10 Computer Implementation

10.1 Introduction

We have seen in the previous chapters that the solution to the nonlinear equilibrium equations is basically achieved using the Newton–Raphson iterative method. In addition, in a finite element context it is advisable to apply the external forces in a series of increments. This has the advantage of enhancing the converging properties of the solution and, where appropriate, provides possible intermediate equilibrium states. Moreover, for path-dependent materials such as those exhibiting plasticity, these intermediate states represent the loading path which needs to be accurately followed. Furthermore, it is clear that the two fundamental quantities that facilitate the Newton–Raphson solution are the residual force and the tangent matrix. In this chapter we shall describe the MATLAB implementation of the solution procedure in the teaching program FLagSHyP (**F**inite element **Larg**e **S**train **Hy**perelasto-plastic **P**rogram).

It is expected that the reader already has some familiarity with the computer implementation of the finite element method in the linear context. Consequently, this chapter will emphasize only those aspects of the implementation that are of particular relevance in the nonlinear finite deformation context.

The program description includes user instructions, a dictionary of variables and functions, and sample input and output for a few typical examples. The program can deal with three-dimensional truss elements and a number of two-dimensional and three-dimensional elements, together with a variety of compressible and nearly incompressible hyperelastic constitutive equations including simple Von Mises hyperelastic–plastic behavior. It can be obtained, together with sample data, as a download from the website `www.flagshyp.com`. Alternatively, it can be obtained by email request to any of the authors: `a.j.gil@swansea.ac.uk`, `r.d.wood@swansea.ac.uk` or `j.bonet@swansea.ac.uk`.

The master m-file `FLagSHyP.m`, which controls the overall organization of the program, is divided into three sections. The first section includes a series of statements designed to add the necessary directories to the path of the program. The second section includes the function `input_data_and_initialisation.m`, which is devoted to the reading of input/control data and the initialization of critical variables including equivalent nodal forces and the initial tangent stiffness matrix. The third section calls one of the three fundamental algorithms incorporated into FLagSHyP, depending on the problem-dependent control variables chosen by the user:

ALG1 The function `Newton_Raphson_algorithm.m` is the basic Newton–Raphson algorithm, to be explained thoroughly in the remainder of this chapter.

ALG2 The function `Line_Search_Newton_Raphson_algorithm.m` combines the basic Newton–Raphson algorithm with the line search procedure described in Section 9.6.2.

ALG3 The function `Arc_Length_Newton_Raphson_algorithm.m` combines the basic Newton–Raphson algorithm with the arc length procedure described in Section 9.6.3.

The basic Newton–Raphson algorithm is described in Box 9.1 and amplified in Box 10.1 to include the line search and arc length options. The items in parentheses in Box 10.1 refer to program segments, each presented in Section 10.7 in a separate box in which comments have been removed and only MATLAB instructions are shown, along with a short description.

Box 10.1: Solution Algorithm (`ALG1`)

- INPUT geometry, material properties and solution parameters (*FLagSHyP segment 2*)
- INITIALIZE $\mathbf{F} = 0$, $\mathbf{x} = \mathbf{X}$ (initial geometry), $\mathbf{R} = 0$ (*FLagSHyP segment 2*)
- FIND initial $\mathbf{K}$ (*FLagSHyP segment 2*)
- LOOP over load increments (*FLagSHyP segment 5*)
  - SET $\lambda = \lambda + \Delta\lambda$, $\mathbf{F} = \lambda\bar{\mathbf{F}}$, $\mathbf{R} = \mathbf{R} - \Delta\lambda\bar{\mathbf{F}}$ (*FLagSHyP segment 5*)
  - IF PRESCRIBED DISPLACEMENTS: (*FLagSHyP segment 5*)
    - UPDATE GEOMETRY (*FLagSHyP segment 5*)
    - FIND $\mathbf{F}, \mathbf{T}, \mathbf{K}, \mathbf{R} = \mathbf{T} - \mathbf{F}$ (*FLagSHyP segment 5*)
  - END IF (*FLagSHyP segment 5*)
  - DO WHILE ($\|\mathbf{R}\|/\|\mathbf{F}\| >$ tolerance) (*FLagSHyP segment 6*)
    - SOLVE $\mathbf{K}\mathbf{u} = -\mathbf{R}$ (*FLagSHyP segment 6*)
    - IF LINE SEARCH FIND η (`ALG2`)
    - IF ARC-LENGTH FIND λ (`ALG3`)
    - UPDATE GEOMETRY $\mathbf{x} = \mathbf{x} + \mathbf{u}$ (*FLagSHyP segment 6*)
    - FIND $\mathbf{F}, \mathbf{T}, \mathbf{K}, \mathbf{R} = \mathbf{T} - \lambda\bar{\mathbf{F}}$ (*FLagSHyP segment 6*)
  - END DO (*FLagSHyP segment 6*)
  - OUTPUT INCREMENT RESULTS (*FLagSHyP segment 7*)
- ENDLOOP (*FLagSHyP segment 8*)

A feature of the FLagSHyP program is the arrangement of variables (and the grouping of those sharing a similar nature) into data structures. The use of data structures facilitates the understanding of all variables in the program, although their use can introduce some computational overheads, noticeable in a large-scale problem. Fundamental data structures are named in capital letters and their "children" in lower-case letters. The following table presents the parent data structures with their general description.

Name	Comments
`PRO.`	Program run mode, input/output files, problem title
`GEOM.`	Nodes, initial and spatial coordinates, elemental volumes
`FEM.`	Finite element information (shape functions) and connectivity

QUADRATURE.	Gauss point information: location and weights
KINEMATICS.	Kinematics quantities: $\boldsymbol{F}$, $\boldsymbol{b}$, J, ...
MAT.	Material properties: ρ, λ, μ, κ, ...
BC.	Boundary conditions: fixed and free
LOAD.	Loads: point loads, gravity, pressure load
GLOBAL.	Global vectors and matrices: residual and stiffness
PLAST.	Plasticity variables
CON.	Program control parameters: line search, arc length, ...

As an example, the table below describes some of the children of parent data structures. A glossary of all the variables in the program is presented in Appendix 10.13 at the end of this chapter.

Parent	**Child**	**Comments**
PRO.	outputfile_name	Output file name
GEOM.	ndime	Number of dimensions
FEM.	mesh.	Data structure with mesh information
QUADRATURE.	element.	Gauss point element information
KINEMATICS.	F	Deformation gradient at Gauss points
MAT.	props	Material properties
BC.	freedof	Free degree-of-freedom vector
LOAD.	n_pressure_loads	Number of pressure surface elements
GLOBAL.	Residual	Global residual force vector
PLAST.	yield.	Data structure for the yield function
CON.	dlamb	Incremental load parameter

10.2 User instructions

In the execution of the master m-file FLagSHyP.m, the directories (and subdirectories) code and job_folder are added to the directory path of the program. For simplicity, all the necessary MATLAB functions comprising the FLagSHyP program are located within subdirectories of the directory code, whereas input/output data files are located within subdirectories of the directory job_folder. Moreover, the program expects the input file (e.g. FLagSHyP_input_file.dat) to be located in the directory job_folder/FLagSHyP_input_file. Notice that while the name (e.g. FLagSHyP_input_file) or extension (e.g. .dat) of the input file is arbitrary, it must coincide precisely with the name of the subdirectory which contains it (e.g. job_folder/FLagSHyP_input_file); otherwise an error will occur and the program will not progress. The input file required by FLagSHyP is described in the following table. The file is free-formatted, so items within a line are simply separated by commas or spaces.

Item 1: Title	**Number of lines:** 1
PRO.title	Problem title
Item 2: Type of element	**Number of lines:** 1
FEM.mesh.element_type	Element type (see Note 1)
Item 3: Number of nodes	**Number of lines:** 1

`GEOM.npoin`	Number of mesh nodes
Item 4: Nodal information	**Number of lines:** `GEOM.npoin`

`ip,`
`BC.icode(ip),`
`GEOM.x(i,ip)`
`[i=1:GEOM.ndime]`
	Node number
	Boundary code (see Note 2)
	x, *y*, *z* coordinates
	[Number of dimensions]

Item 5: Number of elements	**Number of lines:** `1`
`FEM.mesh.nelem`	Number of elements
Item 6: Element information	**Number of lines:** `FEM.mesh.nelem`

`ie,`
`MAT.matno(ie),`
`FEM.mesh.connectivity(ie,i)`
`[i=1:FEM.mesh.n_nodes_elem]`
	Element number
	Material number
	Connectivities
	[Number of nodes per element]

Item 7: Number of materials	**Number of lines:** `1`
`MAT.nmats`	Number of different materials
Item 8: Material properties	**Number of lines:** `MAT.nmats`

`im,`
`MAT.matyp(im),`
`MAT.props(ipr,im)`
`[ipr=1:npr(dependent upon material)]`
	Material number
	Constitutive equation type
	Properties (see Note 3)
	[Number of properties]

Item 9: Load information	**Number of lines:** `1`

`n_point_loads,`
`BC.n_prescribed_displacements,`
`LOAD.n_pressure_loads,`
`LOAD.gravt(i)`
`[i=1:GEOM.ndime]`
	Number of loaded nodes
	Number of nonzero prescribed displacements
	Number of surface (line) elements with pressure
	Gravity vector
	[Number of dimensions]

Item 10: Nodal loads	**Number of lines:** `n_point_loads`

`ip,`
`force_value(i),`
`[i=1:GEOM.ndime]`
	Node number
	Force vector
	[Number of dimensions]

Item 11: Prescribed displacements	**Number of lines:** `BC.n_prescribed_displacements`

`ip,`
`id,`
`prescribed_value`
	Node number
	Spatial direction
	Nominal prescribed displacement

Item 12: Pressure loads	**Number of lines:** `LOAD.n_pressure_loads`

`ie,`
`FEM.mesh.connectivity_faces(ie,in),`
`LOAD.pressure(ie)`
`[in=1:FEM.mesh.connectivity_faces]`
	Surface element number
	Force vector
	Nominal pressure (see Note 4)
	[Number of nodes per element]

Item 13: Control information	**Number of lines:** `1`

`CON.nincrm`
`CON.xlmax`
`CON.dlamb`
`CON.miter`
`CON.cnorm`
`CON.searc`
`CON.ARCLEN.arcln`
`CON.OUTPUT.incout`
`CON.ARCLEN.itarget`
`CON.OUTPUT.nwant`
`CON.OUTPUT.iwant`
	Number of load/displacement increments
	Maximum value of load-scaling parameter
	Load parameter increment
	Maximum number of iterations per increment
	Convergence tolerance
	Line search parameter (if `0.0` not in use)
	Arc length parameter (if `0.0` not in use)
	Output counter (e.g. `5` for every five increments)
	Target iterations per increment (see Note 5 below)
	Single output node (`0` if not used)
	Output degree of freedom (`0` if not used)

Note 1: The following element types (`FEM.mesh.element_type`) are recognized (see

also Section 10.4 for more details about these elements):

`truss2`: 2-noded truss
`tria3`: 3-noded linear triangle
`tria6`: 6-noded quadratic triangle
`quad4`: 4-noded bilinear quadrilateral
`tetr4`: 4-noded linear tetrahedron
`tetr10`: 10-noded quadratic tetrahedron
`hexa8`: 8-noded trilinear hexahedron.

Different element types cannot be mixed in a mesh. Given the element name, the program automatically sets the number of nodes per element (`FEM.mesh.n_nodes_elem`), the number of dimensions (`GEOM.ndime`), and the number of Gauss points (`QUADRATURE.element.ngauss`). It also identifies the associated type of surface or line element for pressure loads and sets the corresponding number of nodes per element (`FEM.mesh.n_face_nodes_elem`) and Gauss points (`QUADRATURE.boundary.ngauss`).

Note 2: The boundary codes (`BC.icode`) are as follows:

0: free

1: *x* prescribed

2: *y* prescribed

3: *x, y* prescribed

4: *z* prescribed

5: *x, z* prescribed

6: *y, z* prescribed

7: *x, y, z* prescribed.

Prescribed degrees of freedom are assumed to be fixed (that is, no displacement) unless otherwise prescribed to be different from zero in input item 9 or 11. If a displacement is imposed in input item 9 or 11, the corresponding degree of freedom must have already been indicated as prescribed in input item 4.

