Quasi–Newton and Secant–Newton methods

In this part of Chapter 6, we discuss the behaviour of the Broyden and BFGS methods in their QN and SN forms. Full Newton–Raphson and modified Newton–Raphson methods are also plotted as a reference.

For the whole analysis, the partial version of the Broyden QN and SN methods have been used, since this version has been shown to give slightly better results than the total version, see Sections 4.4 and 5.4.

Example TRUSS

In this case, we study the truss example, Figure 4.2. In Figures 6.8 and 6.9, the behaviours of the following methods are shown: fNR, mNR, inverse Broyden (partial version), BFGS, secant inverse Broyden (partial version) and BFGSS. These two figures correspond to the first and last load steps, respectively increments 1 and 8.

Table 6.3 sums up the costs of the 6 methods, in terms of iterations and total CPU time.

Method	Iterations	CPU time
fNR	37	933
mNR	90	2 034
qbroyden_p	44	1 239
bfgs	44	1 313
sbroyden_p	44	1 176
bfgss	44	1 185

Table 6.3: Total costs of the NR, QN and SN methods for the truss test.

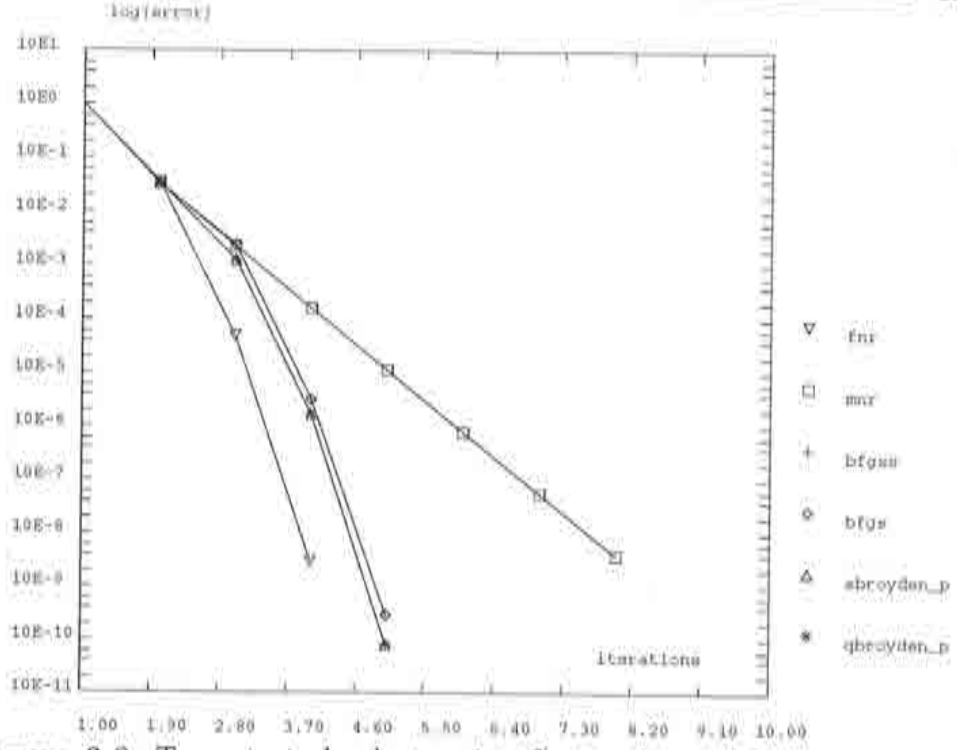


Figure 6.8: Truss test, load step 1. Comparison of Newton, Quasi-Newton and Secant-Newton methods.

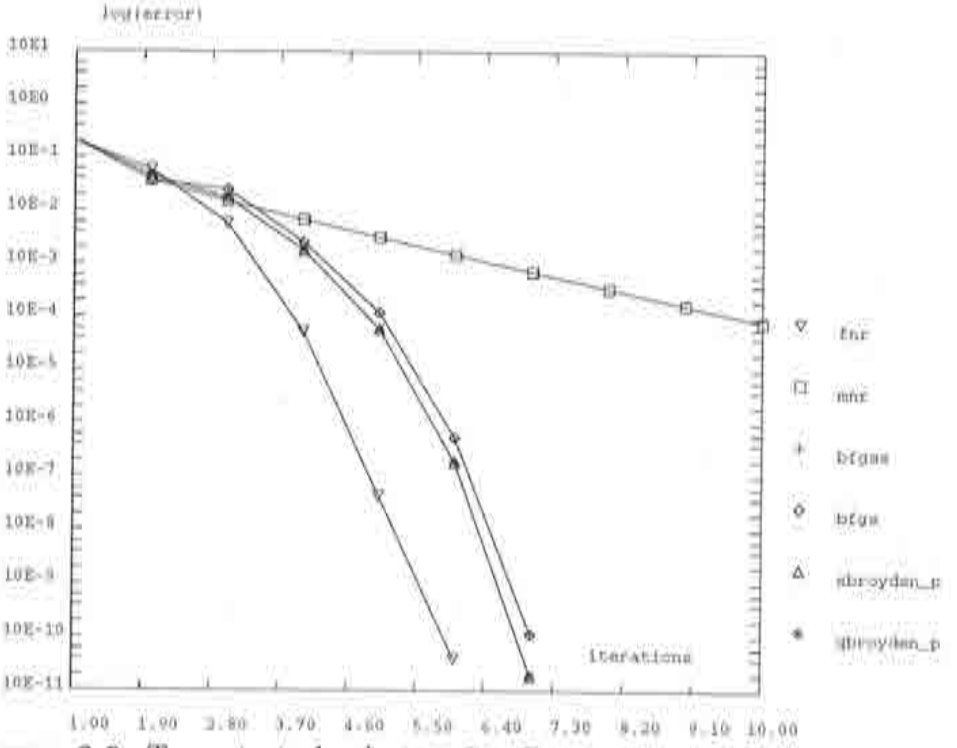


Figure 6.9: Truss test, load step 8. Comparison of Newton, Quasi-Newton and Secant-Newton methods.

Remarks:

- For this test, the figures show the expected behaviour of the QN and SN methods. That is, they present a rate of convergence that ranges from the quadratic convergence of the fNR to the linear convergence of the mNR.

- This one-dimensional test is too simple to discriminate between the Quasi–Newton and Secant–Newton methods: the QN and SN form of the methods are confounded and also Broydens and BFGSs show a practically identical behaviour.

- QN and SN methods have a similar total cost, which lies between those of the fNR and mNR methods.

- In this particular case, the fNR method has required the computation and factorisation of 37 tangent stiffness matrices, see Table 6.3, whereas the secant Broyden method has calculated just 8 tangent stiffness matrices in all. Nevertheless, the fNR still has a lower cost. This can be explained by regarding the nature of the test: the truss example is a one-dimensional problem; this means that the computation of a tangent stiffness matrix for this problem is a simple task. Hence the advantage of the fNR.

Example CYLINDER

When solving the cylinder problem, see Figure 4.7, the curves in Figures 6.10 and 6.11, associated to steps 4 and 12, are obtained.

The iterations and CPU times demanded by each method are given in Table 6.4.

Method	Iterations	CPU time
fNR	55	1 779
mNR	651	16 023
qbroyden_p	73	2 430
bfgs	61	2 142
sbroyden_p	73	2 392
bfgss	67	2 116

Table 6.4: Total costs of the NR, QN and SN methods for the cylinder test.

Remarks:

- For this example also the expected results for the QN and SN methods are obtained.
- For load step 4, all the QN and SN methods show a similar behaviour. A slightly better response is observed for the BFGS.
- For load step 12, the QN and SN forms of each method are difficult to differentiate from one another. BFGS and BFGSS show a better behaviour than the Broydens.
- The better behaviour of the BFGS and BFGSS methods is confirmed in Table 6.4.
- As for secant methods in general, although they have been developed as refinements of the mNR method, they turn up to supply a much better result that the mNR in its original form.

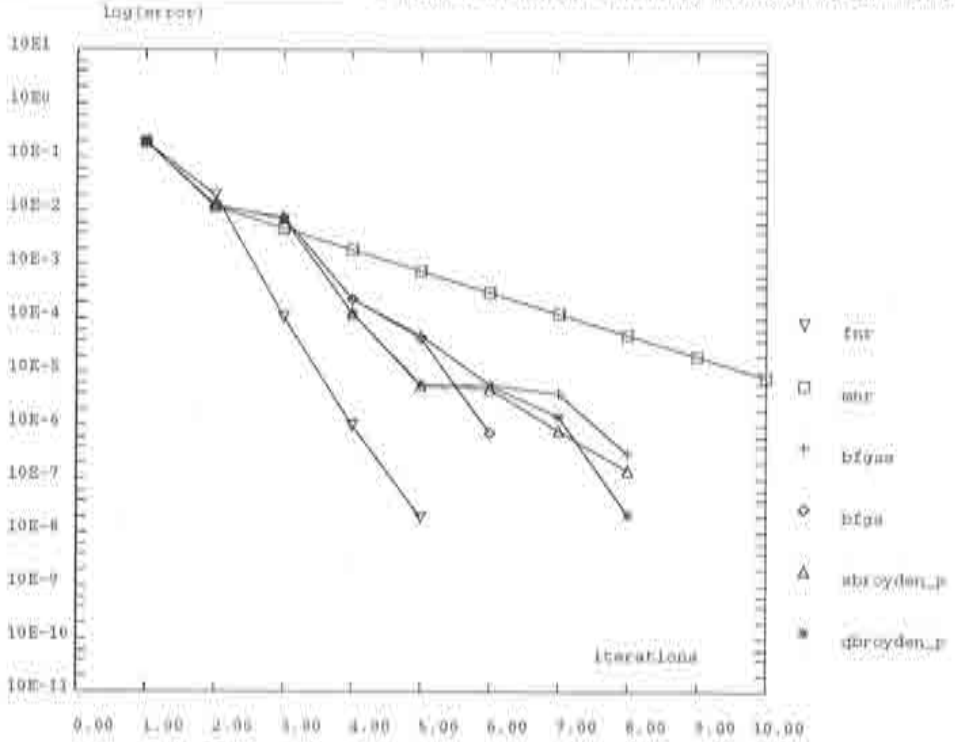


Figure 6.10: Cylinder test, load step 4. Comparison of Newton, Quasi-Newton and Secant-Newton methods.

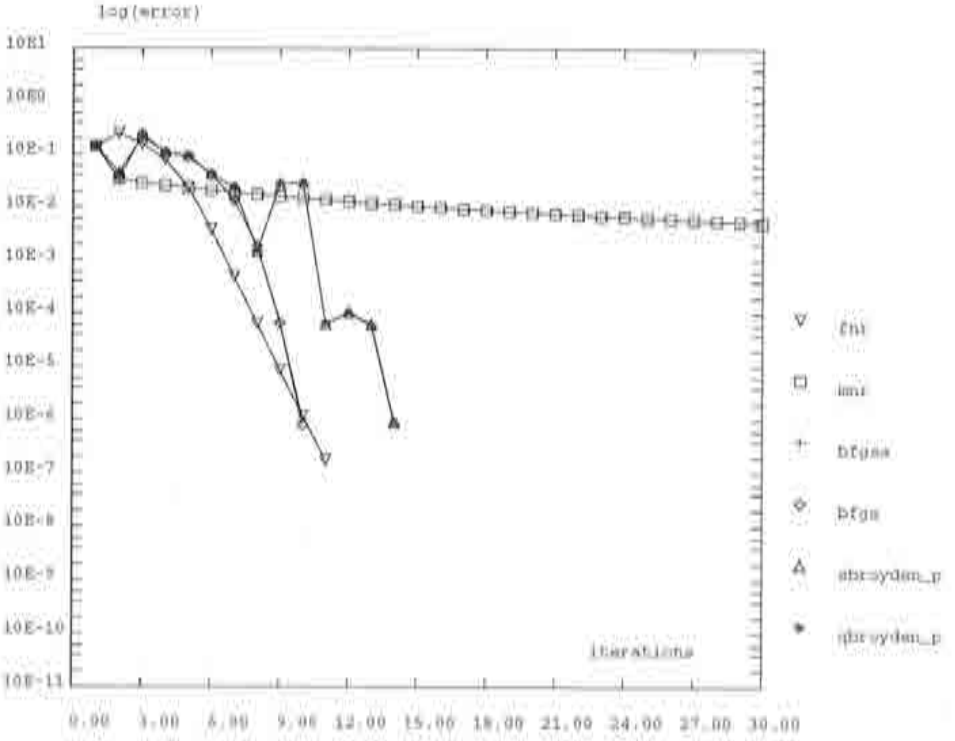


Figure 6.11: Cylinder test, load step 12. Comparison of Newton, Quasi-Newton and Secant-Newton methods.

Example SHELL

For the shell example, see Figure 4.11, load steps 1, 17, 26 and 38 are shown in Figures 6.12 to 6.15.

The amount of iterations and CPU times are accounted for in Table 6.5.

Method	Iterations	CPU time
fNR	411	11 967
mNR	—	—
qbroyden_p	504	16 241
bfgs	458	15 279
sbroyden_p	524	15 789
bfgss	512	15 706

Table 6.5: Total costs of the NR, QN and SN methods for the shell test.

Remarks:

- This test is too nonlinear for the mNR to achieve convergence.
- BFGS requires less iterations than the rest of QN/SN methods, but the total CPU time is very similar. An explanation of this fact is that the BFGS method uses a more sophisticated update which involves SPD stiffness matrices, thus resulting in a smaller amount of iterations; but each iteration costs more than those associated to the rest of the QN/SN methods, and therefore the total CPU time is higher.
- Since this example is reasonably small (6 axisymmetrical elements), the computation of tangent stiffness matrices involves a small cost. It is for this reason that the fNR method requires both fewer iterations and CPU time.

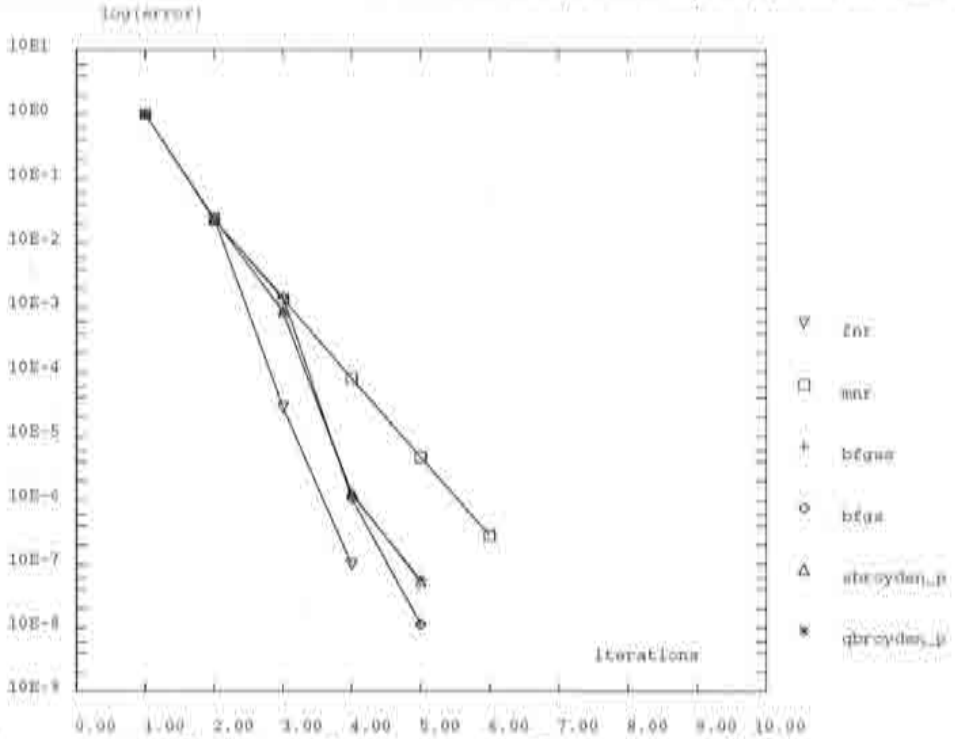


Figure 6.12: Shell test, load step 1. Comparison of Newton, Quasi-Newton and Secant-Newton methods.

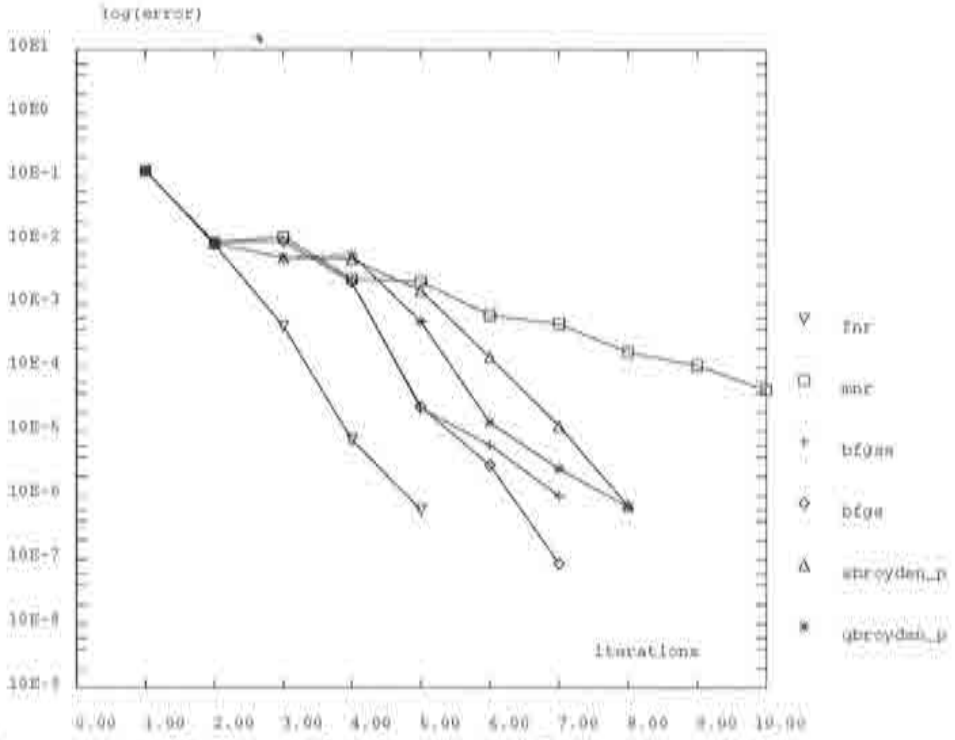


Figure 6.13: Shell test, load step 17. Comparison of Newton, Quasi-Newton and Secant-Newton methods.