Note 3: The following constitutive equations have been implemented (see also Section 10.6):

1: plane strain or three-dimensional compressible neo-Hookean

2: one-dimensional stretch-based hyperelastic plastic (`truss2` only)

3: plane strain or three-dimensional hyperelastic in principal directions

4: plane stress hyperelastic in principal directions

5: plane strain or three-dimensional nearly incompressible neo-Hookean

6: plane stress incompressible neo-Hookean

7: plane strain or three-dimensional nearly incompressible hyperelasticity in principal directions

8: plane stress incompressible hyperelasticity in principal directions

17: plane strain or three-dimensional nearly incompressible hyperelastic plastic in principal directions.

The corresponding list of properties `MAT.props(i,im)` to be read in Item 8[†] are shown in the following table. For simplicity, the columns of the table have been named as `props(i)`, which refer to the specific variable name `MAT.props(i,im)`. In this table, ρ represents the density in the reference configuration, λ and μ are the Lamé coefficients, $\kappa = \lambda + 2\mu/3$ is the bulk modulus,[*] h is the thickness for plane stress cases, E is the Young's modulus, ν is Poisson's ratio, and *area* is the initial cross-sectional area. Finally, τ_y and H are the yield stress and hardening parameter, respectively.

Note 4: For surface elements the direction of positive pressure is given by the right-hand screw rule following the surface element numbering. For line elements the positive pressure is at 90° turning counterclockwise to the direction of increasing node numbering.

Type	`props(1)`	`props(2)`	`props(3)`	`props(4)`	`props(5)`	`props(6)`
1	ρ	μ	λ	–	–	–
2	ρ	E	ν	*area*	τ_y	H
3	ρ	μ	λ	–	–	–
4	ρ	μ	λ	h	–	–
5	ρ	μ	κ	–	–	–
6	ρ	μ	h	–	–	–
7	ρ	μ	κ	–	–	–
8	ρ	μ	h	–	–	–
17	ρ	μ	λ	–	τ_y	H

Note 5: Typically, a nonlinear structural analysis solution is carried out in a number of incremental steps, `CON.incrm=1:CON.nincr`, where `CON.nincr` is the chosen maximum number of steps.

The applied *loads* can be point forces, pressure forces, or even prescribed displacements. Any input value of these items is nominal and will be multiplied by the load parameter increment `CON.dlamb`. The nominal load multiplied by `CON.dlamb` is called an *increment in load*, $\Delta\mathbf{F}$ (or $\Delta\mathbf{u}$ for prescribed displacements).

Typically, at each load step `CON.incrm`, the applied load is increased by an amount equal to $\Delta\mathbf{F}$. This means that the nominal load has been multiplied by `lambda=CON.incrm×CON.dlamb` to give the actual load $\mathbf{F}$, where `lambda` is called the *load-scaling parameter*.

If arc length control is employed then `lambda` is controlled indirectly; see Section 9.6.3. A positive value of `CON.ARCLEN.arcln` will produce a variable arc length, the value of which is determined by the desired number of iterations per increment `itarget`. Irrespective of the magnitude input by the user (i.e. `CON.ARCLEN.arcln=1`), the program will work out the most appropriate value,

discarding that entered by the user. A negative value (simply as an indicator) for CON.ARCLEN.arcln will provide a constant arc length. In this latter case, some experimentation with values will be necessary. If the arc length option is not to be used, input CON.ARCLEN.arcln =0 and CON.ARCLEN.itarget =0.

There is an additional control item, CON.xlmax, which is the maximum value of the load-scaling parameter, and the program will stop when (CON.incrm × CON.dlamb) > CON.xlmax, even if CON.incrm< CON.nincr. Alternatively, the program will stop when CON.incrm = CON.nincr even if (CON.nincr × CON.dlamb) < CON.xlmax.

At each load step the Newton–Raphson iteration attempts to achieve equilibrium within a maximum allowed number of iterations, input as CON.miter. Equilibrium is achieved when the residual force $||\mathbf{R}|| <$ CON.cnorm, where CON.cnorm is the convergence tolerance, usually about 10^{-6}.

Convergence toward equilibrium can be improved by using the line search option, which minimizes the residual force in the direction of the Newton–Raphson iterative change in the position of the structure. Line search is activated and controlled by the parameter CON.searc, which should have a value of about 0.5. The line search algorithm cannot be used in conjunction with the arc length method (a warning message is displayed in that case and the program stops).

To facilitate easy plotting of load displacement graphs, output from a single node and single degree of freedom (at that node) can be specified. Parameter CON.OUTPUT.nwant specifies the node and CON.OUTPUT.iwant the degree of freedom. The output is in a file always called flag.out which contains the increment number, the coordinate relating to the degree of freedom, the force relating to the degree of freedom, the current value of CON.xlamb, and (if used) the current arc length value CON.ARCLEN.arcln.

The somewhat contrived example shown in Figure 10.1 has been chosen to illustrate as many diverse features of these input instructions as possible. The required input file is listed in Box 10.2. Note that point loads, gravity loads, pressure loads, and prescribed displacements are all subject to the same incrementation in the solution procedure.

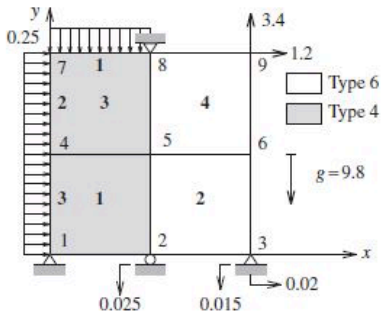


Figure 10.1 Simple two-dimensional example.

Box 10.2: Input File for Example in Figure 10.1

```
2-D Example quad4
9
1 3 0.0 0.0
2 2 1.0 0.0
3 3 2.0 0.0
4 0 0.0 1.0
5 0 1.0 1.0
6 0 2.0 1.0
7 0 0.0 2.0
8 3 1.0 2.0
9 0 2.0 2.0
4
1 1 1 2 5 4
2 2 6 5 2 3
3 1 5 8 7 4
4 2 5 6 9 8
2
1 4
1.0 100. 100. 0.1
2 6
1.0 100. 0.1
1 3 3 0.0 −9.8
9 1.2 3.4
3 1 0.02
2 2 −0.025
3 2 −0.015
1 8 7 0.25
2 7 4 0.25
3 1 4 −0.25
2 10.0 5.0 25 1.e−10 0.0 0.0 1 5 7 1
```

10.3 Output file description

The program FLagSHyP produces a largely unannotated output file that is intended as an input file for a postprocessor to be supplied by the user. The output file is only produced every `CON.OUTPUT.incout` increments. The contents and structure of this file are shown in the table below.

Item 1: Title	**Number of lines:** `1`

`PRO.title,`
`' at increment: ', num2str(CON.incrm),`
`' load: ', num2str(CON.xlamb)`
	Problem title
	Increment number
	Current load parameter

Item 2: Type of element	**Number of lines:** `1`
`FEM.mesh.element_type`	Element type
Item 3: Number of nodes	**Number of lines:** `1`
`GEOM.npoin`	Number of mesh nodes
Item 4: Nodal information `ip,`	**Number of lines:** `GEOM.npoin` Node number

`BC.icode(ip),`
`GEOM.x(i,ip),`
`force(i)`
`[i=1:GEOM.ndime]`
	Boundary code
	x, y, z coordinates
	External force or reaction
	[Number of dimensions]

Item 5: Number of elements	**Number of lines:** `1`
`FEM.mesh.nelem`	Number of elements
Item 6: Element information	**Number of lines:** `FEM.mesh.nelem`

`ie,`
`MAT.matno(ie),`
`FEM.mesh.connectivity(ie,i)`
`[i=1:FEM.mesh.n_nodes_elem]`
	Element number
	Material number
	Connectivities
	[Number of nodes per element]

Item 7: Stress information	**Number of lines:** `QUADRATURE.element.ngauss` ×`FEM.mesh.nelem`

`Stress(i,ig,ie)`
`[i=1:stress_components]`
`[ig=1:QUADRATURE.element.ngauss]`
`[ie=1:FEM.mesh.nelem]`
	Cauchy stress for each Gauss point
	(2-D) stress: $\sigma_{xx}, \sigma_{xy}, \sigma_{yy}$ (plane strain)
	(2-D) stress: $\sigma_{xx}, \sigma_{xy}, \sigma_{yy}, h$ (plane stress)
	(3-D) stress: N_x (truss)
	(3-D) stress: $\sigma_{xx}, \sigma_{xy}, \sigma_{xz}, \sigma_{yy}, \sigma_{yz}, \sigma_{zz}$

The output file produced by FLagSHyP for the simple example in Figure 10.1 is listed in Box 10.3.

Box 10.3: Output File for Example in Figure 10.1

```
2-D Example      at increment:      1,       load:  5
quad4
9
1 3  0.0000E+00  0.0000E+00 -3.3614E+00  9.5002E-01
2 2  1.1889E+00 -1.2500E-01  0.0000E+00 -2.1952E+00
3 3  2.1000E+00 -7.5000E-02 -1.2617E+00 -2.2113E+00
4 0  2.9056E-01  7.8088E-01  0.0000E+00 -2.4500E+00
5 0  1.2833E+00  1.0620E+00  0.0000E+00 -4.9000E+00
6 0  2.0531E+00  1.2262E+00  0.0000E+00 -2.4500E+00
7 0  5.0207E-02  1.6092E+00  0.0000E+00 -1.2250E+00
8 3  1.0000E+00  2.0000E+00 -3.8769E+00 -4.3497E-02
9 0  2.3964E+00  3.8249E+00  6.0000E+00  1.5775E+01
4
1 1 1 2 5 4
2 2 6 5 2 3
3 1 5 8 7 4
4 2 5 6 9 8
 3.1165E+01  1.6636E+01 -2.9752E+01  9.9858E-02
 3.7922E+01  7.0235E+00  2.9804E+01  9.2369E-02
 9.8170E+00  2.8948E+01  2.3227E+01  9.6515E-02
-9.1664E+00  5.2723E+01 -5.2341E+01  1.0566E-01
-3.1460E+01  9.0191E+00  6.9610E+01  9.7692E-02
-4.4255E+01  1.9009E+01  4.0029E+01  1.0422E-01
-1.0503E+01  1.4344E+01  5.8661E+01  9.4115E-02
-1.0937E+00  4.3534E+00  8.4855E+01  8.8759E-02
 2.9733E+00  4.9849E+00 -8.6633E+00  1.0056E-01
-2.5993E+00  1.0535E+01 -4.9380E+00  1.0075E-01
-1.0028E+01  1.6380E+01 -2.4223E+01  1.0326E-01
-3.7416E+00  1.0076E+01 -2.8318E+01  1.0306E-01
 1.8711E+01  2.7033E+01  1.2770E+02  8.0604E-02
 5.8710E+01  9.3889E+01  5.0464E+02  5.2100E-02
 1.4861E+02  2.3372E+02  7.0689E+02  3.9520E-02
 1.3288E+02  1.6687E+02  3.5422E+02  5.4008E-02
```

```
2-D Example      at increment:      2,       load:  10
quad 4
9
1 3  0.0000E+00  0.0000E+00 -6.0853E+00  2.5627E+00
2 2  1.3519E+00 -2.5000E-01  0.0000E+00 -3.9191E+00
3 3  2.2000E+00 -1.5000E-01 -2.4435E+00 -2.9205E+00
4 0  5.4010E-01  6.6991E-01  0.0000E+00 -4.9000E+00
5 0  1.5590E+00  1.1437E+00  0.0000E+00 -9.8000E+00
6 0  2.2245E+00  1.2882E+00  0.0000E+00 -4.9000E+00
7 0  1.9116E-01  1.3055E+00  0.0000E+00 -2.4500E+00
8 3  1.0000E+00  2.0000E+00 -8.4712E+00 -2.7232E+00
9 0  3.3987E+00  6.1513E+00  1.2000E+01  3.1550E+01
4
1 1 1 2 5 4
2 2 6 5 2 3
3 1 5 8 7 4
4 2 5 6 9 8
 6.2596E+01  2.1249E+01 -3.2758E+01  9.6870E-02
 6.1948E+01  9.8381E+00  5.4321E+01  8.5200E-02
 2.1019E+01  4.4812E+01  4.5486E+01  9.2526E-02
-1.5069E+01  1.0427E+02 -1.0393E+02  1.1028E-01
-5.0536E+01  1.8529E+01  1.0546E+02  1.0025E-01
-5.4947E+01  3.3161E+01  9.1325E+01  1.0362E-01
-1.1718E+01  3.2504E+01  1.1797E+02  8.9494E-02
-9.7154E+00  1.7872E+01  1.2969E+02  8.6976E-02
 2.1962E+01  8.2142E+00 -4.1974E+00  9.8174E-02
-2.0453E-02  1.3036E+01  7.8808E+00  9.9204E-02
-3.3571E+01  3.7568E+01 -3.3248E+01  1.0611E-01
-2.7830E+00  2.9571E+01 -4.8372E+01  1.0477E-01
 8.3822E+01  6.9453E+01  3.6196E+02  5.1329E-02
 1.6278E+02  4.2678E+02  1.7024E+03  2.9913E-02
 4.7504E+02  9.9613E+02  2.7018E+03  1.8205E-02
 4.1084E+02  6.3881E+02  1.3761E+03  2.4400E-02
```

10.4 Element types

Nodes and Gauss points in a given finite element can be numbered in a variety of ways. The numbering scheme chosen in FLagSHyP is shown in Figures 10.2 and 10.3.