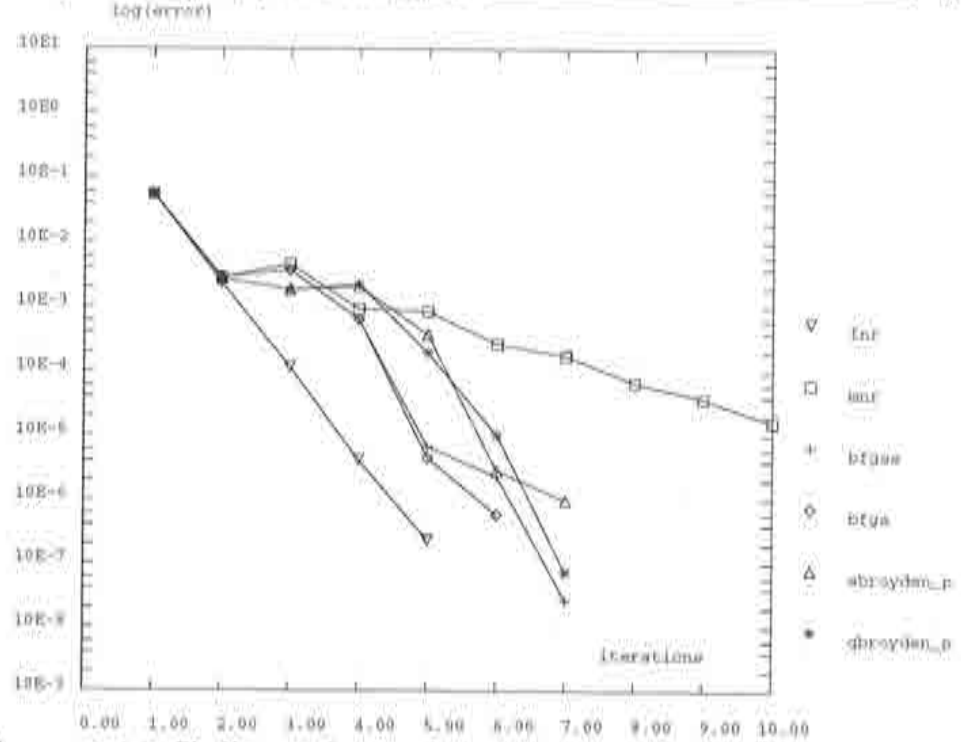


Figure 6.14: Shell test, load step 26. Comparison of Newton, Quasi-Newton and Secant-Newton methods.

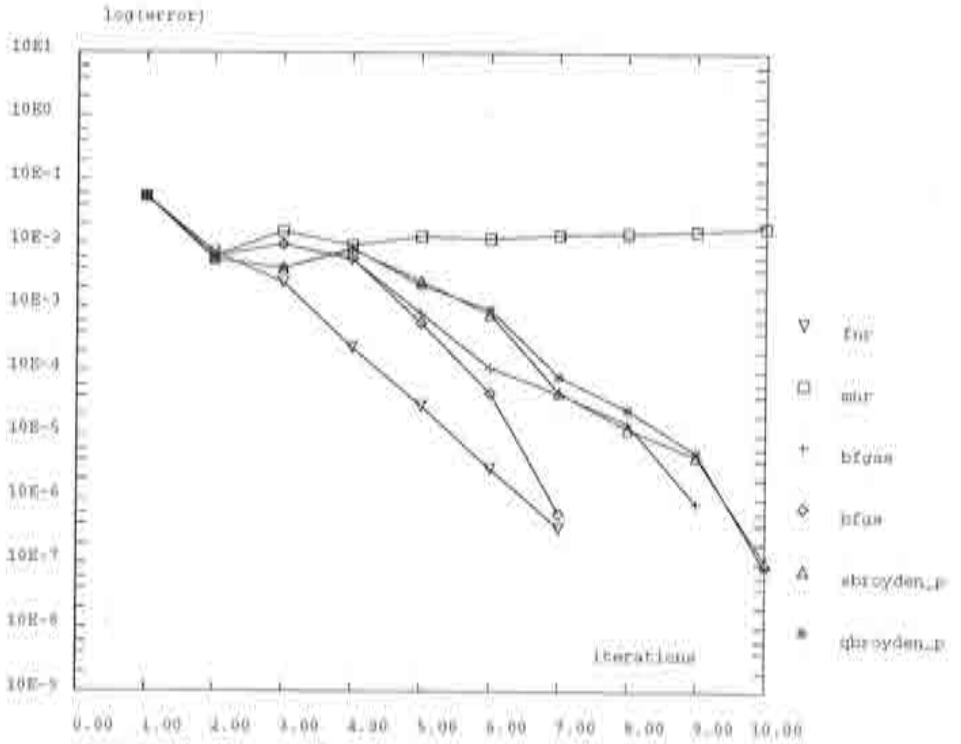


Figure 6.15: Shell test, load step 38. Comparison of Newton, Quasi-Newton and Secant-Newton methods.

Example NECKING

For this example, we consider the necking problem in Figure 4.15. The results associated to load steps 2 and 3 are given in Figures 6.16 and 6.17, respectively. In Table 6.6, the total iterations and CPU times for the NR, QN and SN methods are added up.

Method	Iterations	CPU time
fNR	2 632	225 336
mNR	—	—
qbroyden_p	1 914	137 826
bfgs	1 583	115 774
sbroyden_p	—	—
bfgss	—	—

Table 6.6: Total costs of the NR, QN and SN methods for the necking test.

Remarks:

- This is a highly nonlinear problem, for which neither the mNR nor the SN methods achieve convergence.
- The high amount of degrees of freedom of the problem causes the fNR to have a high cost with respect to the QN.
- Between the QN methods, the BFGS method shows a better behaviour, both in terms of iterations and of CPU time, since it uses SPD iteration matrices.

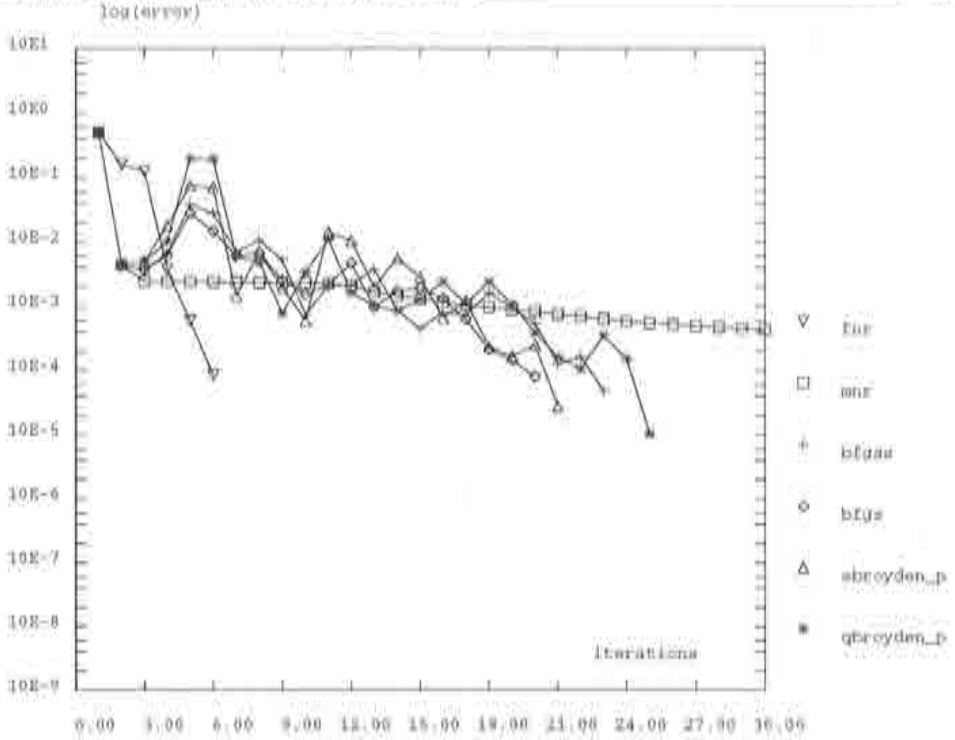


Figure 6.16: Necking test, load step 2. Comparison of Newton, Quasi-Newton and Secant-Newton methods.

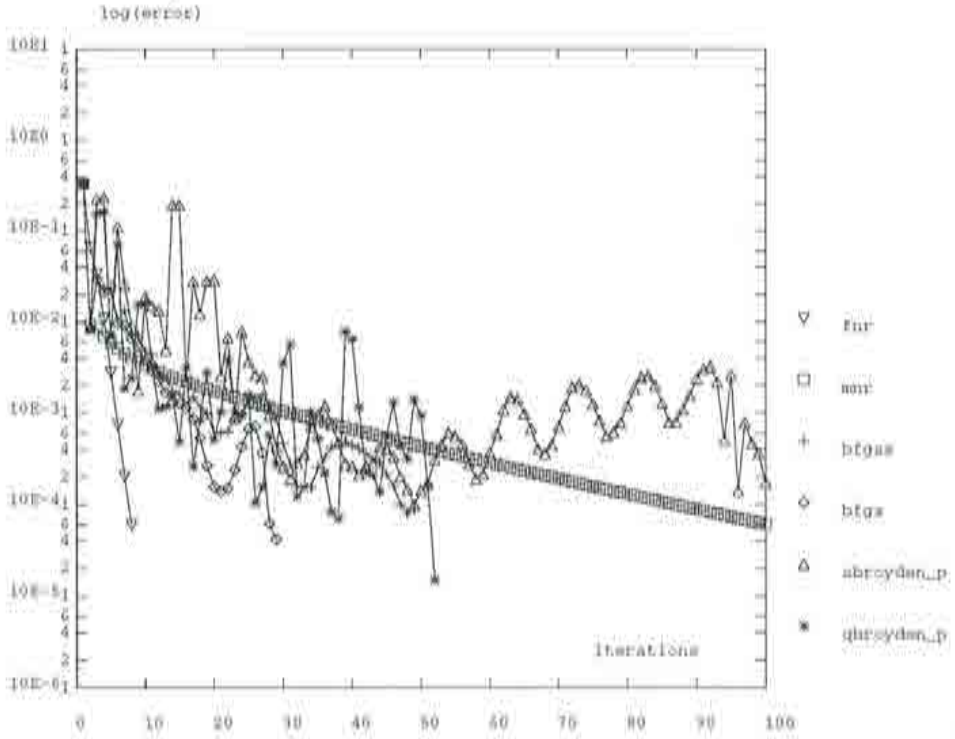


Figure 6.17: Necking test, load step 3. Comparison of Newton, Quasi-Newton and Secant-Newton methods.

Example THERMIC

The last example shown in this chapter is the thermic test presented in Figure 4.20. The goal that is pursued with this example is to compare the behaviour of the BFGS and the Broyden methods, both QN and SN, for a non-symmetrical problem.

Figure 6.18 and Table 6.7 show the results obtained using the Quasi-Newton methods with $m = 2$ in the constitutive equation (4.60). Secant-Newton methods are compared, with $m = 1.5$, in Figure 6.19 and Table 6.8. For both problems, the partial version of the Broyden method tends to be superior to the BFGS method, as expected for a non-symmetrical problem.

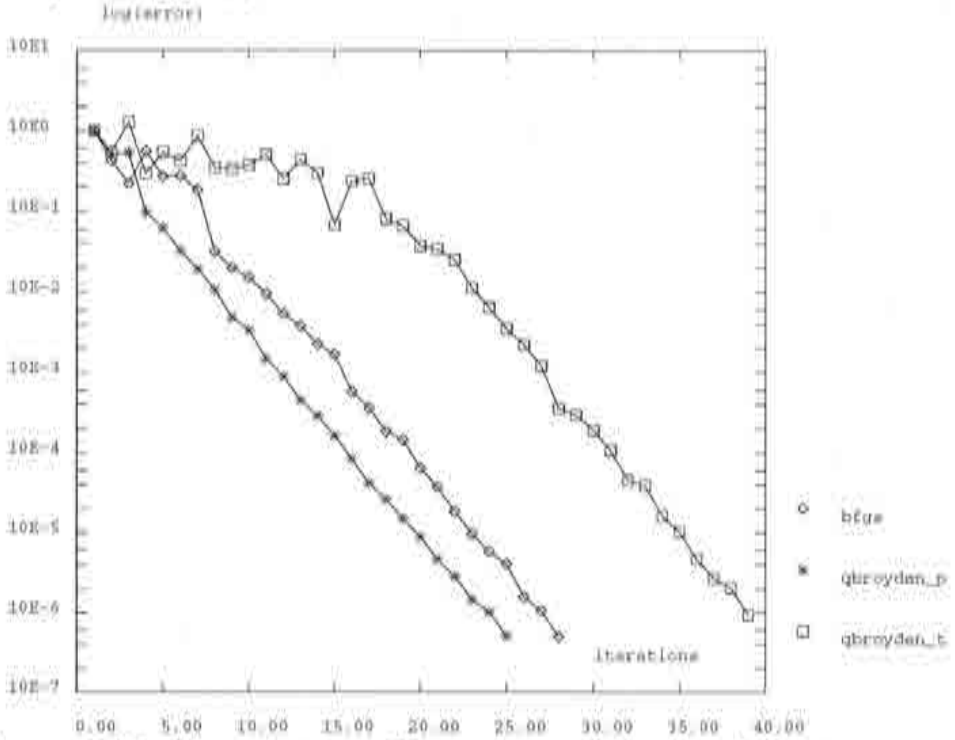


Figure 6.18: Thermic test. Comparison of the Quasi-Newton methods.

Method	Iterations	CPU time
qbroyden_p	25	654
qbroyden_t	39	1 373
bfgs	28	1 015

Table 6.7: Total costs of the Quasi-Newton methods for the thermic test

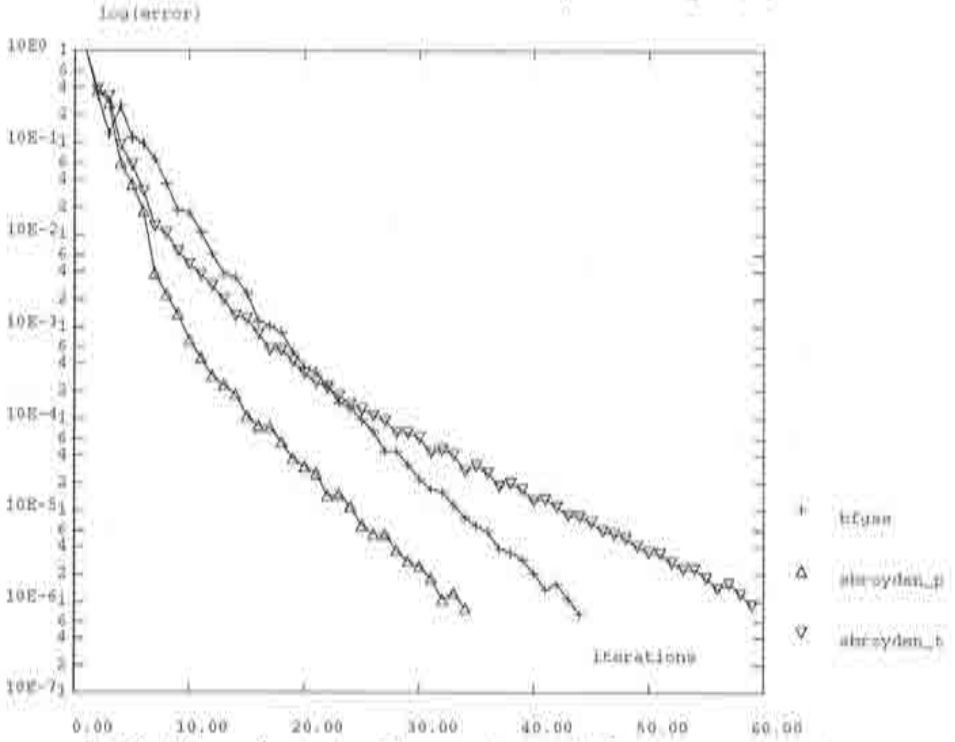


Figure 6.19: Thermic test. Comparison of the Secant-Newton methods.

Method	Iterations	CPU time
sbroyden_p	34	588
sbroyden_t	59	1 051
bfgss	44	779

Table 6.8: Total costs of the Secant-Newton methods for the thermic test.

Chapter 7

Line searches

7.1 Theory and detailed flowchart

So far, we have discussed several methods that supply different solution techniques for nonlinear problems. In this chapter, we will be concerned with performing simple modifications of those solutions to yield better results.

We begin by assuming that $\boldsymbol{\delta u}^{k+1}$ is the correction vector provided by some NR, QN or SN method at iteration k. Up until this point, displacement updates have been performed following the expression

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^{k} + \boldsymbol{\delta u}^{k+1}. \tag{7.1}$$

What we are going to do now is keep the *advance direction* determined by $\boldsymbol{\delta u}^{k+1}$, but modify its modulus so that a different update

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^{k} + \eta_{k+1}\, \boldsymbol{\delta u}^{k+1} \tag{7.2}$$

is obtained. The scale factor η_{k+1} in equation (7.2) is the *advance length*, [10,18,27], which has been implicitly set to 1 in the previous chapters . Obviously,

we want to compute η_{k+1} so that vector $\boldsymbol{u}^{k+1}$ in (7.2) be better than the one in (7.1). In other words, we will be *searching* for a better solution $\boldsymbol{u}^{k+1}$ along the *line* determined by point $\boldsymbol{u}^k$ and the direction $\boldsymbol{\delta u}^{k+1}$.

From the beginning, our main objective has consisted of finding a displacement vector $\boldsymbol{u}^{k+1}$ for which $\boldsymbol{r}(\boldsymbol{u}^{k+1}) = \boldsymbol{r}(\boldsymbol{u}^k + \boldsymbol{\delta u}^{k+1}) = \boldsymbol{0}$. In the line-search context, since $\boldsymbol{u}^k$ and $\boldsymbol{\delta u}^{k+1}$ are fixed, all functions of $\boldsymbol{u}^{k+1}$ can be rewritten, thanks to equation (7.2), as functions of η_{k+1}. Thus, our problem reduces to finding the value of the scalar η_{k+1} for which

$$\boldsymbol{r}(\eta_{k+1}) = \boldsymbol{r}(\boldsymbol{u}^k + \eta_{k+1}\ \boldsymbol{\delta u}^{k+1}) = \boldsymbol{0}. \tag{7.3}$$

What happens is that, in general, the real solution $\boldsymbol{u}^{k+1}$ to $\boldsymbol{r}(\boldsymbol{u}^{k+1}) = \boldsymbol{0}$ does not lie on the search line. This means that we can *minimise* the vectorial function of residual forces given in (7.3), but not cancel it.

A typical alternative requirement consists of finding the value of η_{k+1} that gives the minimum absolute value of the function ψ defined by

$$\psi(\eta_{k+1}) = (\boldsymbol{\delta u}^{k+1})^T\ \boldsymbol{r}(\boldsymbol{u}^k + \eta_{k+1}\ \boldsymbol{\delta u}^{k+1}). \tag{7.4}$$

The expression of ψ given in (7.4) is a linearised version of the *total potential energy* associated to point $\boldsymbol{u}^k + \eta_{k+1}\ \boldsymbol{\delta u}^{k+1}$, [10]. This means that we have transformed the vectorial problem of minimising the out-of-balance forces in (7.3) into the scalar problem of minimising the energy function (7.4).

We could use any minimisation technique to obtain η_{k+1} for which $|\psi(\eta_{k+1})|$ is the least possible, [14]. Nevertheless, we want to use a *simple* strategy, since the evaluation of function ψ involves the evaluation of a vector of residual forces, which is an expensive operation from a computational point of view; furthermore, more accurate values of η_{k+1} will give smaller norms of the residual forces, but, as we have already seen, not cancel them. For this reason, most of the line-search procedures, [10,31], demand for the modulus of $\psi(\eta_{k+1})$ just to be small in comparison to the modulus of $\psi(0)$, *i.e.*,

$$|\psi(\eta_{k+1})| < \varepsilon\ |\psi(0)|, \tag{7.5}$$

with ε being the *line-search tolerance*. This tolerance is set to large values that range from 0.5, [18], to 0.8, [10], thus not making (7.5) a restrictive condition.

To find an η_{k+1} that meets equation (7.5), we want to use a technique which profits of the fact that the values of $\psi(0)$ and $\psi(1)$ are almost known:

- For $\eta_{k+1} = 0$, the residual forces equal $\boldsymbol{r}(\eta_{k+1}) = \boldsymbol{r}(0) = \boldsymbol{r}(\boldsymbol{u}^k) = \boldsymbol{r}^k$, where vector $\boldsymbol{r}^k$ can be obtained from the previous iteration.

- For $\eta_{k+1} = 1$, we have $\boldsymbol{r}(\eta_{k+1}) = \boldsymbol{r}(1) = \boldsymbol{r}(\boldsymbol{u}^{k+1}) = \boldsymbol{r}^{k+1}$, which needs to be calculated anyhow at the end of iteration k+1.

Therefore, to obtain the values of

$$\psi(0) = (\boldsymbol{\delta u}^{k+1})^T \, \boldsymbol{r}^k \tag{7.6}$$

$$\psi(1) = (\boldsymbol{\delta u}^{k+1})^T \, \boldsymbol{r}^{k+1}, \tag{7.7}$$

we will only need to perform two scalar products.

The first thing we need to do is check if condition (7.5) is satisfied for $\eta_{k+1} = 1$. If this is not the case, we perform a simple *interpolation* in the plane (η, ψ) between points $\eta_{k+1} = 0$ and $\eta_{k+1} = 1$, Figure 7.1. This results in

$$\eta_{k+1,1} = \frac{\psi(0)}{\psi(0) - \psi(1)} \tag{7.8}$$

where the additional subindex 1 in $\eta_{k+1,1}$ indicates that (7.8) is a prediction of η_{k+1}.

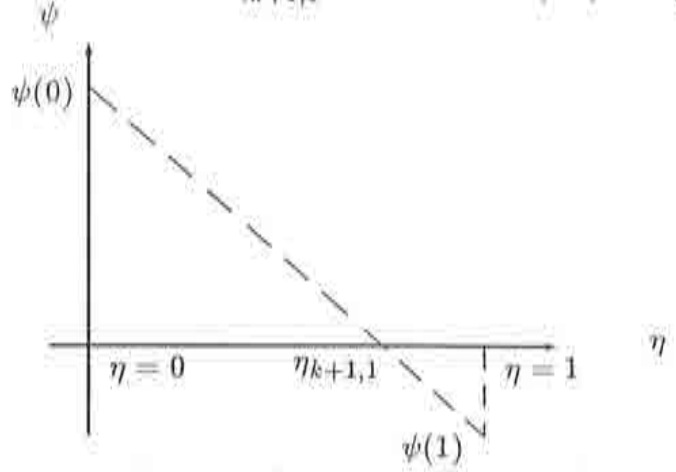


Figure 7.1: Line-search interpolation procedure.

A common approach is to compute $\eta_{k+1,1}$ from (7.8), update $\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \eta_{k+1,1}\, \boldsymbol{\delta u}^{k+1}$ and pass on to the next iteration $k+2$. However, if condition (7.5)

$$|\psi(\eta_{k+1,1})| < \varepsilon\, |\psi(0)|$$

is not met, we can just as well go on performing 'line-search iterations' by interpolating between points $(0\ ,\ \psi(0))$ and $(\eta_{k+1,1}\ ,\ \psi(\eta_{k+1,1}))$, then between points $(0\ ,\ \psi(0))$ and $(\eta_{k+1,2}\ ,\ \psi(\eta_{k+1,2}))$, and so on. In this case, the value for the step-length $\eta_{k+1,i}$ associated to line-search iteration i is computed as

$$\eta_{k+1,i} = \eta_{k+1,i-1}\, \frac{\psi(0)}{\psi(0) - \psi(\eta_{k+1,i-1})}. \tag{7.9}$$

If the ratio $\psi(\eta_{k+1,i-1})/\psi(0)$ is positive, that is, if $\psi(0)$ and $\psi(\eta_{k+1,i-1})$ have the same sign, then an *extrapolation* instead of an interpolation is automatically performed. This extrapolation can give very large values of $\eta_{k+1,i}$, which in general will not lead to good results. To avoid this situation, we specify an upper bound for η_{k+1}.

In a similar way, for large negative values of $\psi(\eta_{k+1,i-1})/\psi(0)$, we will get a value of $\eta_{k+1,i+1}$ that is very close to zero. This is also a bad situation, as point $\boldsymbol{u}^{k+1}$ will be placed very near to $\boldsymbol{u}^k$ and thus the algorithm will not progress. Consequently, we also specify a lower bound for η_{k+1}.

If a previously set number of line-search iterations is not sufficient to verify the line-search convergence condition (7.5), we are going to update

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \eta_{k+1,\mathrm{opt}}\, \boldsymbol{\delta u}^{k+1} \tag{7.10}$$

and pass on to the next iteration $k+2$. The term $\eta_{k+1,\mathrm{opt}}$ in (7.10) represents the *optimum* value of η_{k+1}, which yields the least ratio $|\psi(\eta_{k+1})|\ /\ |\psi(0)|$.

Figure 7.2 gives a flowchart that lists all the instructions needed to perform a complete line search.

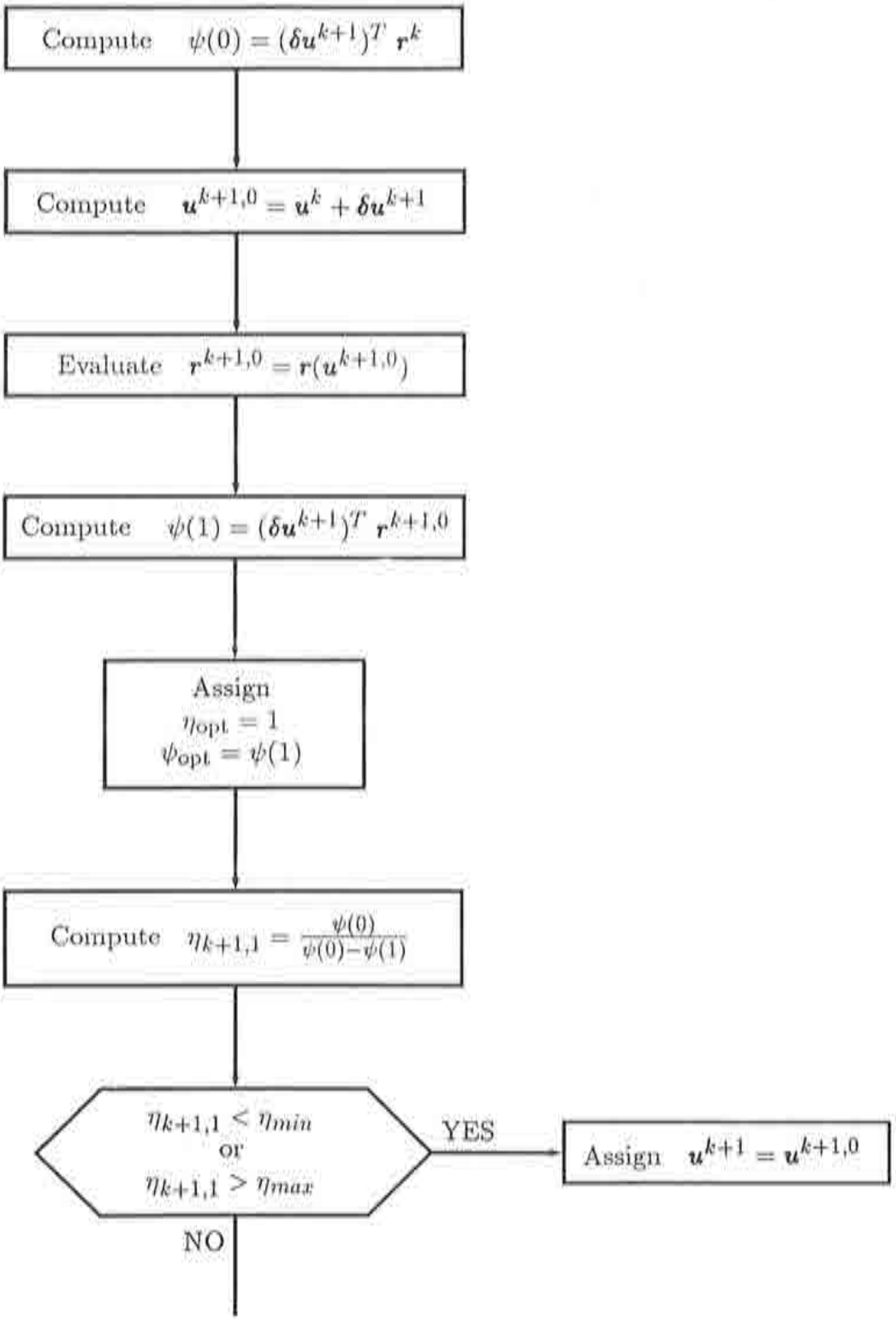


Figure 7.2: Flowchart for the line-search procedure.

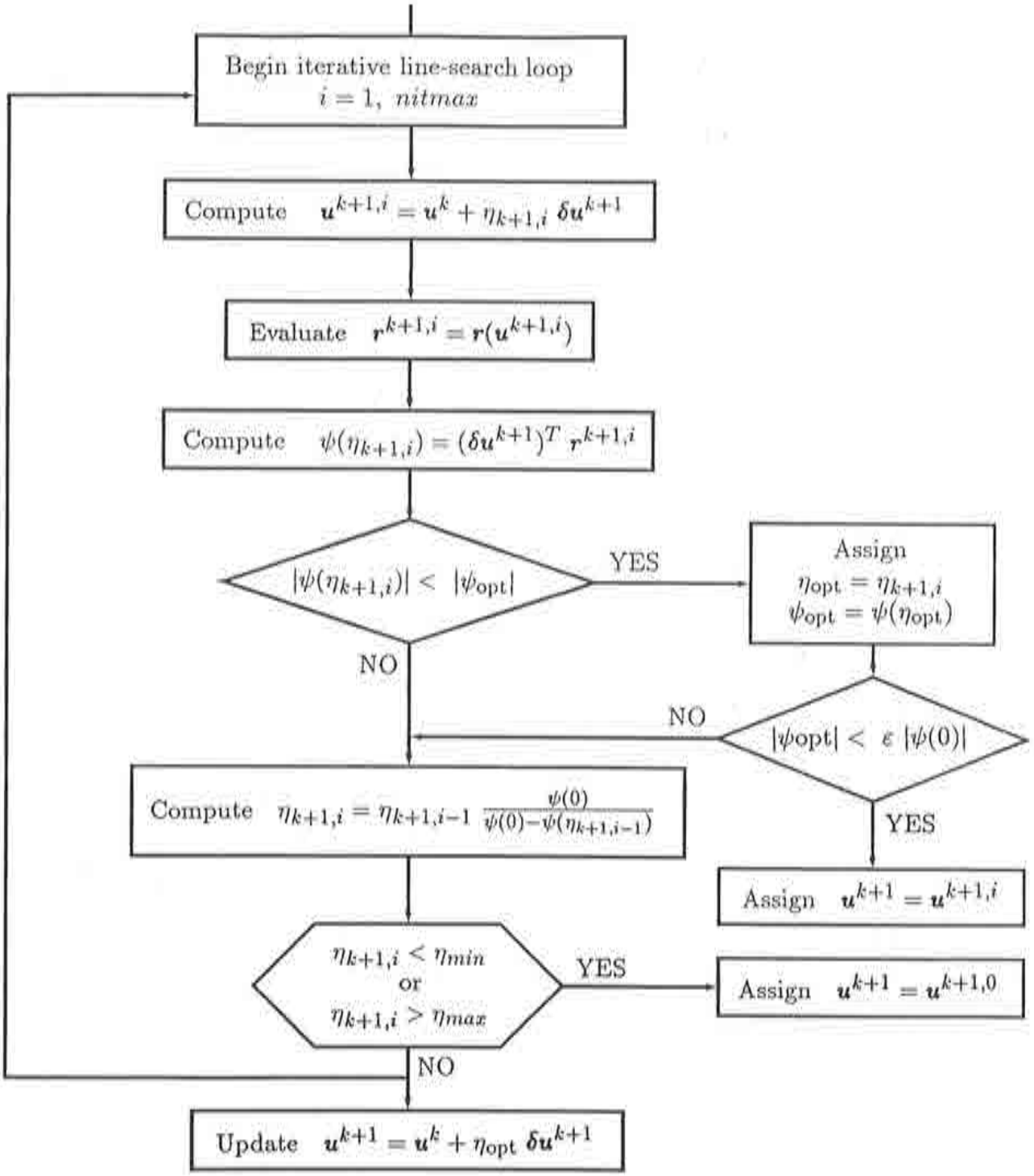


Figure 7.2: Flowchart for the line-search procedure (continued).

7.2 Performing line searches with Castem

When using Lagrange multipliers to treat boundary conditions, augmented vectors $\boldsymbol{U} = (\boldsymbol{u}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ are obtained. This means that two different line-search techniques are possible:

Option T (Total approach)

Update the solution following the expression

$$\boldsymbol{U}^{k+1} = \boldsymbol{U}^{k} + \eta_{k+1}\, \boldsymbol{\delta U}^{k+1}, \tag{7.11}$$

where the advance length η_{k+1} affects the whole augmented vector $\boldsymbol{\delta U}^{k+1}$. For this option, η_{k+1} is an approximation to the minimiser of the function

$$\psi(\eta_{k+1}) = (\boldsymbol{\delta U}^{k+1})^{T}\, \boldsymbol{R}(\boldsymbol{U}^{k} + \eta_{k+1}\, \boldsymbol{\delta U}^{k+1}). \tag{7.12}$$

Option P (Partial approach)

Compute the new approximation $\boldsymbol{U}^{k+1}$ performing a line search update only on the displacements $\boldsymbol{u}$

$$\boldsymbol{U}^{k+1} = (\boldsymbol{u}^{k} + \eta_{k+1}\, \boldsymbol{\delta u}^{k+1}, \boldsymbol{\lambda}^{k} + \boldsymbol{\delta\lambda}^{k+1}, \boldsymbol{\mu}^{k} + \boldsymbol{\delta\mu}^{k+1}), \tag{7.13}$$

where η_{k+1} is a close value to the stationary solution of

$$\psi(\eta_{k+1}) = (\boldsymbol{\delta u}^{k+1})^{T}\, \boldsymbol{R}_1(\boldsymbol{U}^{k} + \eta_{k+1}\, \boldsymbol{\delta U}^{k+1}). \tag{7.14}$$

To compare the two options, we first want to remark that the vector of residual forces in (7.12), $\boldsymbol{R}(\boldsymbol{U}^{k} + \eta_{k+1}\, \boldsymbol{\delta U}^{k+1})$, is null except for the first block of components $\boldsymbol{R}_1$. Indeed, thanks to the fact that constraints are linear on $\boldsymbol{u}$ and using equation (3.29),

$$\boldsymbol{R}(\boldsymbol{U}^k + \eta_{k+1}\,\boldsymbol{\delta U}^{k+1}) = \begin{pmatrix} \boldsymbol{R}_1(\boldsymbol{U}^k + \eta_{k+1}\,\boldsymbol{\delta U}^{k+1}) \\ \boldsymbol{A}_e\,(\boldsymbol{u}^k + \eta_{k+1}\,\boldsymbol{\delta u}^{k+1}) - {}^{n+1}\boldsymbol{b} \\ \boldsymbol{A}_e\,(\boldsymbol{u}^k + \eta_{k+1}\,\boldsymbol{\delta u}^{k+1}) - {}^{n+1}\boldsymbol{b} \end{pmatrix} =$$

$$= \begin{pmatrix} \boldsymbol{R}_1(\boldsymbol{U}^k + \eta_{k+1}\,\boldsymbol{\delta U}^{k+1}) \\ \boldsymbol{0} + \eta_{k+1}\,\boldsymbol{0} = \boldsymbol{0} \\ \boldsymbol{0} + \eta_{k+1}\,\boldsymbol{0} = \boldsymbol{0} \end{pmatrix}.$$

Therefore,

$$(\boldsymbol{\delta U}^{k+1})^T\ \boldsymbol{R}(\boldsymbol{U}^k + \eta_{k+1}\,\boldsymbol{\delta U}^{k+1}) =$$

$$= (\boldsymbol{\delta u}^{k+1} \quad \boldsymbol{\delta \lambda}^{k+1} \quad \boldsymbol{\delta \mu}^{k+1}) \begin{pmatrix} \boldsymbol{R}_1(\boldsymbol{U}^k + \eta_{k+1}\,\boldsymbol{\delta U}^{k+1}) \\ \boldsymbol{0} - \eta_{k+1}\,\boldsymbol{0} \\ \boldsymbol{0} - \eta_{k+1}\,\boldsymbol{0} \end{pmatrix} =$$

$$= (\boldsymbol{\delta u}^{k+1})^T\ \boldsymbol{R}_1(\boldsymbol{U}^k + \eta_{k+1}\,\boldsymbol{\delta U}^{k+1}),$$

which means that equations (7.12) and (7.14) define exactly *the same function* ψ.

Hence, the alternative updates (7.11) and (7.13) use the same value of the step-length η_{k+1}. These two updates have been tested on several examples; the following conclusions have been drawn from the obtained results:

- The two versions show a similar behaviour
- but the Total version introduces a distortion of the Lagrange multipliers which delays convergence.
- The Partial version consumes less CPU time.

In view of these results, we decide to use the Partial version (7.13)-(7.14) of the line search.

7.3 Numerical examples

In this section, two numerical examples are shown where the line search procedure detailed in Figure 7.2 has been employed.

Example TRUSS

The initial stress method with line search acceleration is also used to analyse the truss problem of Figure 4.2. The same method without acceleration is employed as a reference.

Figures 7.3, 7.4, 7.5 and 7.6 show the behaviour of the two strategies for various steps. For this very simple test, a reduced number of iterations per increment is required with the accelerated algorithm. The associated computational cost can be found in Table 7.1.

Method	Iterations	CPU time
k0	200	4 235
k0_ls	32	1 113

Table 7.1: Total costs of the initial stress method with and without line search for the truss test.

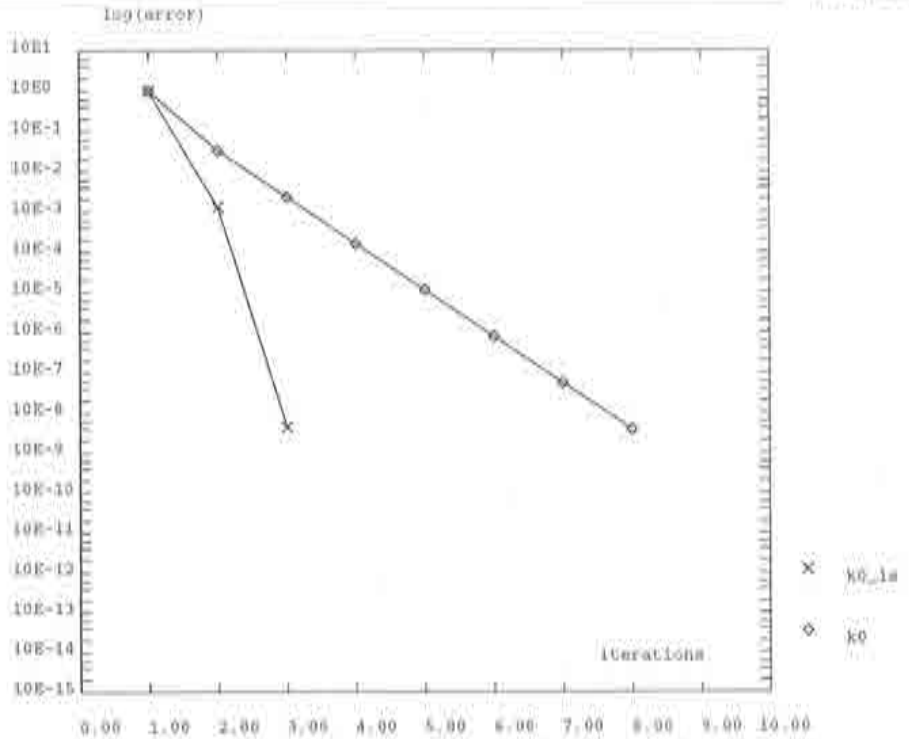


Figure 7.3: Truss test, load step 1. Influence of line-search on the initial stress method.