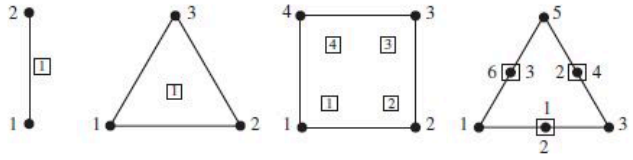


Figure 10.2 Numbering of two-dimensional elements.

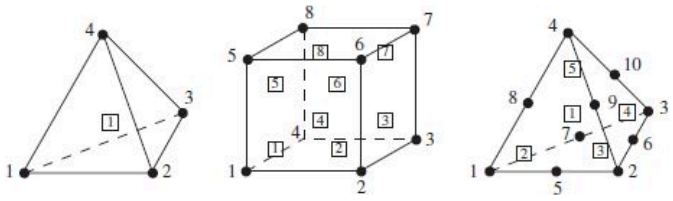


Figure 10.3 Numbering of three-dimensional elements.

In order to avoid the common repetitious use of shape functions for each mesh element, FLagSHyP (with the exception of `truss2`) stores in memory the shape

functions and their nondimensional derivatives for each Gauss point of the chosen element type. This information is stored in the two-dimensional array `FEM.interpolation.element.N` and the three-dimensional array `FEM.interpolation.element.DN_chi` used to store shape functions and derivatives, respectively. Specifically, the dimensions of `FEM.interpolation.element.N` are the number of nodes per element (`FEM.mesh.n_nodes_elem`) and the number of Gauss points per element (`QUADRATURE.element.ngauss`), which can be depicted as follows:

$$\begin{bmatrix} N_1(\xi_1,\eta_1,\zeta_1) & \cdots & N_1(\xi_m,\eta_m,\zeta_m) \\ \cdots & \cdots & \cdots \\ N_n(\xi_1,\eta_1,\zeta_1) & \cdots & N_n(\xi_m,\eta_m,\zeta_m) \end{bmatrix} \quad \begin{matrix} m = \texttt{QUADRATURE.element} \\ \texttt{.ngauss;} \\ n = \texttt{FEM.mesh.n\_nodes\_elem.} \end{matrix}$$

Analogously, the dimensions of `FEM.interpolation.element.DN_chi` are the number of dimensions (`GEOM.ndime`), the number of nodes per element (`FEM.mesh.n_nodes_elem`), and the number of Gauss points per element (`QUADRATURE.element.ngauss`):

$$\begin{bmatrix} \frac{\partial N_1}{\partial \xi}(\xi_i,\eta_i,\zeta_i) & \cdots & \frac{\partial N_n}{\partial \xi}(\xi_i,\eta_i,\zeta_i) \\ \frac{\partial N_1}{\partial \eta}(\xi_i,\eta_i,\zeta_i) & \cdots & \frac{\partial N_n}{\partial \eta}(\xi_i,\eta_i,\zeta_i) \\ \frac{\partial N_1}{\partial \zeta}(\xi_i,\eta_i,\zeta_i) & \cdots & \frac{\partial N_n}{\partial \zeta}(\xi_i,\eta_i,\zeta_i) \end{bmatrix} \quad \begin{matrix} i = 1 : \texttt{QUADRATURE.element} \\ \texttt{.ngauss;} \\ n = \texttt{FEM.mesh.n\_nodes\_elem.} \end{matrix}$$

The coordinates (in the nondimensional isoparametric domain) of all the quadrature points necessary for the accurate integration of a finite element are stored in the two-dimensional array `QUADRATURE.element.Chi`, of dimensions `GEOM.ndime` and `FEM.mesh.n_nodes_elem`, while their corresponding weights are stored in the one-dimensional array `QUADRATURE.element.W`, of dimension `FEM.mesh.n_nodes_elem`.

Exactly the same type of arrays are constructed for the line or surface elements and stored in `FEM.interpolation.boundary.N` (for shape functions), `FEM.interpolation.boundary.DN_chi` (for shape functions derivatives), and `QUADRATURE.boundary.W` and `QUADRATURE.boundary.Chi` (for Gauss point information).

10.5 Solver details

For the solution of the resulting system of linear algebraic equations $\mathbf{K}\mathbf{u} = -\mathbf{R}$, FLagSHyP uses the MATLAB command $\mathbf{u} = -\mathbf{K}\backslash\mathbf{R}$, where $\mathbf{K}$ denotes the assembled global stiffness matrix (`GLOBAL.K`) and $\mathbf{R}$ is the assembled global residual vector (`GLOBAL.RESIDUAL`). MATLAB will aim to take advantage of possible symmetries in the problem, redirecting to the most appropriate linear algebra solver, with the final

objective of minimizing the computational time while maintaining the accuracy of the final solution. Specifically, tailor-made solvers can be utilized for dealing with a sparse matrix $\mathbf{K}$ as in the present case.

In order to take advantage of the sparse solver capabilities, the final assembled stiffness matrix $\mathbf{K}_c + \mathbf{K}_\sigma + \mathbf{K}_\kappa - \mathbf{K}_p$ is stored in a temporary one-dimensional vector array (`global_stiffness`), along with two other temporary one-dimensional vector arrays (of the same length as `global_stiffness`), namely (`indexi`) and (`indexj`), where the rows (`indexi`) and columns (`indexj`) corresponding to the entries of the stiffness matrix are saved. By making use of the MATLAB intrinsic function `sparse.m`, these vectors are assembled into `GLOBAL.K`.

Finally, it is important to remark that the linear algebra solver is not restricted to the typical case of a symmetric stiffness matrix. In other words, it allows for the most general case of pressure loads which can lead to an unsymmetric stiffness matrix. This results in the need to store both lower and upper triangular entries of the assembled stiffness matrix $\mathbf{K}$. However, as stated above, if the symmetry of $\mathbf{K}$ is detected when the MATLAB command $\mathbf{Ku} = -\mathbf{R}$ is executed, a specific symmetric solver will then be employed.

Algorithm 1 shown below depicts in a simplified diagram the way in which the element tangent stiffness matrix is formed as a series of loops beginning with element and Gauss quadrature and cascading down to nodes and dimensions loops. Upon completion of these calculations, the sparse assembly process is carried out. Notice that this process is repeated for the assembly of each of the various components of the assembled global stiffness matrix, namely the constitutive matrix component $\mathbf{K}_c$ (refer to the function `constitutive_matrix.m`), the geometric or initial stress matrix component $\mathbf{K}_\sigma$ (refer to the function `geometric_matrix.m`), the volumetric mean dilatation stiffness component $\mathbf{K}_\kappa$ (refer to the function `volumetric_mean_dilatation_matrix.m`), and the external force pressure stiffness component $\mathbf{K}_p$ (refer to the function `pressure_load_matrix.m`).

Algorithm 1: Calculation and storage of assembled constitutive matrix entries

Input : `counter=0; indexi(:)=0; indexj(:)=0;`
`global_stiffness(:)=0`

Output : `GLOBAL.K`

```
for ielement=1:FEM.mesh.nelem do
    for igauss=1:QUADRATURE.element.ngauss do
        for anode=1:FEM.mesh.n_nodes_elem do
            for bnode=1:FEM.mesh.n_nodes_elem do
                for i=1:GEOM.ndime do
                    for j=1:GEOM.ndime do
                        Evaluate indexi(counter);
                        Evaluate indexj(counter);
                        Evaluate stiffness coefficient k;
                        Assign global_stiffness(counter) ← k;
                        counter ← counter + 1;
                    end
                end
            end
        end
    end
end

GLOBAL.K=sparse(indexi,indexj,global_stiffness)
```

To illustrate the assembly of the global stiffness matrix, consider the somewhat artificial example in Figure 10.4, comprising three triangular elements with one degree of freedom per node and two Gauss points per element. The figure shows the process by which a stiffness coefficient k per Gauss point, shown in Algorithm 1, is assigned to the appropriate location in the `global_stiffness` vector in a sequence determined by `counter`. Also observe in Figure 10.4 how the MATLAB function `sparse.m` assembles a typical global stiffness entry from its Gauss point contributions.

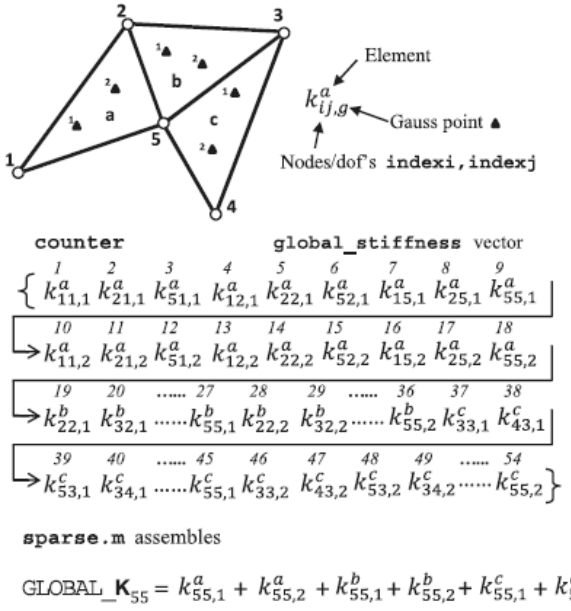


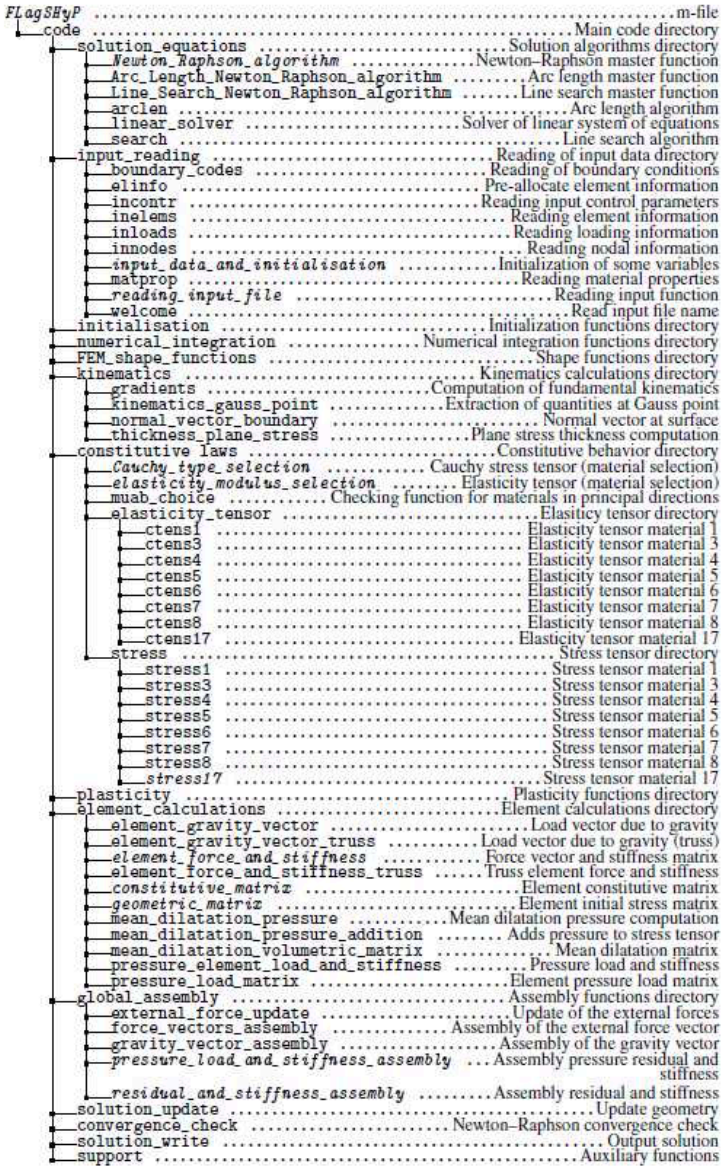
Figure 10.4 Stiffness matrix assembly; `counter` refers to the location of the stiffness

coefficient k in the `global stiffness vector`.