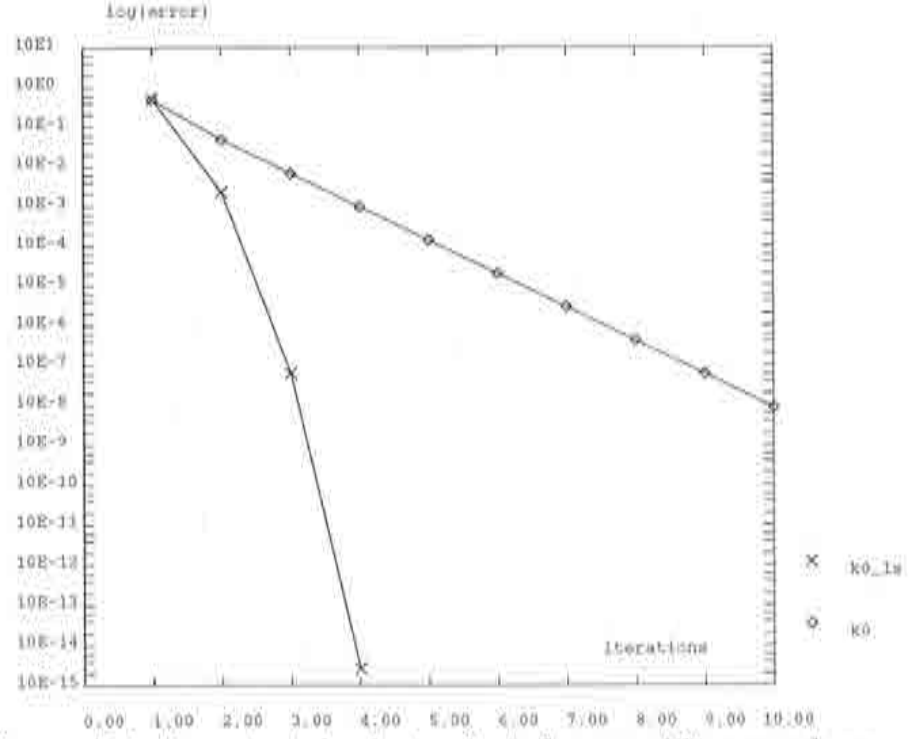


Figure 7.4: Truss test, load step 2. Influence of line-search on the initial stress method.

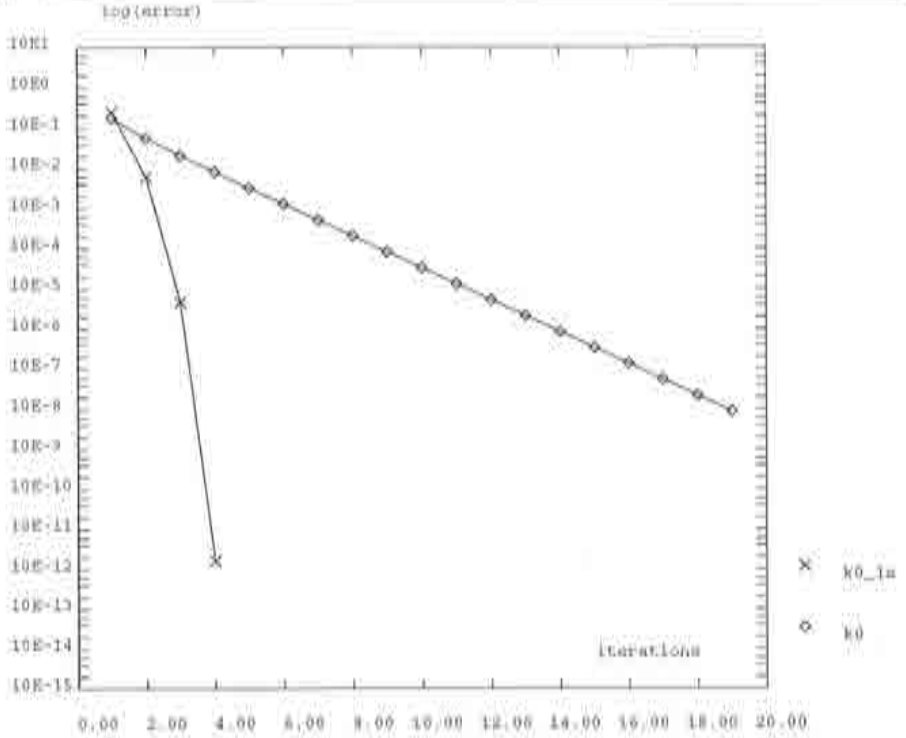


Figure 7.5: Truss test, load step 5. Influence of line-search on the initial stress method.

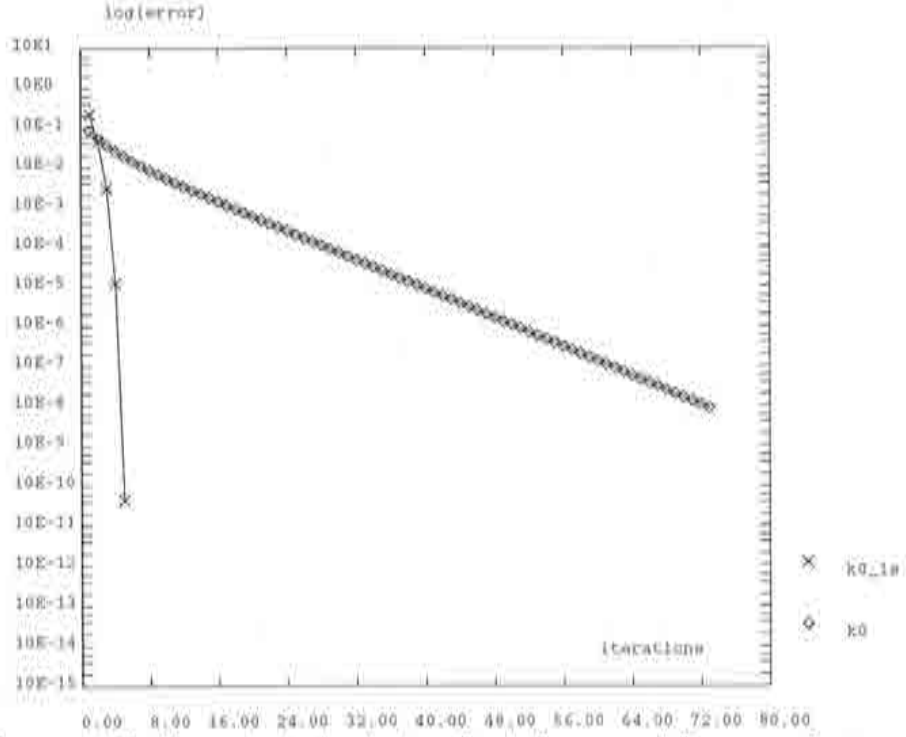


Figure 7.6: Truss test, load step 8. Influence of line-search on the initial stress method.

Example CYLINDER

A similar comparison is carried out with the cylinder test of Figure 4.7. The influence of the line search can be assessed in Figures 7.7 and 7.8, corresponding to steps 4 and 6, and in Table 7.2. It can be seen that, in spite of an oscillatory behaviour, the acceleration manages an important reduction in the computational cost.

Method	Iterations	CPU time
k0	788	19 152
k0_ls	282	9 331

Table 7.2: Total costs of the initial stress method with and without line search for the cylinder test

Remarks: The line-search acceleration is especially effective with low-order methods such as the initial stress method.

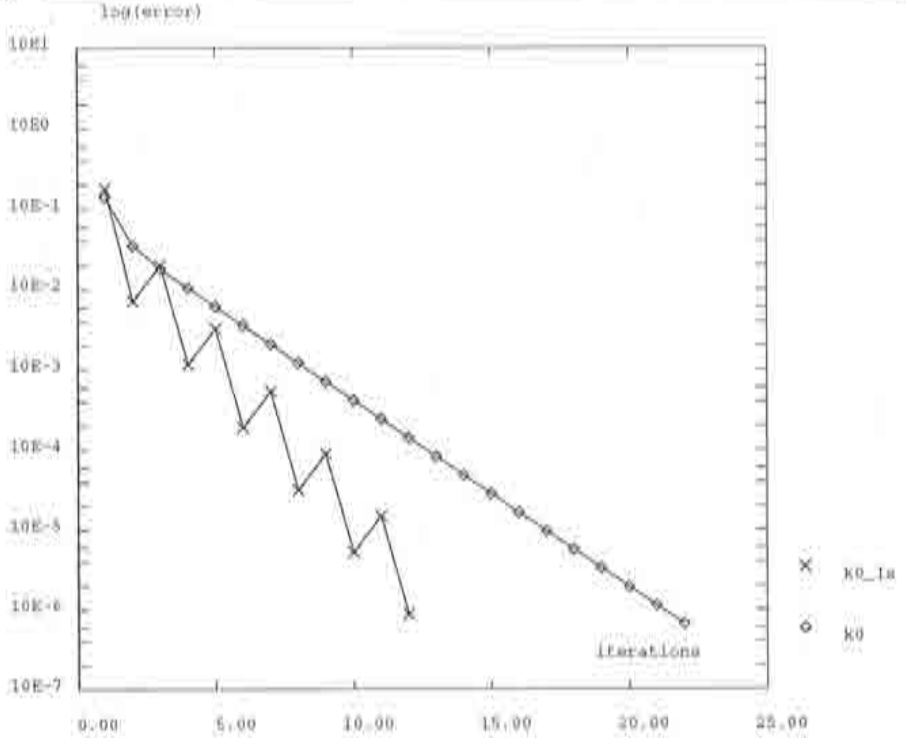


Figure 7.7: Cylinder test, load step 4. Influence of line-search on the initial stress method.

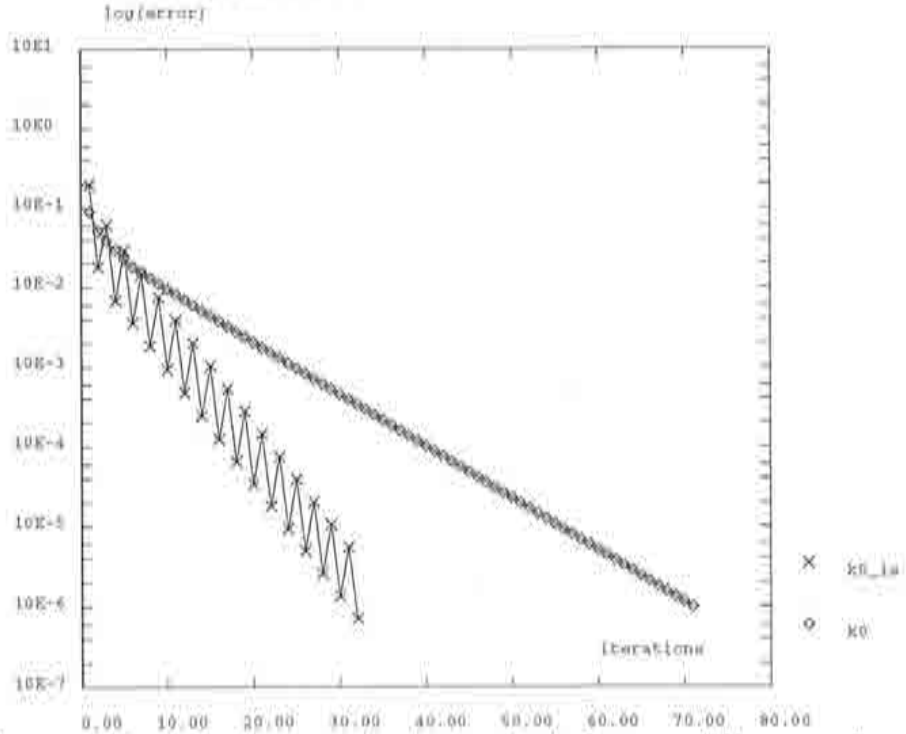


Figure 7.8: Cylinder test, load step 6. Influence of line-search on the initial stress method.

Chapter 8

Arc-length methods

8.1 Some words about load control and the need for continuation techniques

In Chapter 3, the incremental/iterative solution for nonlinear problems was introduced. As we have already seen, this procedure consists of splitting up the total load $\boldsymbol{f}_e$ into N *increments of load* ${}^n\Delta\boldsymbol{f}$, so that

$$\boldsymbol{f}_e = {}^1\Delta\boldsymbol{f} + {}^2\Delta\boldsymbol{f} + \ldots + {}^{N-1}\Delta\boldsymbol{f} + {}^N\Delta\boldsymbol{f} = {}^N\boldsymbol{f}. \tag{8.1}$$

Once this fragmentation is performed, we only need to focus on each load step. Indeed, all the discussed methods deal with the calculation of ${}^{n+1}\boldsymbol{u}$ for which $\boldsymbol{f}_i({}^{n+1}\boldsymbol{u}) - {}^{n+1}\boldsymbol{f} = \boldsymbol{0}$, given that ${}^n\boldsymbol{u}$ is already known from the previous increment. This is also known as a *load control* procedure.

In this chapter we start by presenting some cases in which the decomposition of the load described in equation (8.1) may not provide a good solution to the problem. Later on, some alternative techniques will be presented.

Figure 8.1 shows a typical "brittle collapse" load-displacement curve: both load and displacement increase until a *limit point* A is achieved; beyond this point,

displacement goes on growing but load starts to decrease. When using a load control procedure, we are going to obtain solutions for all loads below the limit load, but the algorithm will fail to find a solution when point A is passed by.

If we are making an analysis of the behaviour of the solid before collapse, we may come to the conclusion that this divergence has occurred precisely because a limit point has been passed by. However, this is a dangerous interpretation to make, since a failure of the algorithm may be also due to a bad choice of the increment of load, etc.

On the contrary, we may want to follow the equilibrium path of the structure; for instance, because it is integrated in some bigger structure, or because we believe that point A is just a local maximum and not the ultimate load, or even because we want to make sure that we have gone past a limit point. In this case, a load control procedure will be of no use.

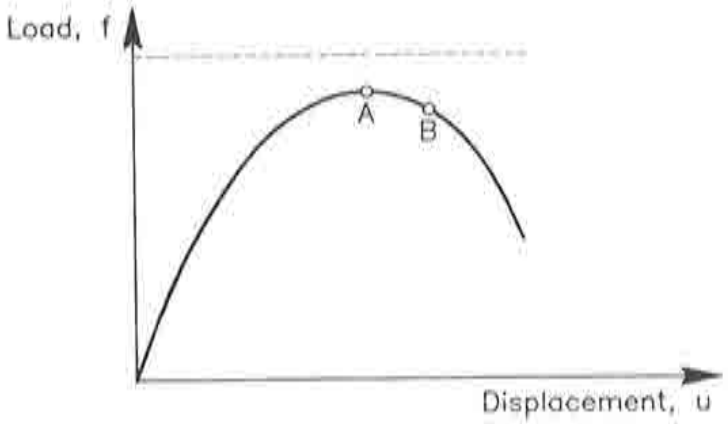


Figure 8.1: Brittle collapse.

The curve in Figure 8.2 describes a *ductile collapse*. This behaviour is identical to the one represented in Figure 8.1, except for the fact that the load stays constant instead of lessening past point A. As in Figure 8.1, if increments of load are applied, the load-displacement curve can be followed up until point A, but no further.

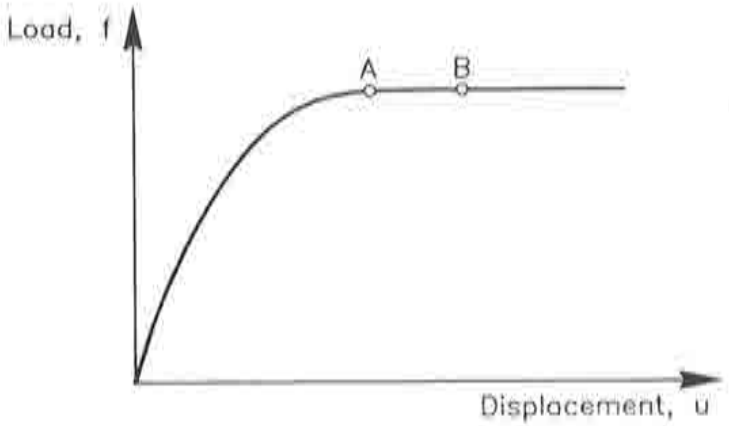


Figure 8.2: Ductile collapse.

Let us consider now the curve in Figure 8.3. In this case, A is a local maximum. If a load control is performed by imposing some small increments of load, we will be able to get close enough to point A. The next increment, if converged, is going to yield point C, passing over everything that happens under the straight line AC. Figure 8.3 describes what is known as a *snap-through* response.

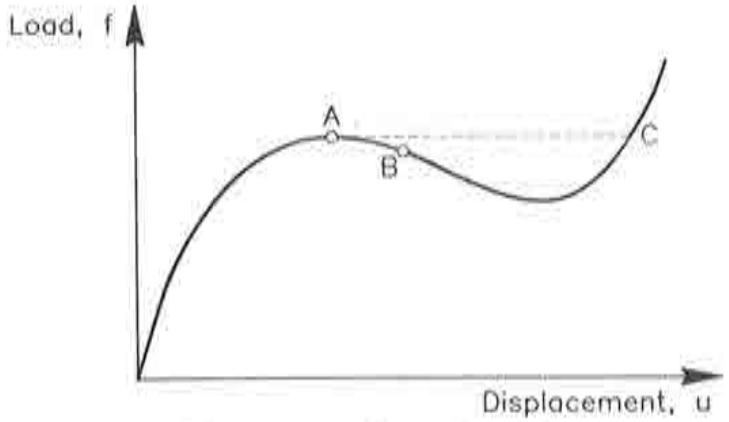


Figure 8.3: Snap-through.

To obtain good solutions to the problems illustrated in Figures 8.1 to 8.3, we could use an alternative technique, which consists of applying increments of displacement rather than of load. This *displacement control* procedure would allow us to follow the path AB and beyond, since in this case we would use increasing values of the displacement axis.

In the case of Figure 8.4, however, neither of the two procedures, load or displacement control, is good enough. A load control would give the same problems as in Figure 8.3, while a displacement control would force loads to jump from A to C, ignoring the *snap-back* behaviour around point B.

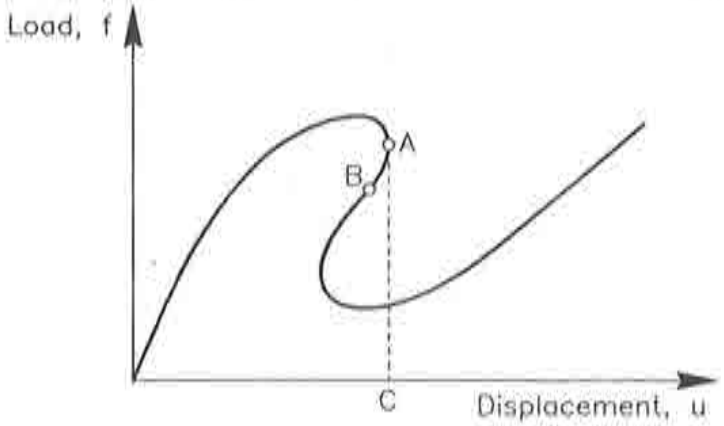


Figure 8.4: Snap-back.

For cases such as the one in Figure 8.4, we will need to use the so-called *continuation methods*, which are discussed in the following section. These methods allow us to obtain points on the load-deflection curve by prescribing the *distance* from one point to the next rather than by requiring increasing values of loads or displacements.