10.6 Program structure

In order to give an overview of the structure of the program FLagSHyP, Box 10.4 lists the main functions comprising the program. The remainder are either functions common to standard linear elasticity finite element codes or subsidiary functions that are not crucial to an understanding of the flow of the program and can be examined via a program download (see www.flagshyp.com). The functions in italic typeface are described in detail in the following sections.

Box 10.4: FLagSHyP Structure



10.7 Master m-file `FLagSHyP`

As stated at the beginning of this chapter, the master m-file `FLagSHyP.m` is divided into three sections, presented below.

FLagSHyP segment 1 – Master m-file

```
clear all;close all;clc;

%%SECTION 1

if isunix()
   dirsep =  '/';
else
   dirsep =  '\';
end
basedir_fem = mfilename('fullpath');
basedir_fem = fileparts(basedir_fem);
basedir_fem = strrep(basedir_fem,['code' dirsep 'support'],");
addpath(fullfile(basedir_fem));
addpath(genpath(fullfile(basedir_fem,'code')));
addpath((fullfile(basedir_fem,'job_folder')));

%%SECTION 2

[PRO,FEM,GEOM,QUADRATURE,BC,MAT,LOAD,CON,CONSTANT,...
 GLOBAL,PLAST,KINEMATICS] = input_data_and_initialisation(basedir_fem);


%%SECTION 3

if abs(CON.ARCLEN.arcln)==0
   if ~CON.searc
      Newton_Raphson_algorithm(PRO,FEM,GEOM,QUADRATURE,BC,MAT,LOAD,...
      CON,CONSTANT,GLOBAL,PLAST,KINEMATICS);
   else
      Line_Search_Newton_Raphson_algorithm(PRO,FEM,GEOM,QUADRATURE,...
      BC,MAT,LOAD,CON,CONSTANT,GLOBAL,PLAST,KINEMATICS);
   end
else
      Arc_Length_Newton_Raphson_algorithm(PRO,FEM,GEOM,QUADRATURE,...
      BC,MAT,LOAD,CON,CONSTANT,GLOBAL,PLAST,KINEMATICS);
end
```

The first section sets up the directory path, which has been previously explained in the user instructions. The second section is the function `input_data_and_initialisation.m`, for the reading of input/control data and the initialization of variables including the initial assembled tangent stiffness and equivalent nodal forces. At this stage, it is appropriate to get an overview of the third section before dealing in detail with the input data and initialization function. Depending on the values chosen by the user for some of the problem-dependent control variables, this redirects the program to one of the three fundamental algorithms incorporated into FLagSHyP, namely `ALG1`, `ALG2`, and `ALG3`.

The three `ALG` MATLAB functions share many sub-functions, but `ALG1` constitutes the majority of the program. As a result, we will focus on the description of this basic algorithm in order to give a general overview of a standard large strain nonlinear finite element computer program. Once this function has been fully understood, it is expected

that readers can then proceed to study the other two functions (with the help of Sections 9.6.2 and 9.6.3).

Within the function `Newton_Raphson_algorithm.m`, it is essential to understand one crucial function, namely, `residual_and_stiffness_assembly.m`, which assembles the residual force (`GLOBAL.Residual`) and the global tangent stiffness matrix (`GLOBAL.K`). This requires the computation of the equivalent nodal forces due to internal stress and the main components of the stiffness matrix for each element, which is carried out in `element_force_and_stiffness.m`. Consequently, it provides a vehicle for examining those aspects of the computation that are particular to finite deformation analysis. In addition, the function `pressure_load_and_stiffness_assembly.m` evaluates the equivalent nodal forces due to surface pressure and the corresponding tangent matrix components. Additional functions included are `linear_solver.m`, which calls the linear solver for the computation of the unknown displacements **u**, `update_geometry.m`, which updates the geometry of the deformed solid, and `check_residual_norm.m`, which verifies whether convergence has been achieved.

Within the function `residual_and_stiffness_assembly.m` mentioned above, it is important to distinguish between truss and continuum elements. Specifically, for truss elements the function `element_force_and_stiffness_truss.m` evaluates the equivalent nodal forces and the components of the tangent stiffness matrix. This function is not described as it is transparent in its replication of the equations in the text. For any other kind of finite element, the counterpart function `element_force_and_stiffness.m` evaluates the same quantities. This latter function is more complex, hence lengthier, as it is not restricted to simplified truss kinematics and it allows for the consideration of a large variety of kinematics hypotheses, for example, near incompressibility, analysis along principal directions, or plane stress.

The function `element_force_and_stiffness.m` calls the utility functions `Cauchy_type_selection.m` and `elasticity_modulus_selection.m` to compute the stress tensor and tangent modulus, respectively, depending upon the material type selection, either a pure hyperelastic or an elasto-plastic material. The remaining functions in the program are either relatively similar to standard finite element elasticity codes or are a direct implementation of equations contained in the book.

Returning to a detailed consideration of the second section of FLagSHyP, this is presented below.

FLagSHyP segment 2 – Input data and initialization

```
function [PRO,FEM,GEOM,QUADRATURE,BC,MAT,LOAD,CON,CONSTANT,GLOBAL,...
          PLAST,KINEMATICS] = input_data_and_initialisation(basedir_fem)

PRO = welcome(basedir_fem);
if ~PRO.rest
  fid = PRO.fid_input;
  [FEM,GEOM,QUADRATURE,BC,MAT,LOAD,CON,PRO,...  GLOBAL]=
    reading_input_file(PRO,fid);

  CONSTANT = constant_entities(GEOM.ndime);
```

```
  CON.xlamb = 0;
  CON.incrm = 0;
  [GEOM,LOAD,GLOBAL,PLAST,KINEMATICS] = initialisation(FEM,GEOM,...
  QUADRATURE,MAT,LOAD,CONSTANT,CON,GLOBAL);
  cd(PRO.job_folder)
  save_restart_file(PRO,FEM,GEOM,QUADRATURE,BC,MAT,LOAD,CON,CONSTANT,...
           GLOBAL,PLAST,KINEMATICS,'internal')
```

The program can start either from an input data file or, when convenient, using data from a restart file written during a previous incomplete analysis. The mode of operation is recorded in the variable `PRO.rest` read by the `welcome.m` function. Function `initialisation.m` sets global residual vector `GLOBAL.Residual` to zero and defines the initial geometry `GEOM.x0` as the current geometry `GEOM.x`. In addition, the equivalent nodal forces due to gravity are evaluated and the function `residual_and_stiffness_assembly.m` is called to compute the global tangent stiffness matrix at the unstressed configuration (this coincides with the small strain linear elastic stiffness matrix). The function `initial_volume.m` evaluates the total mesh volume and prints this on the screen. This is particularly useful in order to check the validity of the geometry given by the input file but also fundamental in the case of dealing with a nearly incompressible material, where the initial elemental volume is necessary for the mean dilatation algorithm. All the relevant information is dumped to a restart file.

FLagSHyP segment 3 – Restart

```
else
  cd(PRO.job_folder);
  load(PRO.restartfile_name);
  [GLOBAL,PLAST] = residual_and_stiffness_assembly(CON.xlamb,GEOM,...
  MAT,FEM,GLOBAL,CONSTANT,QUADRATURE.element,PLAST,KINEMATICS);
end
```

The program can restart from a previously converged incremental step in order to progress to subsequent increments. This can be useful when wishing to continue the analysis to obtain further incremental solutions. In the following it is assumed that the data are read for the first time.

FLagSHyP segment 4 – Reading Input data

```
function [FEM,GEOM,QUADRATURE,BC,MAT,LOAD,CON,PRO,GLOBAL] =...
         reading_input_file(PRO,fid)

print_statement('beginning_reading')
PRO.title = strtrim(fgets(fid));
[FEM,GEOM,QUADRATURE] = elinfo(fid);

switch FEM.mesh.element_type
  case 'truss2'
    FEM.interpolation.element = [];
    FEM.interpolation.boundary = [];
  otherwise
    QUADRATURE.element = element_quadrature_rules(FEM.mesh.element_type)
    ;
```

```
    QUADRATURE.boundary = edge_quadrature_rules(FEM.mesh.element_type);
    FEM = shape_functions_iso_derivs(QUADRATURE,FEM,GEOM.ndime);
end

[GEOM,BC,FEM] = innodes(GEOM,fid,FEM);
[FEM,MAT] = inelems(FEM,fid);
BC = find_fixed_free_dofs(GEOM,FEM,BC);
MAT = matprop(MAT,FEM,fid);
[LOAD,BC,FEM,GLOBAL] = inloads(GEOM,FEM,BC,fid);
CON = incontr(BC,fid);

fclose('all');
print_statement('end_reading')
```

Function `elinfo.m` reads the element type `FEM.mesh.element_type` and establishes the basic information regarding the specific type of finite element to be used. Function `innodes.m` reads nodal input data, namely initial coordinates (`GEOM.x`) and boundary conditions (`BC.icode`). Function `inelems.m` reads relevant element information, namely connectivity (`FEM.mesh.connectivity`) and material type (`MAT.matno`). Function `find_fixed_free_dofs.m` classifies the degrees of freedom in fixed and free, based on the nodal boundary condition information. Function `matprop.m` reads in material property data (`MAT.props`). Function `inloads.m` inputs loading and prescribed displacement items. Finally, function `incontr.m` reads the solution control parameters (`CON`).

FLagSHyP segment 5 – Increment loop

```
function Newton_Raphson_algorithm(PRO,FEM,GEOM,QUADRATURE,BC,MAT,...
LOAD,CON,CONSTANT,GLOBAL,PLAST,KINEMATICS)

while ((CON.xlamb < CON.xlmax) && (CON.incrm < CON.nincr))
  CON.incrm = CON.incrm + 1;
  CON.xlamb = CON.xlamb + CON.dlamb;

  [GLOBAL.Residual,GLOBAL.external_load] = ...
  external_force_update(GLOBAL.nominal_external_load,...
  GLOBAL.Residual,GLOBAL.external_load,CON.dlamb);

  if LOAD.n_pressure_loads
     GLOBAL = pressure_load_and_stiffness_assembly(GEOM,MAT,...
     FEM,GLOBAL,LOAD,QUADRATURE.boundary,CON.dlamb);
  end

  if  BC.n_prescribed_displacements > 0
     GEOM.x = update_prescribed_displacements(BC.dofprescribed,...
     GEOM.x0,GEOM.x,CON.xlamb,BC.presc_displacement);
     [GLOBAL,updated_PLAST] = ...
     residual_and_stiffness_assembly(CON.xlamb,...
     GEOM,MAT,FEM,GLOBAL,CONSTANT,QUADRATURE.element,PLAST,KINEMATICS);

     if LOAD.n_pressure_loads
        GLOBAL = pressure_load_and_stiffness_assembly(GEOM,...
        MAT,FEM,GLOBAL,LOAD,QUADRATURE.boundary,CON.xlamb);
     end

end
```

The `while` controls the load or prescribed displacement incrementation. This remains active while the load-scaling parameter `CON.xlamb` is less than the maximum `CON.xlmax` and the increment number is smaller than the total number of increments `CON.nincr`.

The imposition of an increment of point or gravity loads immediately creates a residual force. This is added to any small residual carried over from the previous increment by the function `external_force_update.m`. This prevents errors in the converged solution (for instance, due to a large tolerance value `CON.cnorm`) from propagating throughout the solution process. The further addition to the residual forces due to an increment in applied surface pressure is evaluated by calling `pressure_load_and_stiffness_assembly.m` with the last argument set to the load parameter increment `CON.dlamb`. At the same time `pressure_load_and_stiffness_assembly.m` evaluates the addition to the tangent matrix caused by possible geometrical changes to the pressure surface and the increment in the magnitude of the pressure.