8.2 The arc-length method. A general formulation including various control strategies

The arc-length method

Arc-length methods were firstly introduced by Riks, [22], and Wempner, [29], and have been thoroughly studied later on by Crisfield, [8,9,10], and others, [6,16,24]. As indicated in the previous section, these methods consist of imposing fixed *arc lenghts* between points on the load-displacement curve, so that any structural behaviour can be captured.

We begin by defining the total load associated to the current increment as

$$^{n+1}\boldsymbol{f} = {}^{n}\boldsymbol{f} + {}^{n+1}\boldsymbol{\Delta f}, \tag{8.2}$$

We are now going to write the increment of load $^{n+1}\boldsymbol{\Delta f}$ as

$$^{n+1}\boldsymbol{\Delta f} = {}^{n+1}\Delta\alpha\ \boldsymbol{f}_e, \tag{8.3}$$

where $\boldsymbol{f}_e$ denotes a fixed referential load and α is the so-called *load-level parameter*. If $^{n+1}\Delta\alpha$ is fixed a priori, then equation (8.2) describes a load control procedure. For the arc length method, however, the value of $^{n+1}\Delta\alpha$ is *unknown* at the beginning of the increment.

Let $^{n+1}\boldsymbol{\Delta u}$ be, as usual, the solution associated to the increment of load $^{n+1}\boldsymbol{\Delta f}$. Then, the expression

$$\sqrt{(^{n+1}\boldsymbol{\Delta u})^T\ {}^{n+1}\boldsymbol{\Delta u} + (^{n+1}\boldsymbol{\Delta f})^T\ {}^{n+1}\boldsymbol{\Delta f}} \tag{8.4}$$

gives the *distance* between points $(^{n}\boldsymbol{u}, {}^{n}\boldsymbol{f})$ and $(^{n+1}\boldsymbol{u}, {}^{n+1}\boldsymbol{f})$ in the load-displacement space. Recalling from (8.3) that $^{n+1}\boldsymbol{\Delta f} = {}^{n+1}\Delta\alpha\ \boldsymbol{f}_e$ and introducing a scale parameter c to weigh the contribution of load and displacement terms, we go on to require for this distance to equal a prescribed *arc length* Δl

$$\sqrt{(^{n+1}\boldsymbol{\Delta u})^T\ {}^{n+1}\boldsymbol{\Delta u} + c\ (^{n+1}\Delta\alpha)^2\ (\boldsymbol{f}_e)^T\ \boldsymbol{f}_e} = \Delta l. \tag{8.5}$$

The value $c = 0$ leads to the so-called *cylindrical* arc-length methods; $c = 1$ yields the *spherical* methods.

The *arc-length condition* (8.5) involves $n + 1$ unknowns (n for $^{n+1}\boldsymbol{\Delta u}$ and one more for the scalar $^{n+1}\Delta\alpha$), which need to be computed from the n equilibrium equations

$$\boldsymbol{f}_i(^{n+1}\boldsymbol{u}) - {}^{n+1}\boldsymbol{f} = \boldsymbol{f}_i(^{n}\boldsymbol{u} + {}^{n+1}\boldsymbol{\Delta u}) - (^{n}\boldsymbol{f} + {}^{n+1}\Delta\alpha\ \boldsymbol{f}_e) = \boldsymbol{0} \tag{8.6}$$

together with the additional condition (8.5).

Riks, [22], and Wempner, [29], suggest to obtain those $n + 1$ variables by using an extended Taylor development. So, we begin by computing a *prediction* $^{n+1}\Delta\alpha^1$

to ${}^{n+1}\Delta\alpha$ and also a prediction ${}^{n+1}\Delta\boldsymbol{u}^1$ to ${}^{n+1}\Delta\boldsymbol{u}$. For this purpose, we need to solve the Newton set of n linear equations

$$ {}^{n+1}\boldsymbol{K}^0\ {}^{n+1}\Delta\boldsymbol{u}^1 = {}^{n+1}\Delta\boldsymbol{f}^1 = {}^{n+1}\Delta\alpha^1\ \boldsymbol{f}_e, \tag{8.7} $$

where we have now a total of $n+1$ unknowns. If we define a new vector ${}^{n+1}\Delta\boldsymbol{u}_T$ through the expression

$$ {}^{n+1}\Delta\boldsymbol{u}^1 = {}^{n+1}\Delta\alpha^1\ {}^{n+1}\Delta\boldsymbol{u}_T, \tag{8.8} $$

then equation (8.7) can be rewritten in terms of ${}^{n+1}\Delta\boldsymbol{u}_T$ as

$$ {}^{n+1}\boldsymbol{K}^0\ {}^{n+1}\Delta\boldsymbol{u}_T = \boldsymbol{f}_e, \tag{8.9} $$

where the T subindex stands for *total* load. The arc-length condition (8.5) can be now applied to the prediction ${}^{n+1}\Delta\boldsymbol{u}^1$ yielding

$$ \sqrt{({}^{n+1}\Delta\boldsymbol{u}^1)^T\ {}^{n+1}\Delta\boldsymbol{u}^1 + c\ ({}^{n+1}\Delta\alpha^1)^2\ (\boldsymbol{f}_e)^T\ \boldsymbol{f}_e} = \Delta l, $$

or also, using ${}^{n+1}\Delta\boldsymbol{u}_T$ rather than ${}^{n+1}\Delta\boldsymbol{u}^1$ and avoiding square roots,

$$ ({}^{n+1}\Delta\alpha^1)^2\ \left[({}^{n+1}\Delta\boldsymbol{u}_T)^T\ {}^{n+1}\Delta\boldsymbol{u}_T + c\ (\boldsymbol{f}_e)^T\ \boldsymbol{f}_e\right] = (\Delta l)^2. \tag{8.10} $$

The sequence of computations will consist of:

1. Compute ${}^{n+1}\Delta\boldsymbol{u}_T$ by solving (8.9).

2. Obtain the value of ${}^{n+1}\Delta\alpha^1$ from (8.10) as

$$ {}^{n+1}\Delta\alpha^1 = \pm\frac{\Delta l}{\sqrt{({}^{n+1}\Delta\boldsymbol{u}_T)^T\ {}^{n+1}\Delta\boldsymbol{u}_T + c\ (\boldsymbol{f}_e)^T\ \boldsymbol{f}_e}}. \tag{8.11} $$

The choice of the sign in (8.11) will be discussed later on.

After ${}^{n+1}\Delta\boldsymbol{u}_T$ and ${}^{n+1}\Delta\alpha^1$, a prediction ${}^{n+1}\Delta\boldsymbol{u}^1$ to the displacements can be obtained from (8.8).

We can now update displacements

$$^{n+1}\boldsymbol{u}^1 = {}^{n}\boldsymbol{u} + {}^{n+1}\Delta\boldsymbol{u}^1 \tag{8.12}$$

and also external forces, computing first the new value of the load level,

$$^{n+1}\alpha^1 = {}^{n}\alpha + {}^{n+1}\Delta\alpha^1 \tag{8.13}$$

where ${}^{n}\alpha$ is such that ${}^{n}\boldsymbol{f} = {}^{n}\alpha\ \boldsymbol{f}_e$. After ${}^{n+1}\alpha^1$ is obtained from (8.13), the updated load ${}^{n+1}\boldsymbol{f}^1$ is obtained as

$$^{n+1}\boldsymbol{f}^1 = {}^{n+1}\alpha^1\ \boldsymbol{f}_e. \tag{8.14}$$

In general, $({}^{n+1}\boldsymbol{u}^1, {}^{n+1}\boldsymbol{f}^1)$ will not be an equilibrium point. Therefore, we will need to iterate on $\boldsymbol{u}$, and also on α since ${}^{n+1}\Delta\alpha$ is not prescribed a priori, until convergence is achieved.

In a general iteration, we look for a displacement correction vector ${}^{n+1}\boldsymbol{\delta u}^{k+1}$ such that

$$\boldsymbol{r}({}^{n+1}\boldsymbol{u}^{k+1}) = \boldsymbol{r}({}^{n+1}\boldsymbol{u}^{k} + {}^{n+1}\boldsymbol{\delta u}^{k+1}) = \boldsymbol{0}. \tag{8.15}$$

Under the arc-length formulation, and dropping $n+1$ left superscripts, (8.15) yields

$$\boldsymbol{K}(\boldsymbol{u}^k)\ \boldsymbol{\delta u}^{k+1} = -\boldsymbol{r}(\boldsymbol{u}^k) + \delta\alpha^{k+1}\ \boldsymbol{f}_e, \tag{8.16}$$

where $\boldsymbol{K}^k$ can be any iteration matrix depending on the method that is used and the term $\delta\alpha^{k+1}\ \boldsymbol{f}_e$ accounts for the variation of external load due to the iterative correction of the load level α.

Equation (8.16) can be inverted symbollically to give the following expression for $\boldsymbol{\delta u}^{k+1}$

$$\boldsymbol{\delta u}^{k+1} = (\boldsymbol{K}(\boldsymbol{u}^k))^{-1}\ (-\boldsymbol{r}(\boldsymbol{u}^k)) + \delta\alpha^{k+1}\ (\boldsymbol{K}(\boldsymbol{u}^k))^{-1}\ \boldsymbol{f}_e. \tag{8.17}$$

If two vectors $\boldsymbol{\delta\bar{u}}^{k+1}$ and $\boldsymbol{\delta u}_T^{k+1}$ are now defined as

$$\boldsymbol{\delta\bar{u}}^{k+1} = (\boldsymbol{K}(\boldsymbol{u}^k))^{-1}\ (-\boldsymbol{r}(\boldsymbol{u}^k)) \tag{8.18}$$

$$\boldsymbol{\delta u}_T^{k+1} = (\boldsymbol{K}(\boldsymbol{u}^k))^{-1}\ \boldsymbol{f}_e, \tag{8.19}$$

then equation (8.17) can be rewritten into

$$\boldsymbol{\delta u}^{k+1} = \boldsymbol{\delta \tilde{u}}^{k+1} + \delta\alpha^{k+1}\, \boldsymbol{\delta u}_T^{k+1}. \tag{8.20}$$

The arc-length condition must then be imposed. We are going to demand for points $(\boldsymbol{u}^0, \boldsymbol{f}^0)$, $(\boldsymbol{u}^{k+1}, \boldsymbol{f}^{k+1})$ to keep *the same distance* Δl all through the iterations. Consequently, the arc-length restriction will be expressed as

$$(\boldsymbol{\Delta u}^{k+1})^T\, \boldsymbol{\Delta u}^{k+1} + c\, (\Delta\alpha^{k+1})^2\, (\boldsymbol{f}_e)^T\, \boldsymbol{f}_e = (\Delta l)^2. \tag{8.21}$$

Recalling that $\boldsymbol{\Delta u}^{k+1} = \boldsymbol{\Delta u}^k + \boldsymbol{\delta u}^{k+1}$, where the arc-length condition is verified for $\boldsymbol{\Delta u}^k$, and using (8.20), equation (8.21) can be manipulated to yield the following quadratic equation on $\delta\alpha^{k+1}$, [8,9,10],

$$a_1\, (\delta\alpha^{k+1})^2 + a_2\, \delta\alpha^{k+1} + a_3 = 0, \tag{8.22}$$

with a_1, a_2, a_3 being the three scalar values given by

$$a_1 = (\boldsymbol{\delta u}_T^{k+1})^T\, \boldsymbol{\delta u}_T^{k+1} + c\, (\boldsymbol{f}_e)^T \boldsymbol{f}_e \tag{8.23}$$

$$a_2 = 2\, (\boldsymbol{\Delta u}^k + \boldsymbol{\delta \tilde{u}}^{k+1})^T\, \boldsymbol{\delta u}_T^{k+1} + \Delta\alpha^k\, c\, (\boldsymbol{f}_e)^T \boldsymbol{f}_e \tag{8.24}$$

$$a_3 = 2\, (\boldsymbol{\Delta u}^k)^T\, \boldsymbol{\delta \tilde{u}}^{k+1} + (\boldsymbol{\delta \tilde{u}}^{k+1})^T\, \boldsymbol{\delta \tilde{u}}^{k+1} \tag{8.25}$$

The sequence of computations can be arranged as follows:

1. Compute $\boldsymbol{\delta \tilde{u}}^{k+1}$ by solving

$$\boldsymbol{K}(\boldsymbol{u}^k)\, \boldsymbol{\delta \tilde{u}}^{k+1} = -\boldsymbol{r}(\boldsymbol{u}^k) \tag{8.26}$$

(equivalent to equation (8.18)).

2. Compute $\boldsymbol{\delta u}_T^{k+1}$ by solving

$$\boldsymbol{K}(\boldsymbol{u}^k)\, \boldsymbol{\delta u}_T^{k+1} = \boldsymbol{f}_e \tag{8.27}$$

(equivalent to equation (8.19)).

3. Compute the values of a_1, a_2, a_3 from (8.23), (8.24) and (8.25).

4. Solve equation (8.22). Here, two cases are possible:

If no real values are obtained, that is, if equation (8.22) has two conjugate complex roots, the process will stop*.

If two real values are obtained, one of them* will be used to compute $\boldsymbol{\delta u}^{k+1}$ from (8.20).

Finally, we can update the load-level parameter, and also displacements and forces, as

$$\alpha^{k+1} = \alpha^k + \delta\alpha^{k+1}$$

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \boldsymbol{\delta u}^{k+1}$$

$$\boldsymbol{f}^{k+1} = \boldsymbol{f}^k + \delta\alpha^{k+1}\, \boldsymbol{f}_e = \alpha^{k+1}\, \boldsymbol{f}_e.$$

Additional considerations on the arc-length procedure

1. Choosing the sign of the prediction ${}^{n+1}\Delta\alpha^1$

As indicated in [13], the natural choice for the sign of ${}^{n+1}\Delta\alpha^1$ is

$$\text{sign}\,({}^{n+1}\Delta\alpha^1) = \text{sign}\Big(({}^n\boldsymbol{\Delta u})^T\ {}^{n+1}\boldsymbol{\Delta u}_T\Big).$$

Writing this scalar product as

$$({}^n\boldsymbol{\Delta u})^T\ {}^{n+1}\boldsymbol{\Delta u}_T = \|{}^n\boldsymbol{\Delta u}\|\ \|{}^{n+1}\boldsymbol{\Delta u}_T\|\ \cos({}^n\boldsymbol{\Delta u}, {}^{n+1}\boldsymbol{\Delta u}_T)$$

we have that

* Further considerations on this situation will be made later on.

- An acute angle between the previous increment of displacement ${}^{n}\boldsymbol{\Delta u}$ and ${}^{n+1}\boldsymbol{\Delta u}_T$ will force us to increase the load level

- whereas an obtuse angle $({}^{n}\boldsymbol{\Delta u}, {}^{n+1}\boldsymbol{\Delta u}_T)$ will yield a negative value of ${}^{n+1}\Delta\alpha^1$ and thus a decrease of the load level.

2. Choosing the appropriate root ${}^{n+1}\delta\alpha^{k+1}$

Let us assume that the quadratic equation

$$a_1\ ({}^{n+1}\delta\alpha^{k+1})^2 + a_2\ {}^{n+1}\delta\alpha^{k+1} + a_3 = 0$$

has two real roots ${}^{n+1}\delta\alpha_1^{k+1}$ and ${}^{n+1}\delta\alpha_2^{k+1}$. We will denote the associated correction vectors respectively as ${}^{n+1}\boldsymbol{\delta u}_1^{k+1}$ and ${}^{n+1}\boldsymbol{\delta u}_2^{k+1}$.

To choose between these two roots, and since we want to continue to move in the same direction as in the previous iteration, we demand for the angle defined by vectors ${}^{n+1}\boldsymbol{\Delta u}^k$ and ${}^{n+1}\boldsymbol{\Delta u}^{k+1}$ to be acute. This means that we need to build the two vectors

$${}^{n+1}\boldsymbol{\Delta u}_i^{k+1} = {}^{n+1}\boldsymbol{\Delta u}^k + {}^{n+1}\boldsymbol{\delta u}_i^{k+1}, \quad i = 1, 2.$$

We will take the correction ${}^{n+1}\delta\alpha_i^{k+1}$ that gives a positive value of the scalar product

$$({}^{n+1}\boldsymbol{\Delta u}^k)^T\ {}^{n+1}\boldsymbol{\Delta u}_i^{k+1}.$$

If both scalar products have the same sign, we will choose, [6], the load-level correction ${}^{n+1}\delta\alpha_i^{k+1}$ that resembles the most to the solution $\delta\alpha_{lin}^{k+1} = -a_3/a_2$ to the linearised version of the quadratic equation, $a_2\ {}^{n+1}\delta\alpha^{k+1} + a_3 = 0$.

3. What to do when complex roots are obtained

When complex values of the roots are obtained, the algorithm will fail. Some authors, however, detect this situation and force the arc length Δl to decrease;

in this way, an additional computational effort is required as we need to go back to the previous iteration, but in return the algorithm does not stop.

Nevertheless, this situation (presence of complex roots) is not likely to happen if an automatic update of the arc length is performed.

4. Automatic update of the arc length

Since an appropriate value of the arc length Δl is unknown a priori, it is not advisable in general to choose a random value for it and keep this value all the way through to the end of the problem. Besides, this *appropriate* value may differ from one step to the next.

Crisfield, [10], suggests a recomputation of Δl at the beginning of each increment following the expression

$$\Delta l_{new} = \sqrt{\frac{N_{opt}}{N_{old}}}\, \Delta l_{old}, \tag{8.28}$$

where Δl_{old} is the arc length that we have used in the previous increment, N_{old} is the total number of iterations that have been needed before convergence and N_{opt} is the desired number of iterations. In this way, if $N_{old} < N_{opt}$, the arc length will be increased. If, on the contrary, the previous increment has required a number of iterations $N_{old} > N_{opt}$, the arc length will be made smaller.

Alternative updates have been suggested. In [2], the expression

$$\Delta l_{new} = \sqrt[4]{\frac{N_{old}}{N_{opt}}}\, \Delta l_{old} \tag{8.29}$$

is employed and compared to Crisfield's update, equation (8.28).

A general formulation of the arc-length method that includes load and displacement control

Let us recall the arc-length condition in its original form

$$(\Delta \boldsymbol{u})^T \, \Delta \boldsymbol{u} + \, c \, (\Delta \alpha)^2 \, (\boldsymbol{f}_e)^T \, \boldsymbol{f}_e = (\Delta l)^2. \tag{8.30}$$

As indicated in [13], equation (8.30) can be generalised by introducing an n-dimensional diagonal matrix $\boldsymbol{P}$ as follows

$$(\Delta \boldsymbol{u})^T \, \boldsymbol{P} \, \Delta \boldsymbol{u} + \, c \, (\Delta \alpha)^2 \, (\boldsymbol{f}_e)^T \, \boldsymbol{f}_e = (\Delta l)^2. \tag{8.31}$$

In this way, most solution strategies can be obtained by defining an adequate $\boldsymbol{P}$ matrix in (8.31).

- The *original Crisfield–Ramm arc-length method* is recovered using $\boldsymbol{P} = \boldsymbol{I}$; indeed, this choice of matrix $\boldsymbol{P}$ turns us back to equation (8.30).

- To perform a *load control*, we only need to set $\boldsymbol{P}$ to a zero matrix and assign $c = 1$.

- A *displacement control* on the m-th degree of freedom $\boldsymbol{u}_m$ will be associated to the following definition of $\boldsymbol{P}$:

$$\boldsymbol{P}_{ii} = \delta_{im}, \quad i = 1, \ldots, n$$

In this case, we also need to set c to zero.

- Finally, a *general arc-length method* can be produced that takes into account only some specified components of $\boldsymbol{u}$; this is accomplished by setting to zero the rest of the associated diagonal terms of matrix $\boldsymbol{P}$.