If prescribed displacements have been imposed, then the current increment in their value will immediately change the current geometry and necessitate a complete re-evaluation of equivalent internal and surface pressure forces and the tangent matrix (this is effectively a geometry update). In this case, function `update_prescribed_displacements.m` will reset the current geometry for those nodes with prescribed displacements based on their initial coordinates, the nominal value of the prescribed displacement, and the current load-scaling parameter `CON.xlamb`. Function `residual_and_stiffness_assembly.m` recomputes the residual vector and the global tangent stiffness matrix due to the new updated configuration. Subsequently, function `pressure_load_and_stiffness_assembly.m` is now called with the last argument set to `CON.xlamb`, to add the total value of the nodal forces due to surface pressure and obtain the corresponding initial surface pressure tangent matrix component.

FLagSHyP segment 6 – Newton–Raphson loop

```
CON.niter = 0;
rnorm = 2*CON.cnorm;
while((rnorm > CON.cnorm) && (CON.niter < CON.miter))
   CON.niter = CON.niter + 1;
   displ = linear_solver(GLOBAL.K,-GLOBAL.Residual,BC.fixdof);
```

The `while` loop controls the Newton–Raphson iteration process. This continues until the residual norm `rnorm` is smaller than the tolerance `CON.cnorm` and, of course, while the iteration number `CON.niter` is smaller than the maximum allowed `CON.miter`. Note that `rnorm` is initialized (`rnorm = 2*CON.cnorm`) in such a way that this loop is completed at least once. The function `linear_solver.m` solves the linear system of equations to obtain the incremental displacements **u**.

FLagSHyP segment 7 – Solution update and equilibrium check loop

```
GEOM.x = update_geometry(GEOM.x,1,displ,BC.freedof);
```

```
    [GLOBAL,updated_PLAST] = ...
    residual_and_stiffness_assembly(CON.xlamb,GEOM,MAT,...
    FEM,GLOBAL,CONSTANT,QUADRATURE.element,PLAST,KINEMATICS);

    if LOAD.n_pressure_loads
        GLOBAL = pressure_load_and_stiffness_assembly(GEOM,MAT,...
        FEM,GLOBAL,LOAD,QUADRATURE.boundary,CON.xlamb);

    end

    [rnorm,GLOBAL] = check_residual_norm(CON,BC,GLOBAL,BC.freedof);
    if (abs(rnorm)>1e7 || isnan(rnorm))
        CON.niter=CON.miter;
        break;
    end
end
```

The function `update_geometry.m` adds the displacements evaluated in segment 6 to the current nodal positions `GEOM.x`. Given this new geometry, the two functions `residual_and_stiffness_assembly.m` and `pressure_load_and_stiffness_assembly.m` serve the same purpose as described in segment 5. Finally, `checkr_residual_norm.m` computes the residual norm `rnorm` given the residual force vector `GLOBAL.Residual` and the total external force vector (including pressure loading).

FLagSHyP segment 8 – Retry and output results

```
    if( CON.niter >= CON.miter)
      load(PRO.internal_restartfile_name)
      fprintf('Solution not converged. Restart from previous...
      step by decreasing the load parameter increment to ...
      half its initally fixed value. {\mibf n}');
      CON.dlamb = CON.dlamb/2;
    else
      PLAST = save_output(updated_PLAST,PRO,FEM,GEOM,QUADRATURE,...
      BC,MAT,LOAD,CON,CONSTANT,GLOBAL,PLAST,KINEMATICS);
    end
end
```

Function `save_output.m` writes out the output file described in Section 10.3. This occurs at every `CON.OUTPUT.incout` increment and is stored in the relevant `example` folder within `job_folder` with a unique identifier. In addition, the function utilizes the MATLAB function `save.m` to dump all variables within the existing workspace. This is continually refreshed to be available for an automatic restart in the event that the Newton–Raphson iteration does not converge, in which case the program will reduce the load increment by half (`CON.dlamb←CON.dlamb/2`).

10.8 Function `residual_and_stiffness_assembly`

This function assembles the residual force vector and the main components of the global tangent stiffness matrix for all elements other than those surface (line) elements subjected to pressure load, which are considered in Section 10.11.

to the current nodal positions `GEOM.x`. Given this new geometry, the two functions `residual_and_stiffness_assembly.m` and `pressure_load_and_stiffness_assembly.m` serve the same purpose as described in segment 5. Finally, `checkr_residual_norm.m` computes the residual norm `rnorm` given the residual force vector `GLOBAL.Residual` and the total external force vector (including pressure loading).

FLagSHyP segment 8 – Retry and output results

```
    if( CON.niter >= CON.miter)
      load(PRO.internal_restartfile_name)
      fprintf('Solution not converged. Restart from previous...
      step by decreasing the load parameter increment to ...
      half its initally fixed value. {\mibf n}');
      CON.dlamb = CON.dlamb/2;
    else
      PLAST = save_output(updated_PLAST,PRO,FEM,GEOM,QUADRATURE,...
      BC,MAT,LOAD,CON,CONSTANT,GLOBAL,PLAST,KINEMATICS);
    end
end
```

Function `save_output.m` writes out the output file described in Section 10.3. This occurs at every `CON.OUTPUT.incout` increment and is stored in the relevant `example` folder within `job_folder` with a unique identifier. In addition, the function utilizes the MATLAB function `save.m` to dump all variables within the existing workspace. This is continually refreshed to be available for an automatic restart in the event that the Newton–Raphson iteration does not converge, in which case the program will reduce the load increment by half (`CON.dlamb`←`CON.dlamb/2`).

10.8 Function `residual_and_stiffness_assembly`

This function assembles the residual force vector and the main components of the global tangent stiffness matrix for all elements other than those surface (line) elements subjected to pressure load, which are considered in Section 10.11.

`residual_and_stiffness_assembly` **part 1 – Initialization**

```
function [GLOBAL,updated_PLAST] = residual_and_stiffness_assembly(xlamb,
...
GEOM,MAT,FEM,GLOBAL,CONSTANT,QUADRATURE,PLAST,KINEMATICS)

updated_PLAST = PLAST;
GLOBAL.external_load = xlamb*GLOBAL.nominal_external_load;

switch FEM.mesh.element_type
  case 'truss2'
    n_components = FEM.mesh.nelem*FEM.mesh.n_dofs_elem^2;
  otherwise
    n_components_mean_dilatation =...
    MAT.n_nearly_incompressible*(FEM.mesh.n_dofs_elem^2);
    n_components_displacement_based = ...
    FEM.mesh.nelem*(FEM.mesh.n_dofs_elem^2+...
    (FEM.mesh.n_nodes_elem^2*GEOM.ndime))*QUADRATURE.ngauss;
```

```
    n_components = n_components_mean_dilatation +...
    n_components_displacement_based;
end

counter = 1;
indexi = zeros(n_components,1);
indexj = zeros(n_components,1);
global_stiffness = zeros(n_components,1);
GLOBAL.T_int = zeros(FEM.mesh.n_dofs,1);
```

The segment above initializes the dimension of the vectors `indexi` and `indexj` required for the allocation of the stiffness terms in the vector `global_stiffness`. In addition, the size of the vector `GLOBAL.T_int` is initialized based upon the total number of degrees of freedom `FEM.mesh.n_dofs`.

`residual_and_stiffness_assembly` **part 2 – Element loop**

```
for ielement=1:FEM.mesh.nelem
    global_nodes = FEM.mesh.connectivity(:,ielement);
    material_number = MAT.matno(ielement);
    matyp = MAT.matyp(material_number);
    properties = MAT.props(:,material_number);
    xlocal = GEOM.x(:,global_nodes);
    x0local = GEOM.x0(:,global_nodes);
    Ve = GEOM.Ve(ielement);
    PLAST_element = selecting_internal_variables_element(PLAST,...
    matyp,ielement);

    switch FEM.mesh.element_type
      case 'truss2'
          [T_internal,indexi,indexj,global_stiffness,counter,...
          PLAST_element] = ...
          element_force_and_stiffness_truss(properties,...
          xlocal,x0local,global_nodes,FEM,PLAST_element,...
          counter,indexi,indexj,global_stiffness,GEOM);
      otherwise
          [T_internal,indexi,indexj,global_stiffness,counter,...
          PLAST_element] = ...
          element_force_and_stiffness(FEM,xlocal,x0local,...
          global_nodes,Ve,QUADRATURE,properties,CONSTANT,...
          GEOM.ndime,matyp,PLAST_element,counter,KINEMATICS,...
          indexi,indexj,global_stiffness);
  end

  GLOBAL.T_int = force_vectors_assembly(T_internal,global_nodes,...
  GLOBAL.T_int,FEM.mesh.dof_nodes);

  updated_PLAST = plasticity_storage(PLAST_element,...
  updated_PLAST,matyp,ielement);
end

GLOBAL.K = sparse(indexi,indexj,global_stiffness);
GLOBAL.Residual = GLOBAL.T_int - GLOBAL.external_load;
```

For all element types the function `residual_and_stiffness_assembly.m` loops over all the elements. For each element other than `truss2`, the function `element_force_and_stiffness.m` is called to obtain the element equivalent force

vector `T_internal` and the element contribution to the global stiffness matrix `global_stiffness`.

Within the element loop, `force_vectors_assembly.m` assembles the element equivalent force `T_internal` into the global equivalent force vector `GLOBAL.T_int`. Once the loop over elements ends, the element stiffness contributions in `global_stiffness` are assembled into the global stiffness matrix `GLOBAL.K`, employing the MATLAB intrinsic function `sparse.m`. In addition, the global residual vector `GLOBAL.Residual` is obtained after subtracting the equivalent external force vector (`GLOBAL.external_load`) from the internal force vector (`GLOBAL.T_int`). When element type is `truss2` the function `element_force_and_stiffness_truss.m` is used for the calculation of the equivalent nodal forces and tangent matrix. The coding is not discussed herein as it is self-explanatory.

The `element_force_and_stiffness.m` function called in part 2 above will now be considered in detail.

`element_force_and_stiffness` **part 1 – Prior to Gauss point loop**

```
function [T_internal,indexi,indexj,global_stiffness,counter,...
          PLAST_element] = element_force_and_stiffness(FEM,xlocal,...
          x0local,element_connectivity,Ve,QUADRATURE,properties,...
          CONSTANT,dim,matyp,PLAST,counter,KINEMATICS,indexi,...
          indexj,global_stiffness)

T_internal = zeros(FEM.mesh.n_dofs_elem,1);
KINEMATICS = gradients(xlocal,x0local,FEM.interpolation.element.DN_chi,.
..
QUADRATURE,KINEMATICS);

switch matyp
   case {5,7,17}
        [pressure,kappa_bar,DN_x_mean,ve] = ...
        mean_dilatation_pressure(FEM,dim,matyp,properties,...
        Ve,QUADRATURE,KINEMATICS);
   otherwise
        pressure = 0;
end
```

Here the element internal equivalent force vector `T_internal` is initialized based upon the number of element degrees of freedom `FEM.mesh.n_dofs_elem`. The function `gradients.m` computes all the kinematics quantities associated with a finite element, such as $\boldsymbol{F}$ and $\boldsymbol{b} = \boldsymbol{F}\boldsymbol{F}^T$ and the pointwise volume ratio $J = \det \boldsymbol{F}$ (or area ratio in the case of plane stress). For convenience, these quantities are computed at all Gauss points of the element and stored in the variables `KINEMATICS.F`, `KINEMATICS.b`, and `KINEMATICS.J`. Other elemental quantities of interest computed in `gradients.m` include the current Cartesian derivatives $\partial N_a/\partial \boldsymbol{x}$ of the shape functions (refer to Equation (9.11a,b)), which are stored in `KINEMATICS.DN_x`, and the weighted Jacobian per Gauss point, which is stored in `KINEMATICS.Jx_Chi`. Using the left Cauchy–Green tensor $\boldsymbol{b}$, it is then straightforward to obtain the Cauchy stress and elasticity tensor for each of the seven material types implemented.

For nearly incompressible materials 5 and 7 and the elasto-plastic material 17, the mean dilatation technique described in Chapters 8 and 9 is used to obtain the internal element pressure p (`pressure`) from the volume ratio $\bar{J}$ (`Jbar` in the program) using Equation (8.53a). This is implemented in function `mean_dilatation_pressure.m`.

Referring to Algorithm 1, we now consider the Gauss integration loop.

`element_force_and_stiffness` **part 2 – Gauss point loop**

```
for igauss=1:QUADRATURE.ngauss
   kinematics_gauss = kinematics_gauss_point(KINEMATICS,igauss);

   [Cauchy,PLAST,plast_gauss] = Cauchy_type_selection(kinematics_gauss,.
   ..
   properties,CONSTANT,dim,matyp,PLAST,igauss);

   c = elasticity_modulus_selection(kinematics_gauss,...
   properties,CONSTANT,dim,matyp,PLAST,plast_gauss,igauss);

   [Cauchy,c] = pressure_addition(Cauchy,c,CONSTANT,pressure,matyp);

   th = thickness_plane_stress(properties,kinematics_gauss.J,matyp);

   JW = kinematics_gauss.Jx_chi*QUADRATURE.W(igauss)*th;

   T = Cauchy*kinematics_gauss.DN_x;

   T_internal = T_internal + T(:)*JW;

   [indexi,indexj,global_stiffness,counter] = constitutive_matrix(FEM,..
   .
   dim,element_connectivity,kinematics_gauss,c,JW,counter,indexi,...
   indexj,global_stiffness);

   DN_sigma_DN = kinematics_gauss.DN_x'*(Cauchy*kinematics_gauss.DN_x);
   [indexi,indexj,global_stiffness,counter] = geometric_matrix(FEM,...
   dim,element_connectivity,DN_sigma_DN,JW,counter,indexi,...
   indexj,global_stiffness);

end
switch matyp
   case {5,7,17}
        [indexi,indexj,global_stiffness,counter] = ...
        volumetric_mean_dilatation_matrix(FEM,dim,...
        element_connectivity,DN_x_mean,counter,...
        indexi,indexj,global_stiffness,kappa_bar,ve);
end
PLAST_element = PLAST;
```

For each Gauss point, and once the relevant kinematics quantities have been extracted, the function `Cauchy_type_selection.m` evaluates the Cauchy stress tensor as a function of the left Cauchy–Green tensor and the properties of the material of the element `ielem` under consideration. Within this function, depending on the value of the variable `MAT.matyp(MAT.matno(ielem))`, a specific stress function is called. Analogously, the function `elasticity_modulus_selection.m` evaluates the elasticity tensor as a function of the left Cauchy–Green tensor and the properties of the material of the element `ielem`.

For materials 5, 7, and 17, the mean dilatation procedure applies and the pressure evaluated in function `mean_dilatation_pressure.m` is now added to the deviatoric Cauchy stress tensor in the function `pressure_addition.m`. Recalling that we remain within the Gauss loop, the Cauchy stress components and the Cartesian derivatives of the shape functions are used to compute and assemble the equivalent nodal forces `T` employing Equations (9.15a–c). Function `constitutive_matrix.m` then evaluates the constitutive component of the tangent matrix according to the indicial Equation (9.35). The initial stress matrix is obtained and assembled in `geometric_matrix.m` using Equation (9.44c). This function will be described in detail in Section 10.10.

In addition, and once the Gauss integration loop is completed, the dilatational component $\mathbf{K}_\kappa$ of the tangent matrix is computed at the element level and assembled in function `mean_dilatation_volumetric_matrix.m` using Equation (9.59a,b).

`Cauchy_type_selection` **– Stress tensor evaluation**

```
function [Cauchy,PLAST,PLAST_gauss] = Cauchy_type_selection(kinematics,.
         ..
         properties,cons,dim,matyp,PLAST,igauss)

PLAST_gauss = [];
switch matyp
   case 1
        Cauchy = stress1(kinematics,properties,cons);
   case 3
        Cauchy = stress3(kinematics,properties,dim);
   case 4
        Cauchy = stress4(kinematics,properties,dim);
   case 5
        Cauchy = stress5(kinematics,properties,cons,dim);
   case 6
        Cauchy = stress6(kinematics,properties,cons);
   case 7
        Cauchy = stress7(kinematics,properties,dim);
   case 8
        Cauchy = stress8(kinematics,properties,dim);
   case 17
        PLAST_gauss.OLD.invCp = PLAST.invCp(:,:,igauss);
        PLAST_gauss.OLD.epbar = PLAST.epbar(igauss);
        [Cauchy,PLAST_gauss] = stress17(kinematics,...
        properties,dim,PLAST_gauss);
        PLAST = plasticity_update(PLAST_gauss.UPDATED,...
        PLAST,igauss,matyp);
end
```

Depending on the value of the variable `MAT.matyp(MAT.matno(ielem))`, the function `Cauchy_type_selection.m` selects a specific stress tensor function.

`elasticity_modulus_selection` **– Elasticity tensor evaluation**

```
function c_tensor = elasticity_modulus_selection(kinematics,...
         properties,cons,dimension,matyp,PLAST,plast_gauss,igauss)
```

```
c_tensor = [];
switch matyp
   case 1
        c_tensor = ctens1(kinematics,properties,cons);
   case 3
        c_tensor = ctens3(kinematics,properties,dimension);
   case 4
        c_tensor = ctens4(kinematics,properties,dimension);
   case 5
        c_tensor = ctens5(kinematics,properties,cons,dimension);
   case 6
        c_tensor = ctens6(kinematics,properties,cons);
   case 7
        c_tensor = ctens7(kinematics,properties,dimension);
   case 8
        c_tensor = ctens8(kinematics,properties,dimension);
   case 17
        plast_gauss.OLD.invCp = PLAST.invCp(:,:,igauss);
        plast_gauss.OLD.epbar = PLAST.epbar(igauss);
        c_tensor = ctens17(kinematics,properties,dimension,plast_gauss);
end
```

Similarly, depending on the value of `MAT.matyp(MAT.matno(ielem))`, the function `elasticity_modulus_selection.m` selects a specific elasticity tensor function.

It is now worthwhile discussing some aspects of the implementation of the stress tensor and elasticity tensor for the various materials. In particular, the stress calculations for the elasto-plastic material 17 will be considered in more detail. Recall that these calculations are particular to each Gauss point; for example, within an element, one Gauss point could be elastic while another could have become plastic.

Material 1. For a compressible neo-Hookean material, the functions `stress1.m` and `ctens1.m` are described in Section 6.4.3. The Cauchy stresses are evaluated in `stress1.m` in terms of the $\boldsymbol{b}$ tensor from Equation (6.29), whereas the coefficients of the elasticity tensor are obtained in `ctens1.m` using Equation (6.40).

Material 3. For a hyperelastic material in principal directions, the principal directions $\boldsymbol{n}_\alpha$ and principal stretches λ_α are computed by the function `gradients.m` and stored in `KINEMATICS.n` and `KINEMATICS.lambda`, respectively. Given these stretches and directions, the function `stress3.m` evaluates the Cartesian components of the Cauchy stress tensor with the help of Equations (6.81) and (6.94). Finally, the function `ctens3.m` uses Equations (6.90), (6.91), and (6.95) to compute the corresponding elasticity tensor.

Material 4. A plane stress hyperelastic material in principal directions is described in Section 6.6.7 and is implemented in a similar way to the previous material, the main difference being in the parameter γ, which is obtained in `stress4.m` and `ctens4.m`. For this material, the current thickness, computed in the function `thickness_plane_stress.m`, is updated in terms of the volume ratio J, the area ratio j, and the initial thickness stored in `MAT.props(4,im)` (see Exercise 4 of Chapter 4). The current thickness is stored as the fourth stress for output purposes.

Material 5. A nearly incompressible neo-Hookean material is discussed in Sections 6.5.2 and 6.5.3. The deviatoric Cauchy stresses are evaluated in `stress5.m` using Equation (6.55a,b). Note that in the mean dilatation method the pressure is constant over the element and therefore evaluated in segment 2 outside the Gauss point loop. This pressure value is added to the deviatoric components in `pressure_addition.m`. Analogously, function `ctens5.m` obtains the deviatoric component of the elasticity tensor and `pressure_addition.m` adds the volumetric component (refer to Equations (6.59a,b)).

Material 6. The plane stress incompressible neo-Hookean material is discussed in Exercise 2 of Chapter 6 and implemented in functions `stress6.m` and `ctens6.m`. Initially, the effective shear coefficient μ' is evaluated, dividing μ by $j^2 = \det_{2\times 2} \boldsymbol{b}$. As shown in Exercise 2 of Chapter 6, an effective lambda coefficient (see Equation (6.40)) emerges as twice the effective shear coefficient. The thickness is finally computed as in material 4, with the exception that due to incompressibility J is equal to 1.

Material 7. The nearly incompressible material in principal directions is discussed in Section 6.6.6. Stretches and principal directions are first evaluated in `gradients.m`. Function `stress7.m` implements Equation (6.94) with λ being set to $-2\mu/3$ to give the deviatoric Cauchy stresses in accordance with Equation (6.107). The internal hydrostatic pressure is subsequently added in `pressure_addition.m`. The deviatoric components of the elasticity tensor are obtained using the function `ctens7.m`. Finally, the volumetric component is obtained as per material 5.

Material 8. This material, implemented in functions `stress8.m` and `ctens8.m`, is identical to material 4, but because of incompressibility $\lambda \to \infty$ and hence $\gamma = 0$, $J = 1$, and $\bar{\lambda} = 2\mu$.

Material 17. This material is presented below in two parts. Part 1 evaluates the yield function and part 2 considers the consequences of this calculation, namely elastic or plastic behavior at the Gauss point.

`stress17` **part 1 – Cauchy stress and elasticity tensors**

```
function [Cauchy,PLAST] = stress17(kinematics,properties,dim,PLAST)

invCp = PLAST.OLD.invCp;
ep = PLAST.OLD.epbar;
mu = properties(2);
DIM = dim;
J = kinematics.J;
F(1:dim,1:dim) = kinematics.F;

be_trial(1:dim,1:dim) = F*(invCp*F');
[V,D] = eig(be_trial);
lambdae_trial = sqrt(diag(D));
na_trial = V;

tauaa_trial = (2*mu)*log(lambdae_trial)- (2*mu/3)*log(J);
switch dim
   case 2
        DIM = 3;
        lambdae_trial(3) = 1/det(invCp);
        tauaa_trial = (2*mu)*log(lambdae_trial)- (2*mu/3)*log(J);
```

```
        na_trial = [[na_trial(:,1);0] [na_trial(:,2);0] [0;0;1]];
        be_trial(3,3) = lambdae_trial(3);
end
tau_trial = zeros(DIM);
for idim=1:DIM
   tau_trial = tau_trial + ...
   tauaa_trial(idim)*(na_trial(:,idim)*na_trial(:,idim)');
end

f = Von_Mises_yield_function(tau_trial,ep,properties(5),properties(6));
```

In part 1 above, the trial left Cauchy–Green tensor, given by Equation (7.41), is first evaluated in order to calculate the associated principal directions and stretches. Subsequently, the trial principal (see Equation (7.43a,b)) Cartesian deviatoric Kirchhoff stresses are computed to enable the yield function to be evaluated in accordance with Equation (7.20a,b).

`stress17` **part 2 – Cauchy stress and elasticity tensors**

```
if f>0
  [Dgamma,nu_a] = radial_return_algorithm(f,tau_trial,...
  tauaa_trial,mu,properties(6));
  lambdae = exp(log(lambdae_trial) - Dgamma*nu_a);
  norm_tauaa_trial = norm(tauaa_trial,'fro');
  tauaa = (1-2*mu*Dgamma/(sqrt(2/3)*norm_tauaa_trial))*tauaa_trial;
  tau = zeros(dim);
  be = zeros(dim);
  for idim=1:DIM
      tau = tau +...
      tauaa(idim)*(na_trial(1:dim,idim)*na_trial(1:dim,idim)');
      be = be + ...
      lambdae(idim)^2*(na_trial(1:dim,idim)*na_trial(1:dim,idim)');
  end
else
  tau = tau_trial(1:dim,1:dim);
  tauaa = tauaa_trial(1:dim);
  Dgamma = 0;
  nu_a = zeros(dim,1);be = be_trial(1:dim,1:dim);
end

Cauchy = tau/J;
Cauchyaa = tauaa/J;
PLAST.stress.Cauchy = Cauchy;
PLAST.stress.Cauchyaa = Cauchyaa;

invF = inv(F);
PLAST.UPDATED.invCp = invF*(be*invF');
PLAST.UPDATED.epbar = ep + Dgamma;

PLAST.trial.lambdae = lambdae_trial(1:dim);
PLAST.trial.tau = tau_trial(1:dim,1:dim);
PLAST.trial.n = na_trial(1:dim,1:dim);

PLAST.yield.f = f;
PLAST.yield.Dgamma = Dgamma;
PLAST.yield.nu_a = nu_a(1:dim);
```

For the elastic case, the same implementation as with material 3 is used to calculate

the deviatoric Cauchy principal stresses. Note that the pressure found in `mean_dilatation_pressure.m` is added to the deviatoric Cartesian stress components in `pressure_addition.m`. Analogously, the deviatoric and volumetric components of the elasticity tensor are identical to those for materials 3 and 5 and computed in `ctens17.m` and `pressure_addition.m`, respectively.