8.3 Flowcharts for procedures ANONLIN and AINCREME. Programming continuation methods in Castem

Procedures ANONLIN and AINCREME

To introduce arc-length techniques in Castem, two *procedures*, ANONLIN and AINCREME, have been developed. These two *procedures* play equivalent roles to NONLIN and INCREME, see Section 3.3, for the load control strategy.

The main differences between the original and the "A" (Arc-length) versions is that the total external load, which is simply updated in NONLIN, needs to be actually *computed* in AINCREME, together with displacement vectors, for the arc-length version.

Figures 8.5 and 8.6 give the flowcharts associated to procedures ANONLIN and AINCREME. It is interesting to remark that now no list of load steps is given as an input to ANONLIN. This means that a control on the load level must be performed; for instance, if $\boldsymbol{f}_e$ is taken as total maximum load, the condition

$$^{n+1}\alpha > 1$$

will force the overall computations to stop.

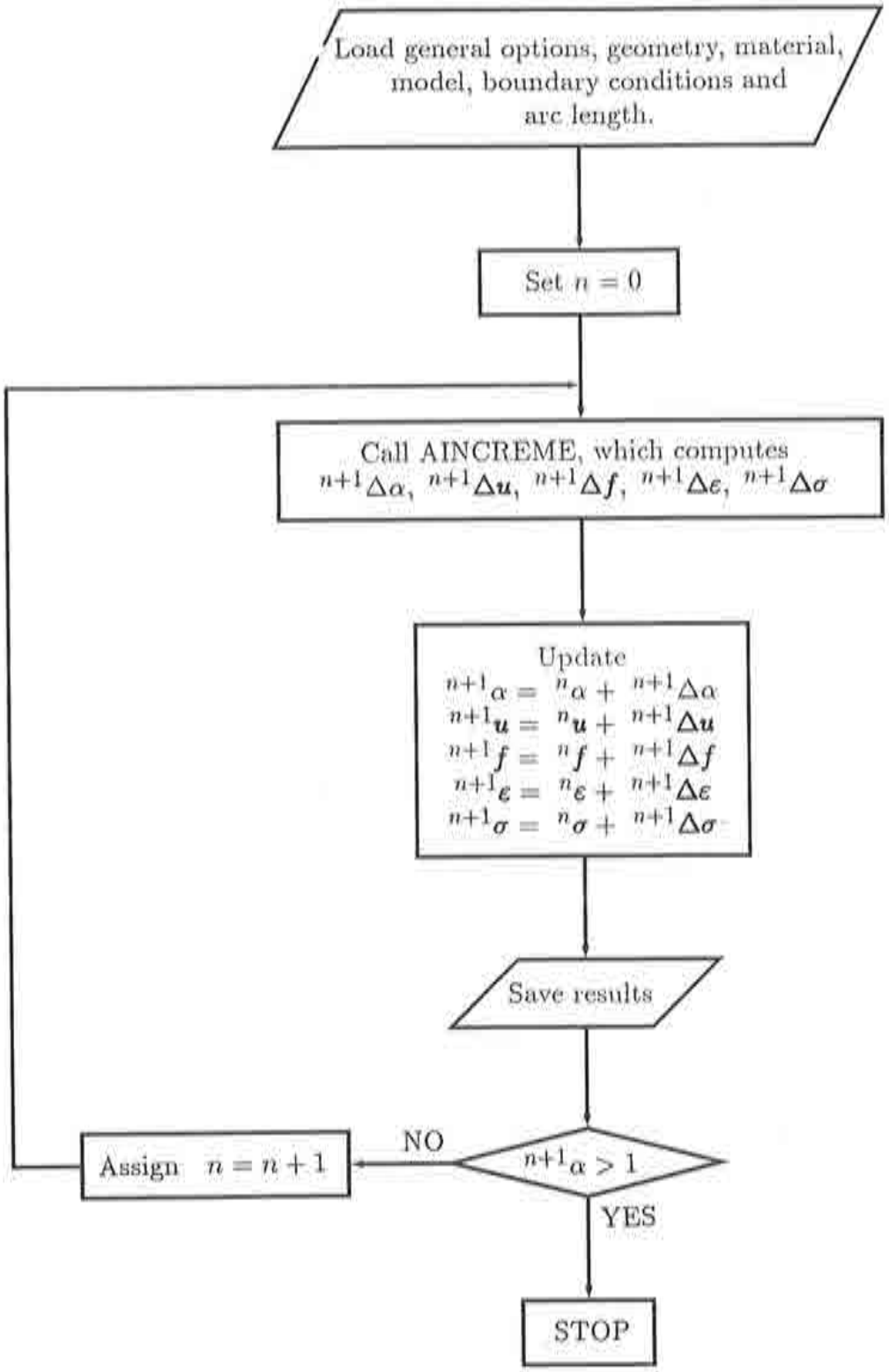


Figure 8.5: Flowchart for *procedure* ANONLIN.

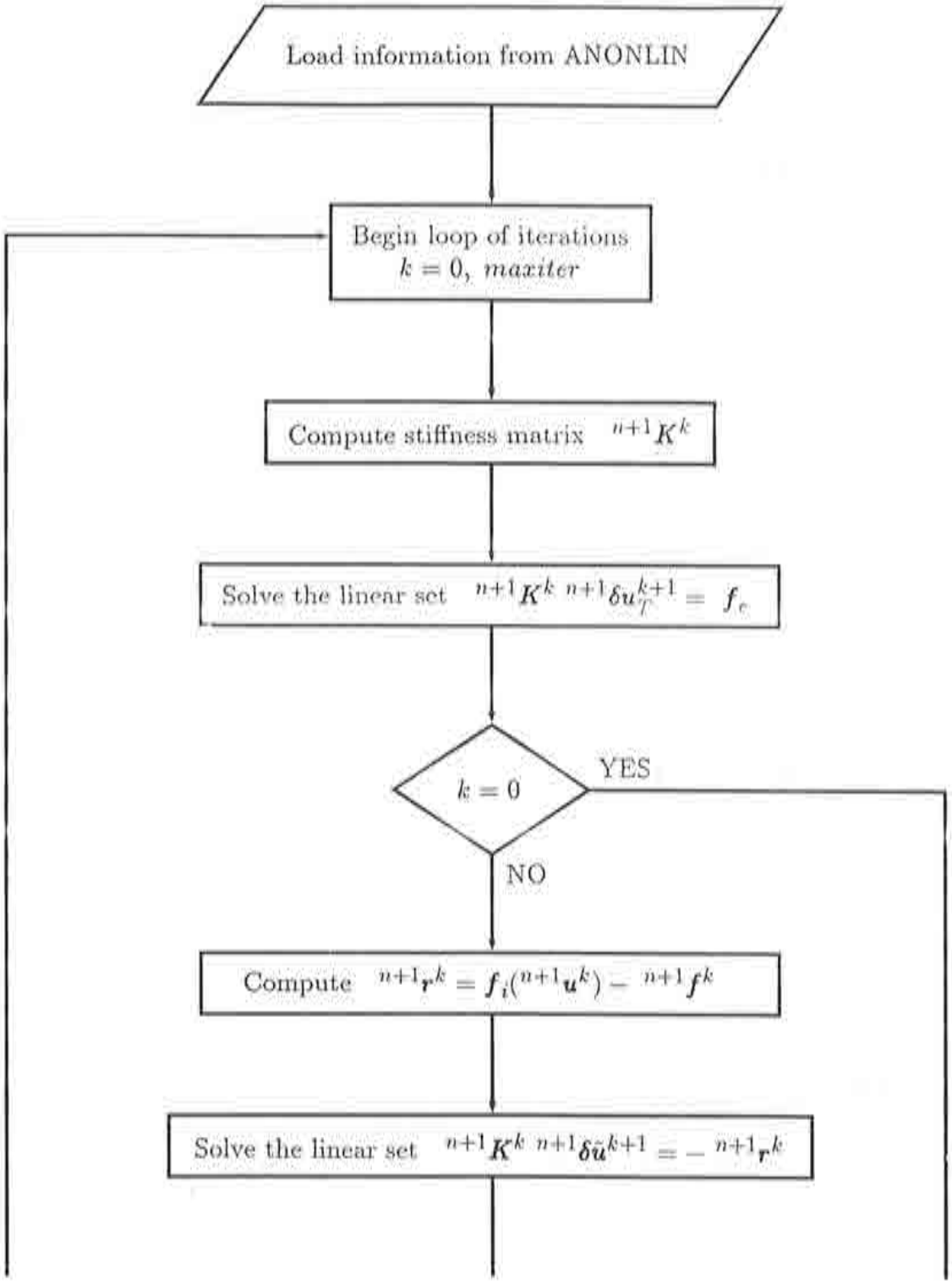


Figure 8.6: Flowchart for *procedure* AINCREME.

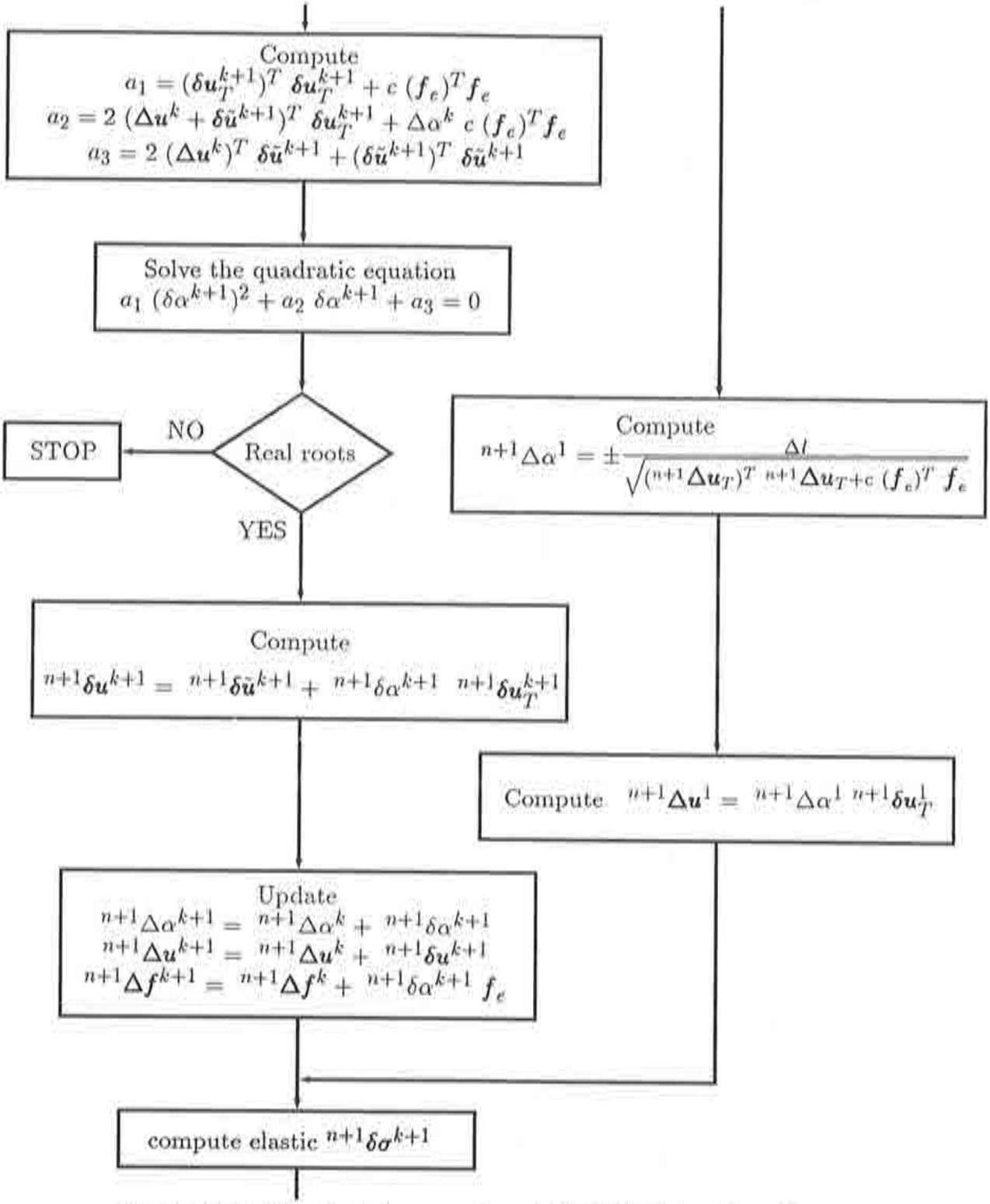


Figure 8.6: Flowchart for *procedure* AINCREME (continued).

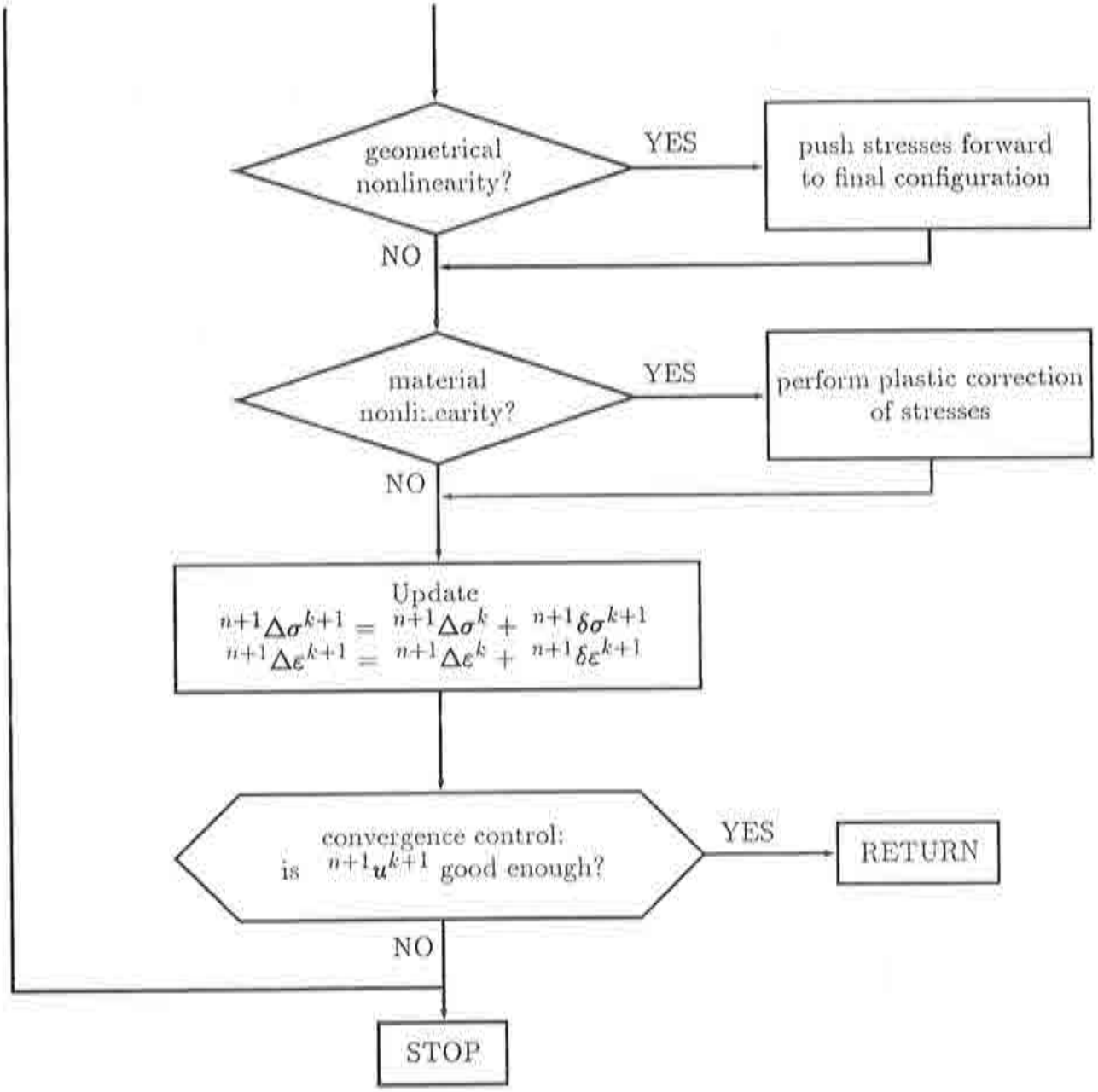


Figure 8.6: Flowchart for *procedure* AINCREME (continued).

Interpretation of the arc-length procedure in a Lagrange-multiplier context

When working with Castem, the number of unknowns increases since we have to deal with extended vectors $\boldsymbol{U}$, $\boldsymbol{R}$.

A reasonable generalisation of the arc-length condition (8.30) consists of using *several* scale parameters c_b, c_f, c_l, so that all vectors, $\boldsymbol{b}_e$ (imposed displacements), $\boldsymbol{f}_e$ (external loads) and $\boldsymbol{\lambda}/\boldsymbol{\mu}$ (Lagrange multipliers) are taken into consideration. The enlarged version of equation (8.30) then reads

$$(\Delta \boldsymbol{u})^T \Delta \boldsymbol{u} + (\Delta\alpha)^2 \Big[c_b (\boldsymbol{b}_e)^T \boldsymbol{b}_e + c_f (\boldsymbol{f}_e)^T \boldsymbol{f}_e + c_l \Big((\Delta\boldsymbol{\lambda})^T \Delta\boldsymbol{\lambda} + (\Delta\boldsymbol{\mu})^T \Delta\boldsymbol{\mu} \Big) \Big] = (\Delta l)^2.$$

As for the rest of the arc-length procedure, we will need to translate the equations in Section 8.2 into their enlarged version. This means, in particular, that both forces ($\boldsymbol{f}_e$) and prescribed displacements ($\boldsymbol{b}_e$) will be updated. We are going to use $c_l = 0$, since numerical tests have shown no improvement otherwise.

For instance, the prediction $\Delta\alpha^1$ will be obtained from

$$^{n+1}\Delta\alpha^1 = \pm \frac{\Delta l}{\sqrt{(^{n+1}\Delta \boldsymbol{U}_T)^T \; ^{n+1}\Delta \boldsymbol{U}_T + c_b \; (\boldsymbol{b}_e)^T \; \boldsymbol{b}_e + c_f \; (\boldsymbol{f}_e)^T \; \boldsymbol{f}_e}}, \tag{8.32}$$

where $^{n+1}\Delta \boldsymbol{U}_T$ is the solution of the linear set

$$\boldsymbol{J}^0 \; ^{n+1}\Delta \boldsymbol{U}_T = \boldsymbol{F}_e. \tag{8.33}$$

In equation (8.33), a new vector $\boldsymbol{F}_e$ has been defined as

$$\boldsymbol{F}_e = \begin{pmatrix} \boldsymbol{f}_e \\ \boldsymbol{b}_e \\ \boldsymbol{b}_e \end{pmatrix}.$$

For a generic iteration $k+1$, two augmented vectors $\boldsymbol{\delta \bar{U}}^{k+1}$ and $\boldsymbol{\delta U}_T^{k+1}$ are obtained by solving

$$\boldsymbol{J}^k \; \boldsymbol{\delta \bar{U}}^{k+1} = -\boldsymbol{R}(\boldsymbol{U}^k) \tag{8.34}$$

$$\boldsymbol{J}^k \; \boldsymbol{\delta U}_T^{k+1} = \boldsymbol{F}_e. \tag{8.35}$$

The three coefficients a_1, a_2, a_3 can then be computed from

$$a_1 = (\boldsymbol{\delta U}_T^{k+1})^T \; \boldsymbol{\delta U}_T^{k+1} + c_b \; (\boldsymbol{b}_e)^T \boldsymbol{b}_e + c_f \; (\boldsymbol{f}_e)^T \boldsymbol{f}_e \tag{8.36}$$

$$a_2 = 2 \; (\boldsymbol{\Delta U}^k + \boldsymbol{\delta \tilde{U}}^{k+1})^T \; \boldsymbol{\delta U}_T^{k+1} + \Delta \alpha^k \Big(c_b \; (\boldsymbol{b}_e)^T \boldsymbol{b}_e + c_f \; (\boldsymbol{f}_e)^T \boldsymbol{f}_e \Big) \tag{8.37}$$

$$a_3 = 2 \; (\boldsymbol{\Delta U}^k)^T \; \boldsymbol{\delta \tilde{U}}^{k+1} + (\boldsymbol{\delta \tilde{U}}^{k+1})^T \; \boldsymbol{\delta \tilde{U}}^{k+1}. \tag{8.38}$$

After solving the quadratic equation (8.22), we compute

$$\boldsymbol{\delta U}^{k+1} = \boldsymbol{\delta \tilde{U}}^{k+1} + \; \delta \alpha^{k+1} \; \boldsymbol{\delta U}_T^{k+1}$$

and finally update

$$\alpha^{k+1} = \alpha^k + \; \delta \alpha^{k+1}$$

$$\boldsymbol{U}^{k+1} = \boldsymbol{U}^k + \; \boldsymbol{\delta U}^{k+1}$$

$$\boldsymbol{F}^{k+1} = \begin{pmatrix} \boldsymbol{f}^{k+1} \\ \boldsymbol{b}^{k+1} \\ \boldsymbol{b}^{k+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}^k \\ \boldsymbol{b}^k \\ \boldsymbol{b}^k \end{pmatrix} + \delta \alpha^{k+1} \; \boldsymbol{F}_e.$$

8.4 Various numerical examples involving snap-through and snap-back

Example TRUSS

We consider first the truss example described in Figure 4.2. As it was already indicated in Section 4.4, this test results in an unreallistic load/deflection curve when analysed through 10 increments of load $\Delta W = 60$.