For the plastic case, the radial return algorithm given by Equations (7.59) and (7.60) is implemented to find the incremental plastic multiplier and the deviatoric Kirchhoff principal stresses. In addition, the elastic left Cauchy–Green tensor is evaluated from Equation (7.49). The Cartesian deviatoric Cauchy stresses are found using Equation (6.81) and added to the pressure in `pressure_addition.m`. The deviatoric component of the tangent modulus, given by Equation (7.62a), is evaluated in `ctens17.m` and added to the volumetric component in `pressure_addition.m`. Finally, the state variables are updated, namely, the Von Mises equivalent strain (Equation (7.61)) and the inverse of plastic right Cauchy–Green tensor given by Equation (7.51a,b)$_b$.

The radial return algorithm is described in Section 7.6.1 and summarized under Material 10.17 in Appendix 10.14 at the end of this chapter. The vector `nu_a`, normal to the yield surface, is given by Equation (7.54a,b) and the incremental plastic multiplier `Dgamma` by Equation (7.59). Trial stretches `lambdae` are evaluated using Equation (7.45a,b). The radial return operation gives the deviatoric Kirchhoff principal stresses `tauaa` using Equation (7.60). Finally, the updated left Cauchy–Green tensor `be` is evaluated using Equation (7.49).

Additionally, Appendix 10.14 contains the constitutive equations for all the material types presented in indicial form with appropriate references to equations within the text.

10.9 Function `constitutive_matrix`

This function implements the constitutive component of the tangent stiffness matrix. Recognizing the continual iterative updating of the geometric and constitutive terms, the function is identical to that found in a standard linear elasticity program.

```
constitutive_matrix

function [element_indexi,element_indexj,element_stiffness,counter] =...
  constitutive_matrix(FEM,dim,element_connectivity,kinematics_gauss,...
  c,JW,counter,element_indexi,element_indexj,element_stiffness)

for bnode=1:FEM.mesh.n_nodes_elem
   for anode=1:FEM.mesh.n_nodes_elem
      for j=1:dim
          indexj = FEM.mesh.dof_nodes(j,element_connectivity(bnode));
          for i=1:dim
              indexi = FEM.mesh.dof_nodes(i,element_connectivity(anode))
              ;
              sum = 0;
              for k=1:dim
                  for l=1:dim
                      sum = sum +...
```

```
                    kinematics_gauss.DN_x(k,anode)*c(i,k,j,l)*...
                    kinematics_gauss.DN_x(l,bnode)*JW;
                end
            end
            element_indexi(counter) = indexi;
            element_indexj(counter) = indexj;
            element_stiffness(counter) = sum;
            counter = counter + 1;
        end
    end
  end
end
```

The constitutive matrix calculation shown above is within the Gauss integration loop shown in Algorithm 1. The innmermost quantity `sum` is a direct implementation of Equation (9.35).

10.10 Function `geometric_matrix`

Although all stiffness matrix calculations are a direct implementation of the relevant equations given in the text, it is instructive to consider the computation and assembly of the initial stress matrix $\mathbf{K}_\sigma$, which is of particular significance in finite deformation analysis.

```
geometric_matrix

function [element_indexi,element_indexj,element_stiffness,...
         counter] = geometric_matrix(FEM,dim,element_connectivity,...
         DN_sigma_DN,JW,counter,element_indexi,element_indexj,...
         element_stiffness)

for bnode=1:FEM.mesh.n_nodes_elem
    for anode=1:FEM.mesh.n_nodes_elem
        DNa_sigma_DNb = DN_sigma_DN(anode,bnode);

        element_indexi(counter:counter+dim-1) = ...
        FEM.mesh.dof_nodes(:,element_connectivity(anode));

        element_indexj(counter:counter+dim-1) = ...
        FEM.mesh.dof_nodes(:,element_connectivity(bnode));

        element_stiffness(counter:counter+dim-1) = ...
        DNa_sigma_DNb*JW;

        counter = counter + dim;
    end
end
```

This function is called from `element_force_and_stiffness.m` within a Gauss and element loop. The variable `DNa_sigma_DNb` contains the quantity $\nabla N_a \cdot \boldsymbol{\sigma} \nabla N_b$ computed according to Equation (9.44c), which is saved in the vector `element_stiffness`. The vectors `element_indexi` and `element_indexj` store the degree-of-freedom numbers corresponding to spatial directions of the nodes `anode` and `bnode`. Given the diagonal structure of every elemental initial stress matrix

contribution $\boldsymbol{K}^{(e)}_{\sigma,ab}$, only diagonal entries are stored.

10.11 Function `pressure_load_and_stiffness_assembly`

For cases where there are surface or line elements with pressure applied, this function assembles the equivalent nodal forces and tangent stiffness matrix contributions.

`pressure_load_and_stiffness_assembly` **– Surface element loop**

```
  function GLOBAL = pressure_load_and_stiffness_assembly(GEOM,...
  MAT,FEM,GLOBAL,LOAD,QUADRATURE,xlamb)
  GLOBAL.nominal_pressure = zeros(FEM.mesh.n_dofs,1);

  GLOBAL.R_pressure = zeros(FEM.mesh.n_dofs,1);
  GLOBAL.K_pressure = sparse([],[],[],FEM.mesh.n_dofs,FEM.mesh.n_dofs);

n_components = ...
(FEM.mesh.n_face_dofs_elem^2*QUADRATURE.ngauss)*LOAD.n_pressure_loads;
indexi = zeros(n_components,1);
indexj = zeros(n_components,1);
global_stiffness = zeros(n_components,1);
counter = 1;

for ipressure = 1:LOAD.n_pressure_loads
  element = LOAD.pressure_element(ipressure);
  global_nodes = FEM.mesh.connectivity_faces(:,element);
  material_number = MAT.matno(element);
  matyp = MAT.matyp(material_number);
  properties = MAT.props(:,material_number);
  xlocal_boundary = GEOM.x(:,global_nodes);

  counter0 = counter;
  [R_pressure_0,indexi,indexj,global_stiffness,counter] = ...
  pressure_load_element_Residual_and_Stiffness(properties,...
  matyp,xlocal_boundary,global_nodes,GEOM.ndime,...
  QUADRATURE,FEM,counter,indexi,indexj,global_stiffness);

  nominal_pressure = LOAD.pressure(ipressure)*R_pressure_0;
  GLOBAL.nominal_pressure = force_vectors_assembly(nominal_pressure,...
  global_nodes,GLOBAL.nominal_pressure,FEM.mesh.dof_nodes);
  global_stiffness(counter0:counter-1) = ...
  LOAD.pressure(ipressure)*xlamb*global_stiffness(counter0:counter-1);
end

GLOBAL.K_pressure = sparse(indexi,indexj,global_stiffness,...
FEM.mesh.n_dofs,FEM.mesh.n_dofs);
GLOBAL.Residual = GLOBAL.Residual + xlamb*GLOBAL.nominal_pressure;
GLOBAL.K = GLOBAL.K - GLOBAL.K_pressure;
```

This function loops over the elements and, for each element, calls the function `pressure_element_load_and_stiffness.m` to evaluate the current unit pressure load vector, which is stored in the variable `R_pressure_0`. Within the latter function, a loop over the Gauss points of the surface or line elements is carried out, where the tangent vectors $\partial \boldsymbol{x}/\partial \xi$ and $\partial \boldsymbol{x}/\partial \eta$ are evaluated. This enables the equivalent nodal forces and the tangent stiffness matrix contributions due to surface pressure to be calculated. For two-dimensional cases, the vector $\partial \boldsymbol{x}/\partial \eta$ is set to $[0, 0, -1]^T$, so that

capabilities of FLagSHyP and illustrate some of the difficulties that may arise in the analysis of highly nonlinear problems. Where appropriate, the yield stress is chosen to give obvious finite deformation elastic behavior prior to the onset of plasticity.

10.12.1 Simple Patch Test

As a simple check on the code, the nonlinear plane strain patch test example shown in Figure 10.5 is studied. Two irregular six-noded elements making up a square are employed and boundary displacements are prescribed as shown in order to obtain a uniform deformation gradient tensor given by

$$\boldsymbol{F} = \begin{bmatrix} 2 & 0 \\ 0 & 3/4 \end{bmatrix}; \quad J = 3/2.$$

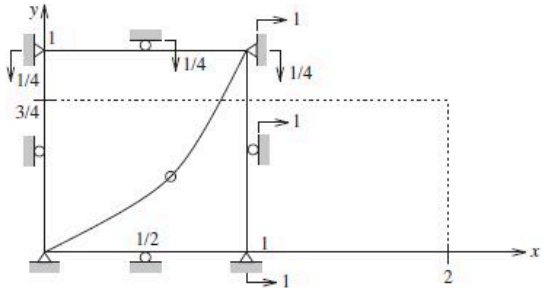


Figure 10.5 Patch test.

Assuming material 1 with values $\lambda = 100$ and $\mu = 100$, the program gives the correct state of uniform stress that can be evaluated from Equation (6.29) as

$$\boldsymbol{\sigma} = \frac{\mu}{J}(\boldsymbol{b} - \boldsymbol{I}) + \frac{\lambda}{J}(\ln J)\boldsymbol{I} = \begin{bmatrix} 227.03 & 0 \\ 0 & -2.136 \end{bmatrix};$$

$$\boldsymbol{b} = \boldsymbol{F}\boldsymbol{F}^T = \begin{bmatrix} 4 & 0 \\ 0 & 9/16 \end{bmatrix}.$$

10.12.2 *Nonlinear Truss*

Plane stress truss: The nonlinear truss example described in Section 1.3.2 can now be re-examined. For this purpose, a single four-noded element is used with the support conditions shown in Figure 10.6(a). In order to minimize the two-dimensional effects, a large length-to-thickness ratio of approximately 100 is used. The initial angle is 45° as in Figure 1.5 of Chapter 1 and the initial thickness is set to $1/\sqrt{2}$ so that the initial area of the truss is 1. Material 8 is used with $\mu = 1/3$, which for $\nu = 0.5$ implies $E = 1$. The displacement of the top node is prescribed either upward or downward in small increments of 0.5. The resulting vertical force is shown in Figure 10.6(b) and is practically identical to that obtained in Section 1.3.2 using the logarithmic strain equation and shown in Figure 1.5.

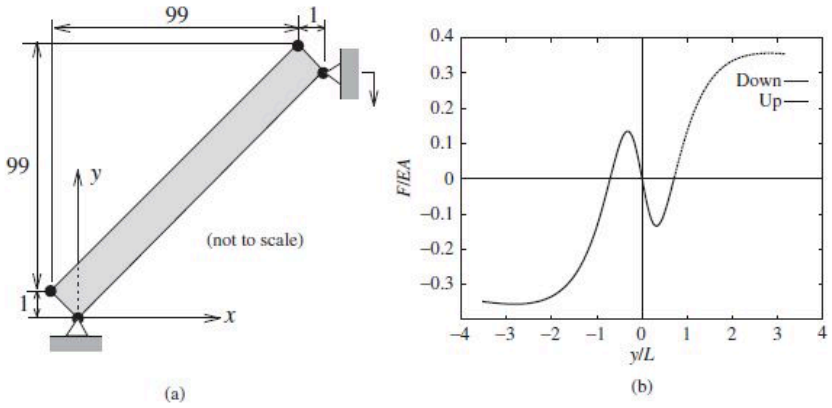


Figure 10.6 Plane stress truss example: (a) Geometry; (b) Load–displacement curve.