If we switch on to an arc-length strategy, we see that this is actually a case of snap-through. Figure 8.7 shows the *real* nondimensional load/deflection curve for this example. This curve has been obtained using a cylindrical arc-length formulation, thus with $c_b = c_f = c_l = 0$. An automatic control is performed using Crisfield's formula (8.28) with a desired number of iterations per increment $N_{opt} = 4$.

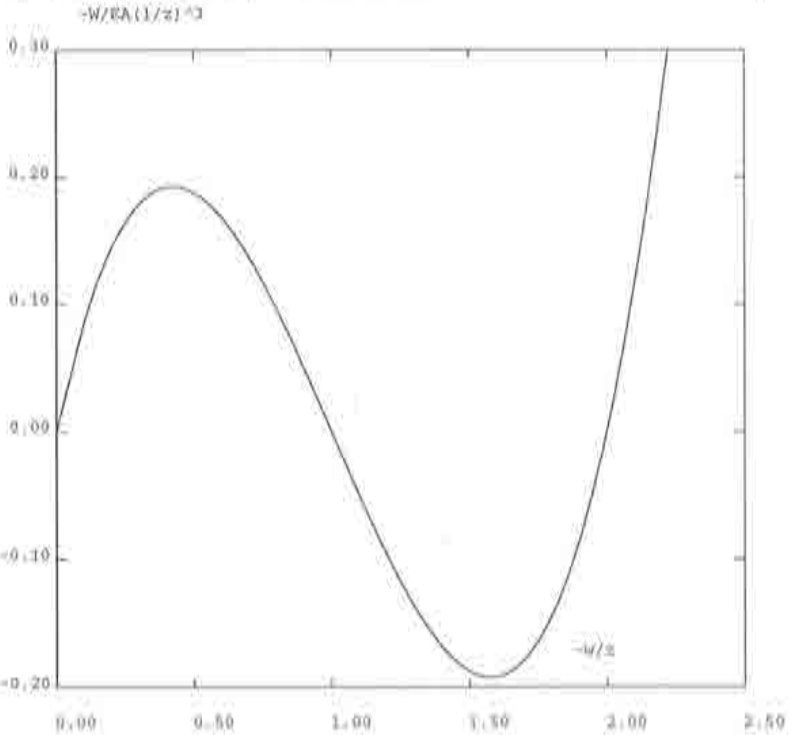


Figure 8.7: Truss test, arc-length formulation. Load vs. vertical displacement.

Example SHALLOW ARCH

In this case, we study the shallow arch represented in Figure 8.8, [6,19]. This arch has a rectangular section and is made of an elastic material. The structure is clamped at both ends and is subject to a central load F. A large-strain analysis of one half of the structure is performed using 10 beam elements. A spherical formulation with $c_b = c_f = 1, c_t = 0$ has been employed. An automatic control (8.28) is also used in this case with $N_{opt} = 4$.

Figure 8.9 shows the load/deflection curve corresponding to the centre point. As it can be seen from the figure, this is also a case of snap-through.

In Figure 8.10, the load is plotted vs. the horizontal reaction H.

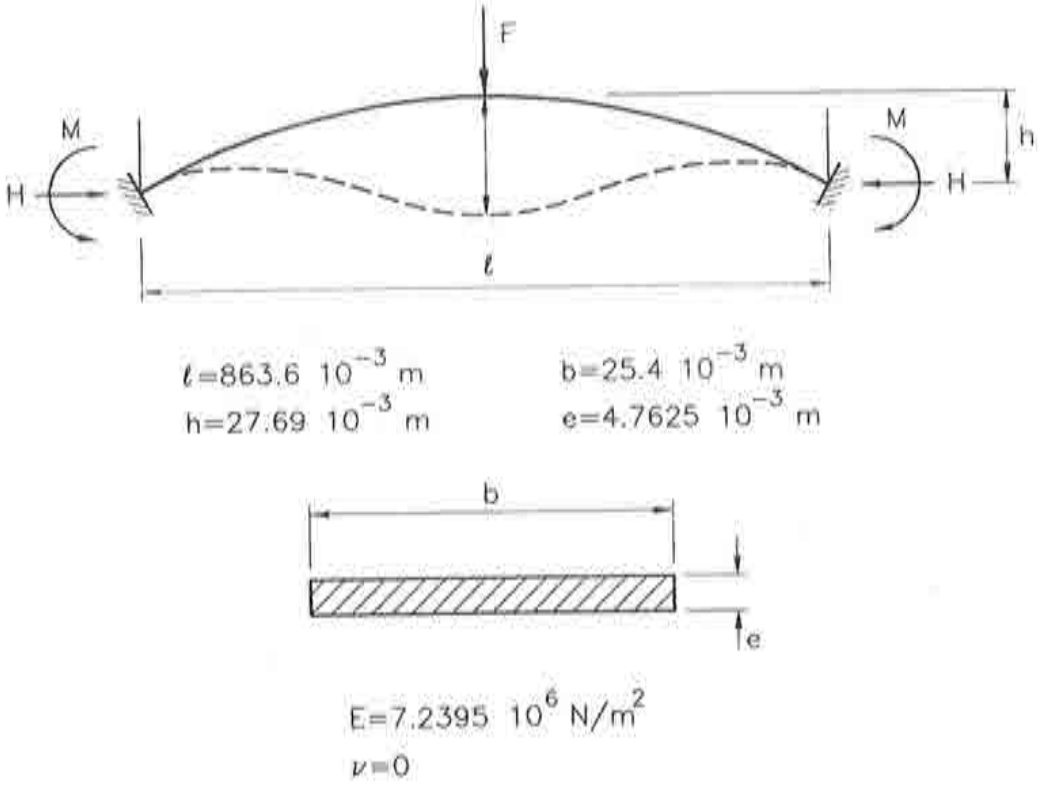


Figure 8.8: Shallow arch test.

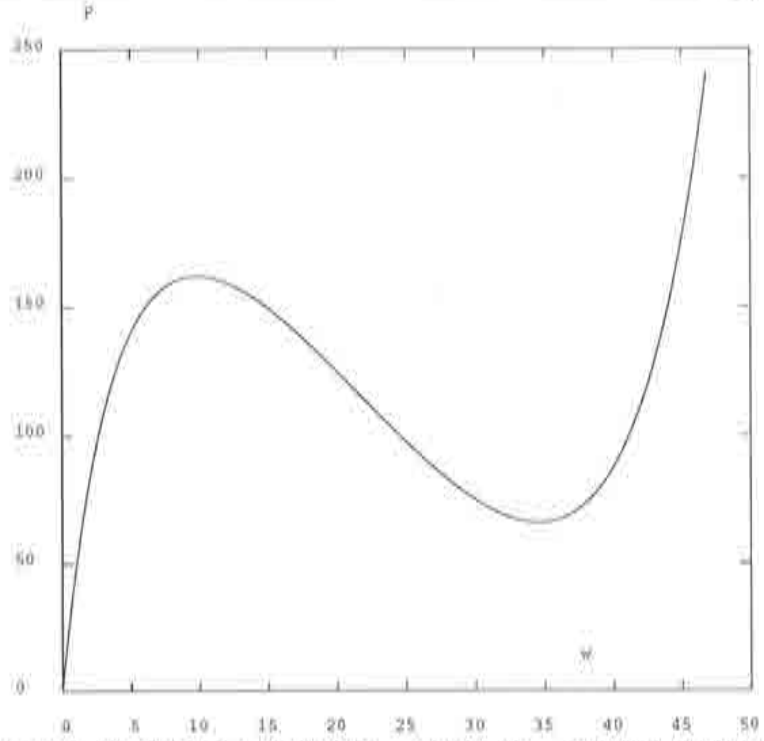


Figure 8.9: Shallow arch test. Load vs. vertical deflection at centre point.

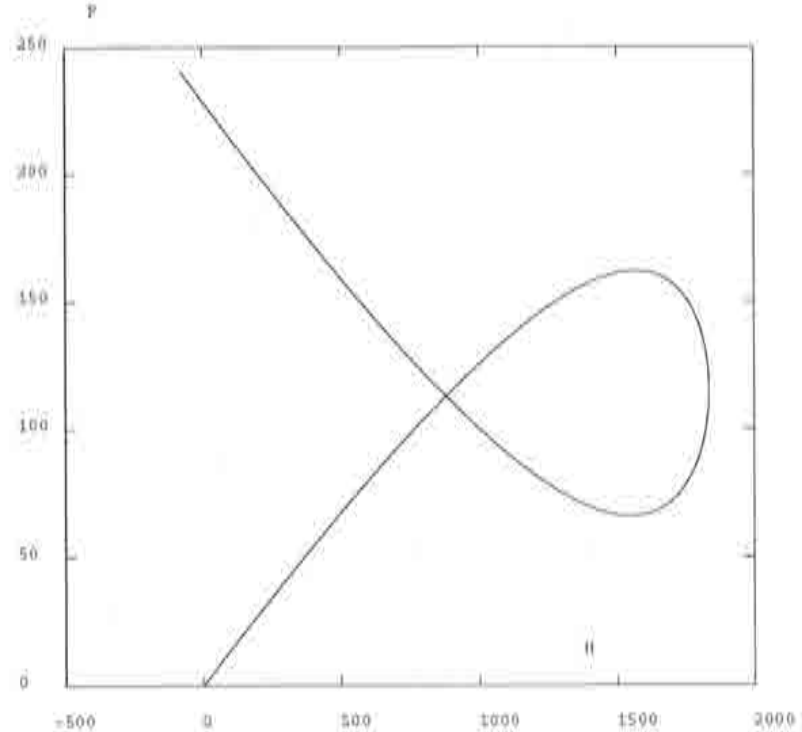


Figure 8.10: Shallow arch test. Load vs. horizontal reaction.

Example SHELL*

For this example, we recover the presentation of the shell test, see Figure 8.11. In this case, we keep all the geometric and mechanic parameters but we load the structure with an *eccentric* force P (hence the *), that is, we use $r \neq 0$. We are going to name the value $ecc = r/0.9$ *eccentricity* of the load.

Figure 8.12 shows the value of the load vs. the deflection of the *central* node for an eccentricity $ecc = 0.42$. This curve has been obtained by using a cylindrical arc-length formulation. The discretization consists of 14 axisymmetrical elements. An automatic update of the arc length is performed using equation (8.28).

In Figure 8.13, the load is plotted vs. the vertical deflection of the *loaded* node. As it can be seen from the figure, this curve has an initial steep slope and a snap-back tendency.

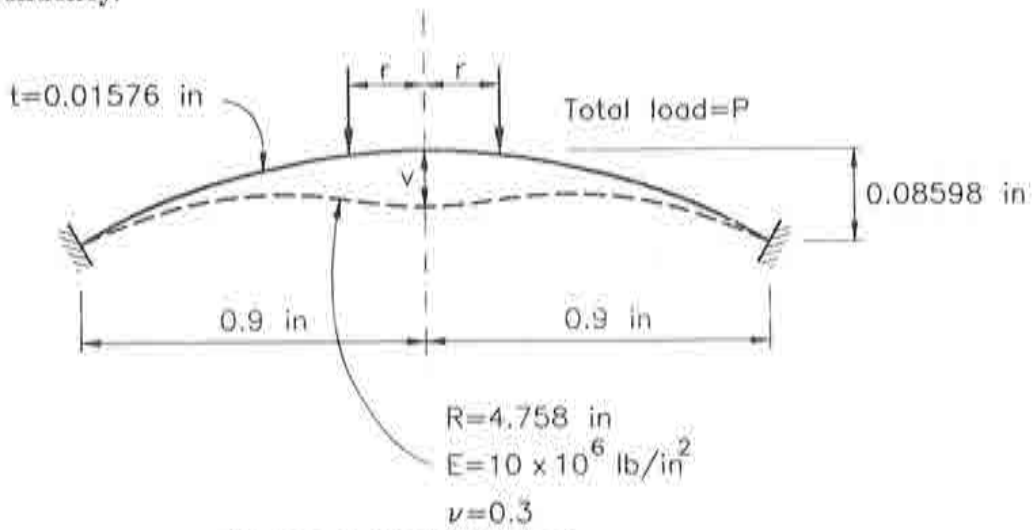


Figure 8.11: Shell* test.

If we continue to load the structure with increasingly eccentric forces, we obtain the curves in Figures 8.14, 8.15, 8.16 and 8.17 (note that different scales are used). These figures show the total load vs. the deflection of the loaded node, corresponding to eccentricities

Figure 8.14 : $ecc = 0.50$, Figure 8.15 : $ecc = 0.60$
Figure 8.16 : $ecc = 0.70$, Figure 8.17 : $ecc = 0.80$

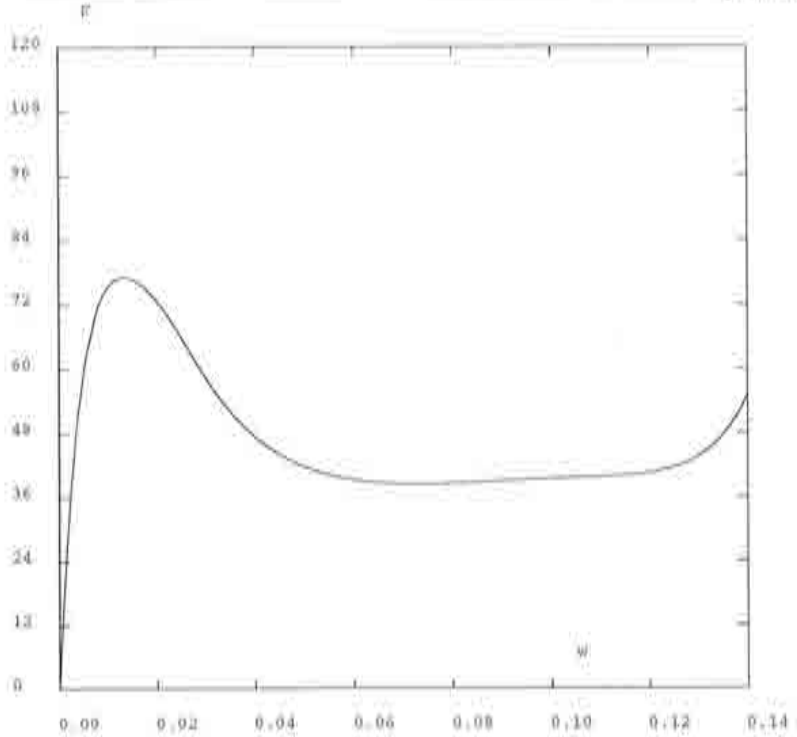


Figure 8.12: Shell* test, eccentricity 0.42. Load vs. vertical deflection of central node.

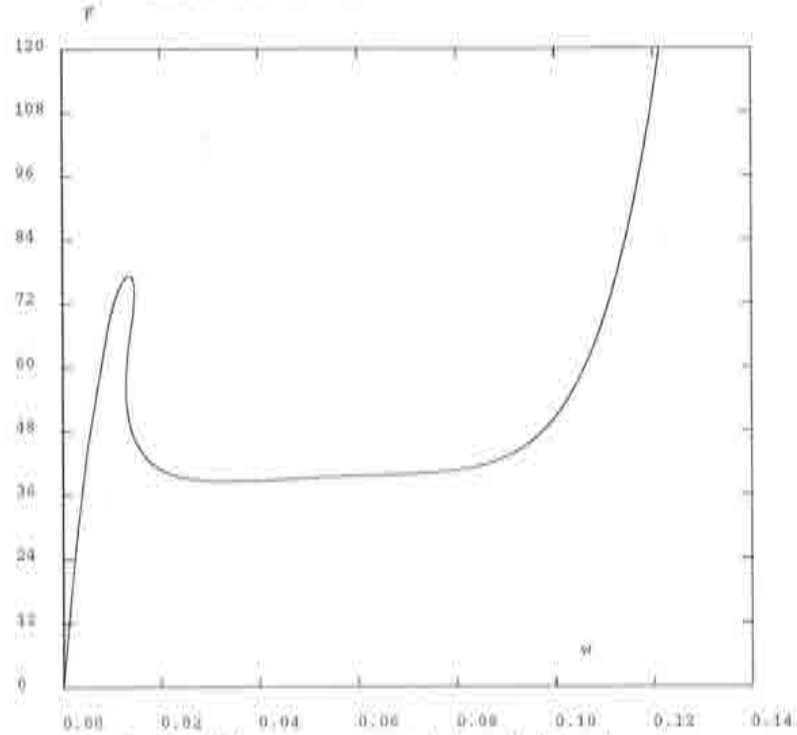


Figure 8.13: Shell* test, eccentricity 0.42. Load vs. vertical deflection of loaded node.

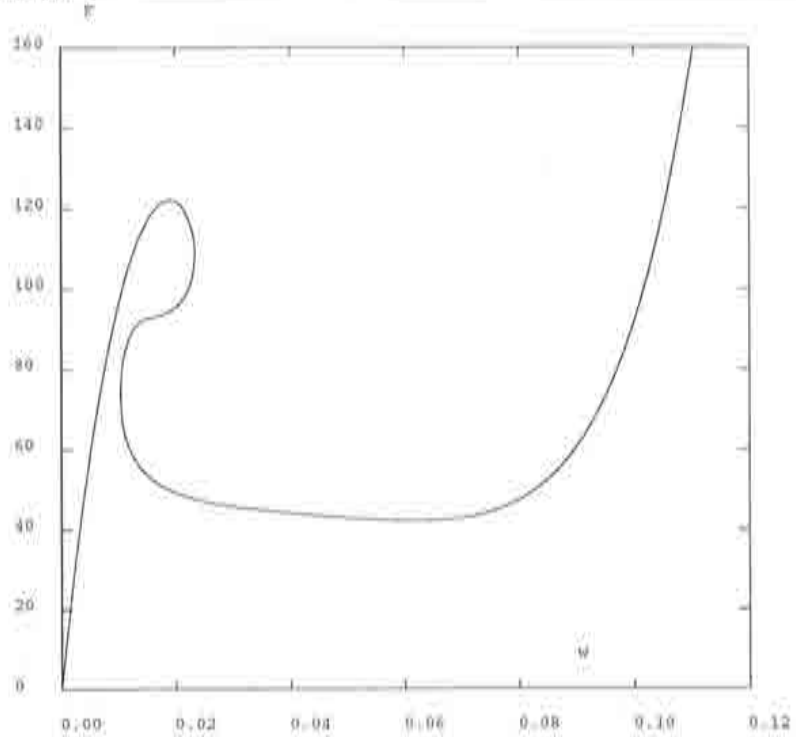


Figure 8.14: Shell* test, eccentricity 0.50. Load vs. vertical deflection of loaded node.

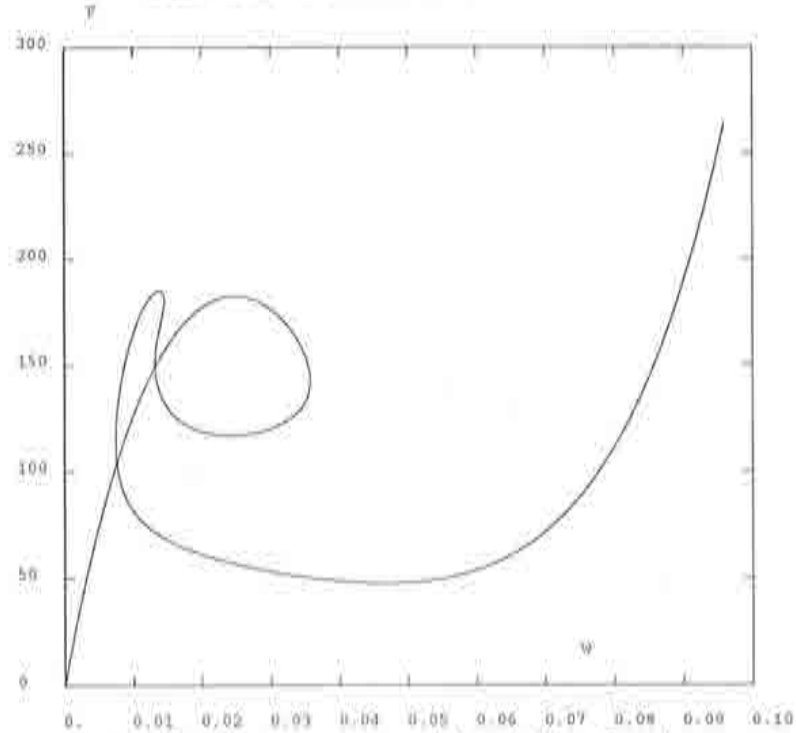


Figure 8.15: Shell* test, eccentricity 0.60. Load vs. vertical deflection of loaded node.

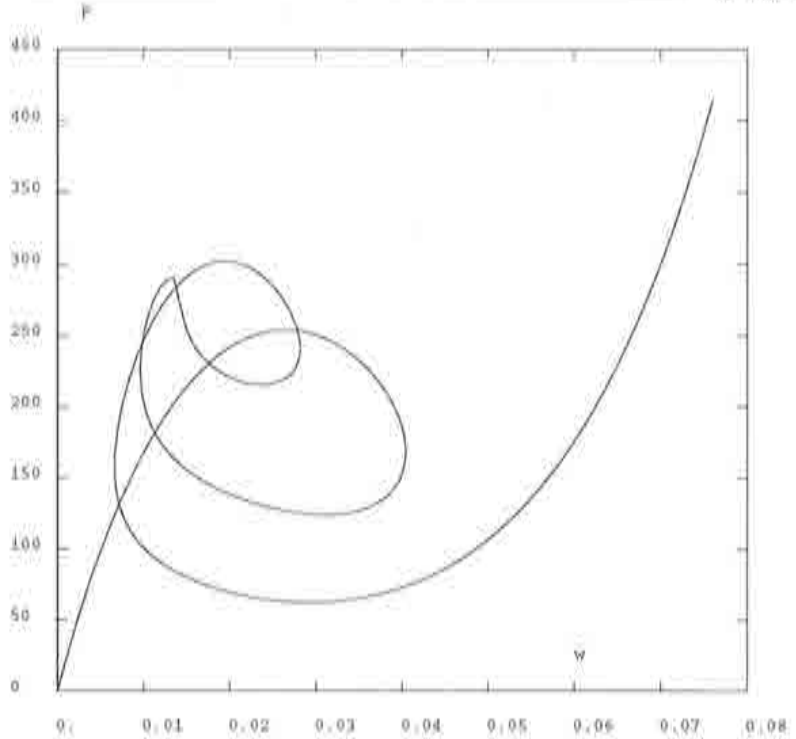


Figure 8.16: Shell* test, eccentricity 0.70. Load vs. vertical deflection of loaded node.

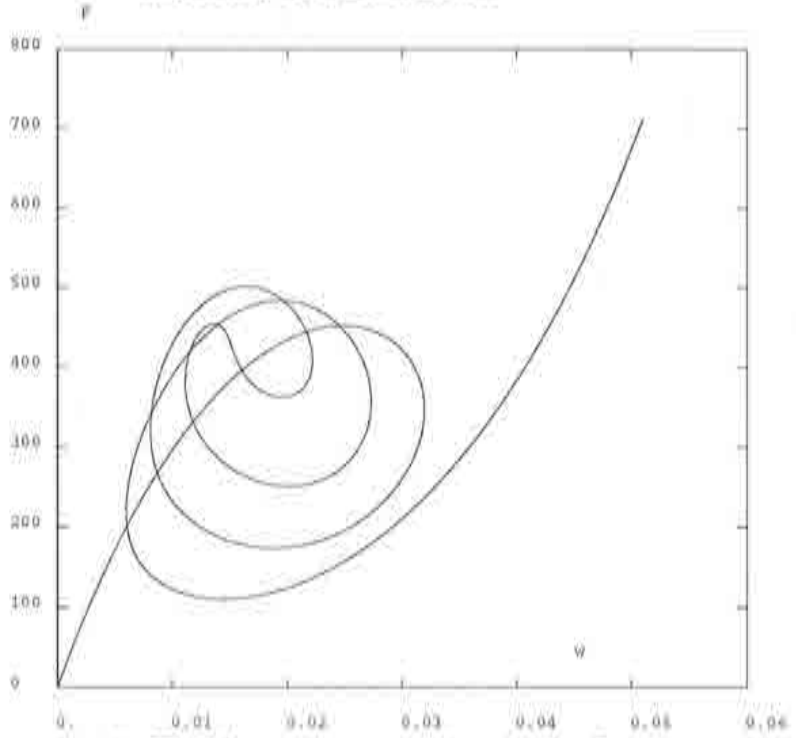


Figure 8.17: Shell* test, eccentricity 0.80. Load vs. vertical deflection of loaded node.

Example SHALLOW DOME

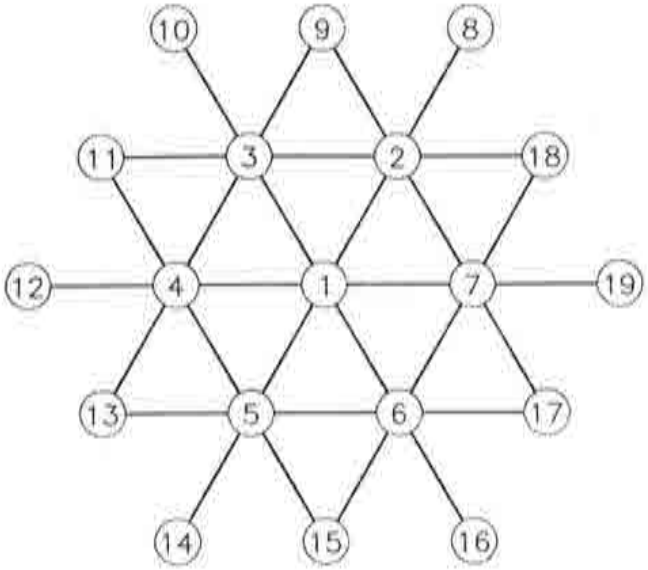
To end with the set of arc-length examples, the following test is studied. A truss structure in the shape of a shallow dome is loaded with a central force, [2,17]. This structure is made of an elastic material and it is pinned at the ground level, see Figure 8.18.

This is a highly nonlinear example with several equilibrium paths. A technique to detect all the paths is discussed in [17].

For a cylindrical formulation and the usual option for the automatic control of the arc length, the *primary* path, [2], has been obtained, Figure 8.19.

The detection of bifurcation points is beyond the scope of this work. However, other equilibrium paths can be traced by modifying manually the resolution strategy, see Figure 8.20. The idea that was used here consists of the following steps:

- Find a feature F that differentiates the chosen path from the others, for instance, two points that have symmetrical deflections for this case and not for the rest of the paths.
- Solve a *new* problem to which F has been introduced as an additional boundary condition.



COORDINATES OF THE NODE POINTS OF DOME STRUCTURE			
Node	X	Y	Z
1	0.0	0.0	6.0
3	−15.0	25.9807	4.5
4	−30.0	0.0	4.5
9	0.0	60.0	0.0
10	−30.0	51.9615	0.0
11	−51.9615	30.0	0.0
12	−60.0	0.0	0.0

$A_i = 0.1$ in^2

Nodes 1 to 7 are free

Nodes 8 to 19 are pinned

$E = 30\ 10^6$ lb/in^2

$\nu = 0$

Figure 8.18: Shallow dome test.

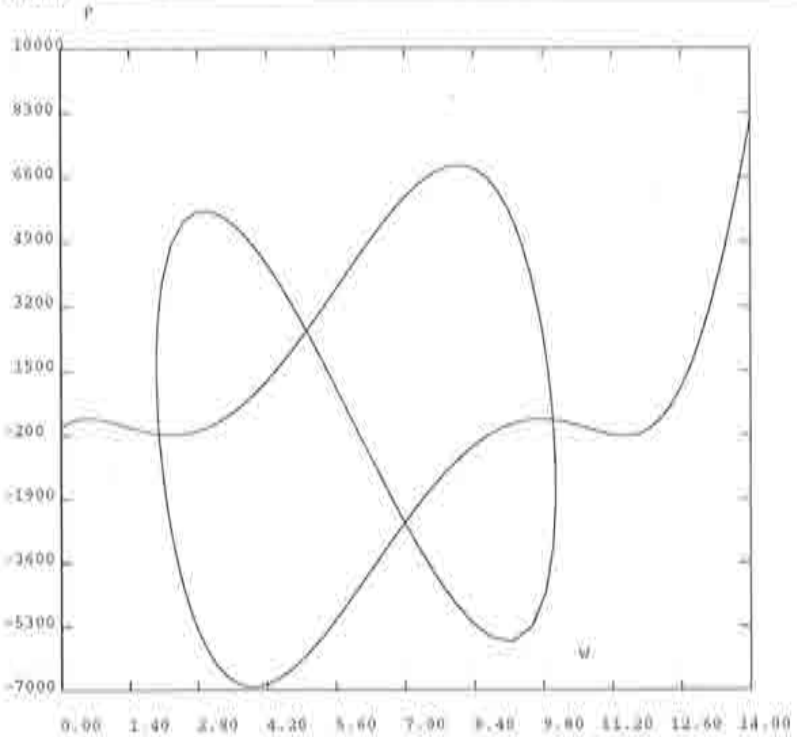


Figure 8.19: Shallow dome test, path 1. Load vs. vertical deflection at point 1.

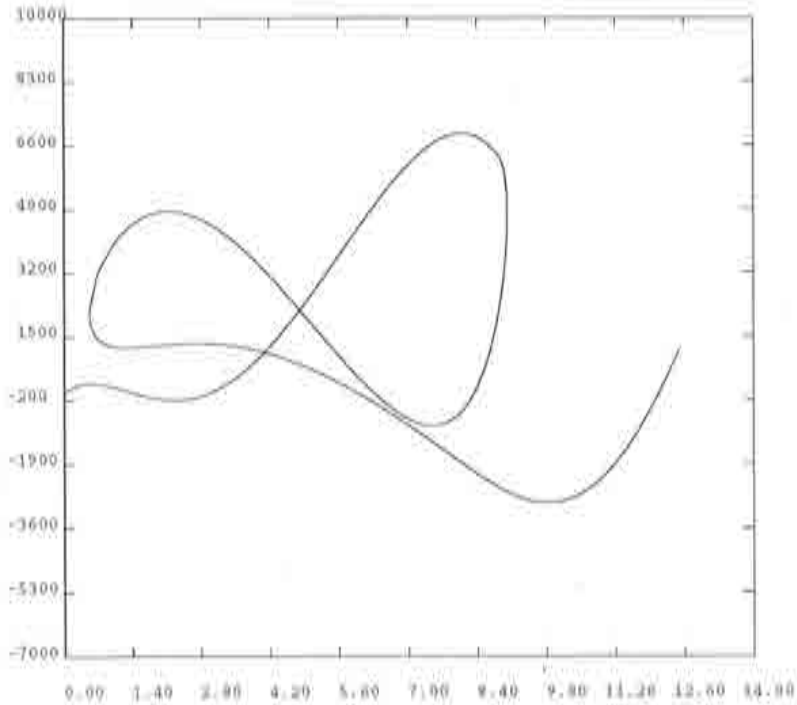


Figure 8.20: Shallow dome test, path 2. Load vs. vertical deflection at point 1.

computational efficiency of the algorithms, in terms of computing time, has not been affected by the object-oriented context, where information is stored and manipulated in a different manner from conventional codes.

Specific Quasi-Newton and Secant-Newton algorithms have been developed to solve nonlinear sets of equations with linear constraints. Their numerical performance has been compared to that of classical methods, with encouraging results. Beyond the context of nonlinear structural analysis of this work, these algorithms might be extended to other fields such as linearly constrained optimization.

References

[1] Bathe, K.J. (1982), "Finite Element Procedures in Engineering Analysis", Prentice Hall, New Jersey, USA.

[2] Bellini, P.X. & Chulya, A. (1987), "An improved automatic incremental algorithm for the efficient solution of nonlinear finite element equations", *Comput. Struct.*, **26**, pp. 99-110.

[3] Bretones, M.A. (1993), "Programación orientada al objeto. Una herramienta de ingeniería y desarrollo", Tesina de Especialidad, Universitat Politècnica de Catalunya, Barcelona.

[4] Brodlie, K.W., Gourlay, A.R. & Greenstadt, J. (1972), "Rank-one and rank-two corrections to positive definite matrices expressed in product form", *J. Inst. Maths. Applics.*, **11**, pp. 73-82.

[5] "Castem2000–manuel d'utilisation" (1991), Rapport 88/176, Laboratoire d'Analyse Mécanique des Structures, Commissariat à l'Énergie Atomique

[6] Cervera, M. , "Nonlinear analysis of reinforced concrete structures using three dimensional and shell finite element models", Ph.D. Thesis, University College of Swansea, Swansea.

[7] Cervera, M. & Lombera, G. (1992), "Omega V.92: Bench-mark tests de mecánica de sólidos en régimen no-lineal. Aplicaciones estáticas y dinámicas 2D y 3D", Informe Técnico IT-70, Centre Internacional de Mètodes Numèrics en Enginyeria, Barcelona.

[8] Crisfield, M.A. (1980), "A fast incremental/iterative solution procedure that handles 'snap-through' ", *Comput. Struct.*, **13**, pp. 55-62.

[9] Crisfield, M.A. (1983), "An arc-length method including line searches and accelerations", *Int. J. Num. Meth. Engng.*, **19**, pp. 1269-1289.

[10] Crisfield, M.A. (1991), "Non-linear finite element analysis of solids and structures", John Wiley & Sons, Chichester.

[11] Dennis , J.E. & Moré, J.J. (1977), "Quasi–Newton methods, motivation and theory", *SIAM Rev.*, **19**, pp. 46-89.

[12] Engelman, M.S. et al. (1981), "The application of Quasi–Newton methods in fluid mechanics", *Int. J. Num. Meth. Engng.*, **17**, pp. 707-718.

[13] Fafard, M. & Massicotte, B. (1993), "Geometrical interpretation of the arc-length method", *Comput. Struct.*, **46**, pp. 603-615.

[14] Fletcher, R. (1987), "Practical methods of optimization", John Wiley & Sons, Chichester.

[15] Hughes, T.J.R. (1987), "The finite element method. Linear static and dynamic finite element analysis", Prentice Hall, Englewood Cliffs, USA.

[16] Kouhia, R. & Mikkola, M. (1989), "Tracing the equilibrium paths beyond simple critical points", *Int. J. Num. Meth. Engng.*, **28**, pp. 2923-2941.

[17] Kwok, H.H., Kamat, M.P. & Watson, L.T. (1985), "Location of stable and unstable equilibrium configurations using a model trust region Quasi–Newton method and tunnelling", *Comput. Struct.*, **21**, pp. 909-916.

[18] Matthies, H. & Strang, G. (1979), "The solution of nonlinear finite element equations", *Int. J. Num. Meth. Engng.*, **14**, pp. 1613-1626.

[19] Oliver, J. (1982), "Una formulación cuasi-intrínseca para el estudio, por el método de los elementos finitos, de vigas, arcos, placas y láminas sometidos a grandes corrimientos en régimen elastoplástico", Tesis Doctoral, Universitat Politècnica de Catalunya, Barcelona.

[20] Oñate, E. (1991), "Cálculo de estructuras por el método de los elementos finitos. Análisis estático lineal", Centre Internacional de Mètodes Numèrics en Enginyeria, Barcelona.

[21] Pegon, P. & Anthoine, A. (1994),"Numerical strategies for solving continuum damage problems involving softening: application to the homogenization of masonry", *Proc. of the Second Intl. Conf. Comput. Struct. Tech.*, Athens.

[22] Riks, E. (1979), "An incremental approach to the solution of snapping and buckling problems", *Int. J. Solids. Struct.*, **15**, pp. 529-551.

[23] Riks, E. (1992), "On formulations of path-following techniques for structural stability analysis", in New Advances in Computational Structural Mechanics, editors P. Ladevèze & O.C. Zienkiewicz, Elsevier.

[24] Schweizerhof, K.H. & Wriggers, P. (1986), "Consistent linearization for path following methods in nonlinear FE analysis", *Comput. Meth. Appl. Mech. Engng.*, **59**, pp. 261-279.

[25] Sherman, J. & Morrison, W.J. (1949), "Adjustment of an inverse matrix corresponding to changes in the elements of a given column or a given roww of the original matrix", *Ann. Math. Statist.*, **20**, p. 621.

[26] Simo, J.C. (1988), "A Framework for Finite Strain Elastoplasticity Based on Maximum Plastic Dissipation and the Multiplicative Decomposition. Part II: Computational Aspects", *Comp. Meth. Appl. Mech. Engrg.*, **68**, pp. 1-31.

[27] Soria, A. (1990), "Contribución al análisis de transitorios térmicos accidentales en los componentes de un reactor nuclear", Tesis Doctoral, Universidad Politécnica de Madrid, Madrid.

[28] "Système Castem2000. Programme Gibi" Reference Manual.

[29] Wempner, G.A. (1971), "Discrete approximations related to nonlinear theories of solids", *Int. J. Solids Struct.*, **7**, pp. 1581-1599.

[30] Zienkiewicz, O.C. & Taylor, R.C. (1989), "The finite element method. Vol. 1, Basic formulation and linear problems", McGraw-Hill, London.

[31] Zienkiewicz, O.C. & Taylor, R.C. (1991), "The finite element method, Vol. 2, Solid and fluid mechanics, dynamics and nonlinearity", McGraw-Hill, London.