It is worth noting that when the truss is pushed downward and reaches the -45° position, the program apparently fails to converge. The reason for this is simply that at this position the stresses in the truss, and hence all the reactions, are zero to within machine accuracy. Consequently, when the convergence norm is evaluated by dividing the norm of the residuals by the norm of the reactions, a practically random number is obtained. In order to escape from this impasse, an artificially large convergence norm must be chosen for the increment that corresponds to this position (0.98 has been used by the authors). The computation can then be continued using normal convergence norms for the rest of the increments. Note that the restarting facilities in FLagSHyP enable this artifice to be easily performed.

Solid truss: The elasto-plastic single truss element example given in Section 3.6.1 is now repeated using a single eight-node hexahedron element with the same dimensions and boundary conditions as those given in Figure 10.6(a). In order to maintain a value of Young's modulus of 210 000 kN/mm^2 and a Poisson's ratio of 0.3, the corresponding constants for material 17 are $\mu = 80\ 769.23$ kN/mm^2 and

$\lambda = 121\ 153.86$ kN/mm^2, the yield stress and hardening parameters remaining the same at $\tau_y = 25\ 000.0$ kN/mm^2 and 1.0 respectively. The analysis was carried out using a fixed arc length of 5.0. Elastic and elasto-plastic load deflection curves are shown in Figure 10.7 (the Z axis now being the vertical direction), and coincide with the results using the single truss element, the elastic analysis being achieved with an artificially high yield stress.

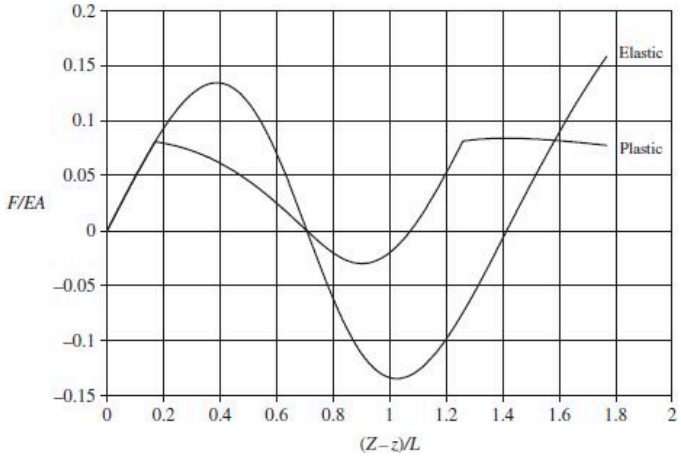


Figure 10.7 Solid truss example: elastic and elasto-plastic load–displacement curves.

10.12.3 Strip with a Hole

This is a well-known plane stress hyperelasticity example where an initial $6.5 \times 6.5 \times 0.079$ mm^3 strip with a hole 0.5 mm in diameter is stretched in the horizontal direction and constrained in the vertical direction, as shown in Figure 10.8. Given the two planes of symmetry, 100 four-noded quadrilateral elements are used to describe a quarter of the problem. A plane stress incompressible neo-Hookean material model is used with $\mu = 0.4225$ N/mm^2. The strip is stretched to six times its horizontal length using only five increments. The load-displacement curve is shown in Figure 10.8(b), and Figure 10.8(c) displays the final mesh.

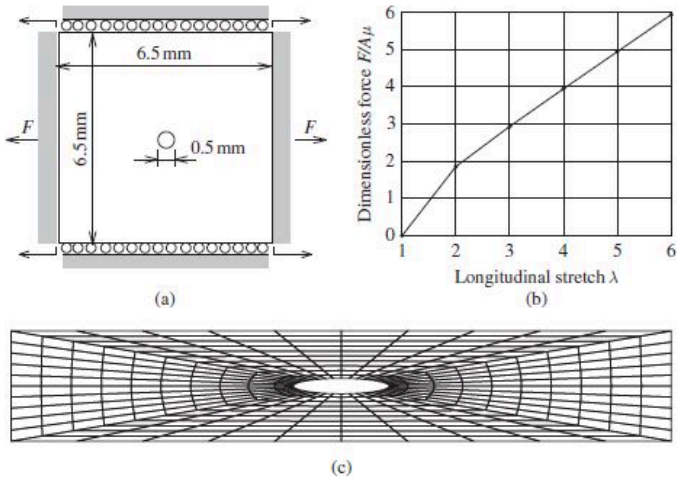


Figure 10.8 Strip with hole: (a) Geometry; (b) Load–stretch curve; (c) Final mesh.

10.12.4 Plane Strain Nearly Incompressible Strip

This well-known example has been included in order to illustrate the difficulties that

can be encountered using the penalty type of nearly incompressible plane strain or three-dimensional materials implemented in FLagSHyP in conjunction with a displacement control process. In this case a 20×20 mm^2 strip is clamped and stretched as shown in Figure 10.9. Because of the symmetry, only a quarter of the problem is actually modeled, using 256 four-noded mean dilatation quadrilateral elements. Material 5 is used with $\mu = 0.4225$ N/mm^2 and $\kappa = 5$ N/mm^2. The final mesh is shown in Figure 10.9(b), where for a total horizontal stretch of 3 a vertical stretch of 0.3711 is observed (smaller values are obtained as κ is increased to reach the incompressible limit).

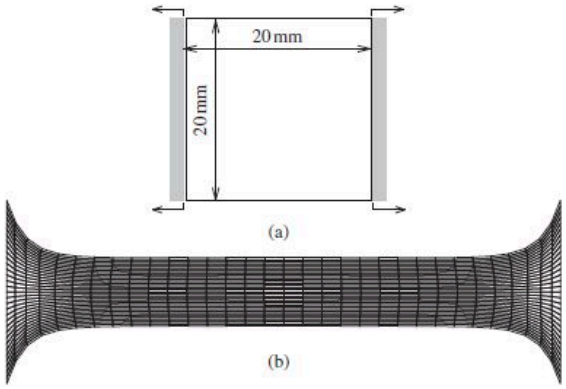


Figure 10.9 Incompressible strip: (a) Geometry; (b) Final mesh.

Although the material is slightly compressible ($\nu \approx 0.46$), the solution requires 200 increments in order to ensure that the Newton–Raphson iterative process reaches convergence in a reasonable number of steps. In fact, if the value of κ is increased in order to approach the incompressible limit (say to 50 or 500), an even larger number of increments is needed. This is in clear contrast with the previous example, which could be run in five increments. Unfortunately, using large increments in the present case leads to large changes in the volumes of those elements containing nodes with prescribed displacements, which, because of the relatively large bulk modulus, give extremely large internal pressures at these elements. These unrealistic pressure values lead in turn to very large residual forces from which the Newton–Raphson process fails to converge. This problem is typically solved using the augmented Lagrangian technique whereby equilibrium is first reached using small initial values of κ, which are then progressively increased, always maintaining equilibrium by further iterations if necessary, until the desired κ/μ ratio is reached. This technique enables very large final values of κ/μ to be used without increasing the number of increments needed. In order to keep the code as simple as possible, however, this technique has not been implemented in FLagSHyP, which implies that a very large number of increments is needed for plane strain or three-dimensional problems involving true incompressible materials and displacement control.

10.12.5 Twisting Column

This example shows the deformation pattern of a relatively complex structure when subjected to a torsion induced behavior via the application of surface loads. The structure is comprised of a horizontal short beam resting on top of a column. The geometry and dimensions (without units) are those shown in Figure 10.10(a). Also in that figure, the finite element structured mesh used for the numerical simulation is displayed, comprising 576 hexahedral elements (`hexa8`), with an element size chosen as $0.25 \times 0.25 \times 0.25$ throughout the entire domain. A three-dimensional compressible

neo-Hookean material has been used with material properties (without units) $\mu = 100$ and $\lambda = 100$, corresponding to a Young's modulus of 250 and a Poisson's ratio of 0.25, respectively.

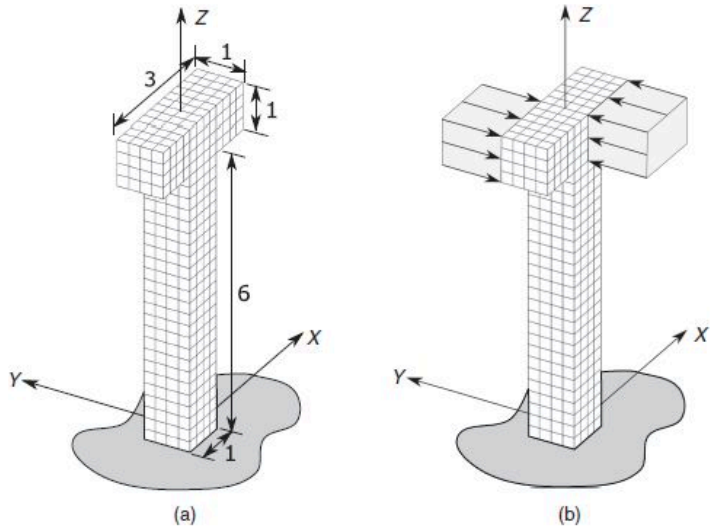


Figure 10.10 Twisting column: (a) Geometry; (b) Pressure load distribution.

A torque is applied on the structure through a uniform positive (compressive) pressure follower load of nominal value 100 on the horizontal structural component, specifically in the regions defined by $\{0 \leq X \leq 1.5\} \cap \{Y = 0.5\} \cap \{Z \geq 6\}$ and $\{1.5 \leq X \leq 0\} \cap \{Y = -0.5\} \cap \{Z \geq 6\}$. A sketch showing this pressure load is displayed in Figure 10.10(b). The load leads to the rotational deformation depicted in Figure 10.11, where a series of snapshots are included for various values of the load increment `CON.xlamb`. The simulation is performed without the need to resort to the line search or arc length algorithms. A variety of possible output plots are possible. However, to get a sense of the rotation of the column, Figure 10.12 shows the force–displacement diagram corresponding to the node defined by initial coordinates $[1, 0.5, 6.5]$. Specifically, graphical representation of the norm of the equivalent nodal force (due to the surface load) at the node is plotted against the component of the nodal displacement along the *OX* direction.

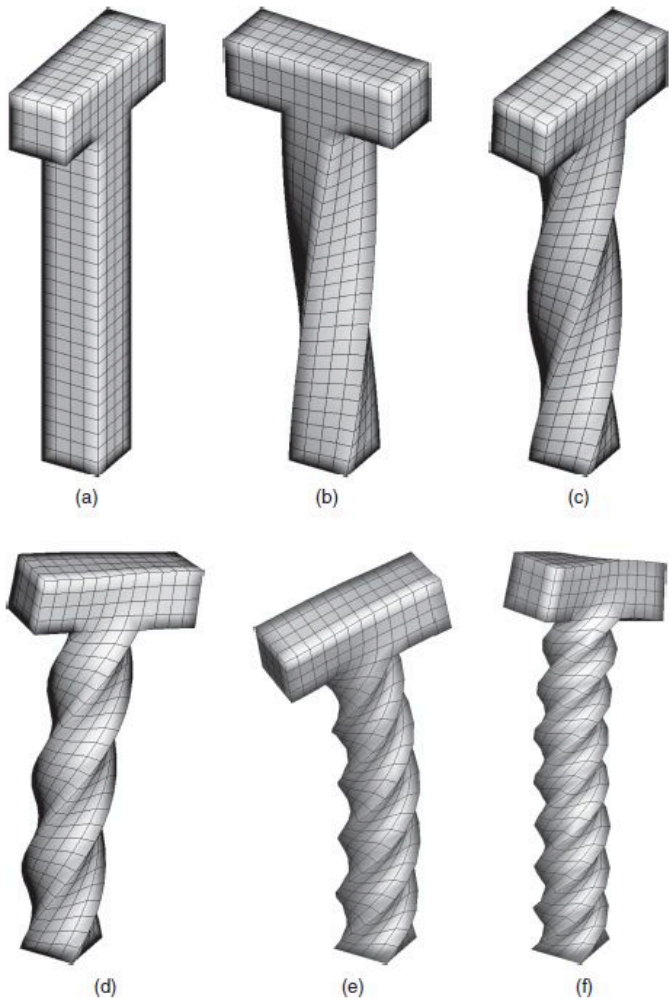


Figure 10.11 Twisting column example: (a) `CON.xlamb`=0.00; (b) `CON.xlamb` = 0.16; (c) `CON.xlamb`=0.32; (d) `CON.xlamb`=0.60; (e) `CON.xlamb`=0.92; (f) `CON.xlamb`=1.24.

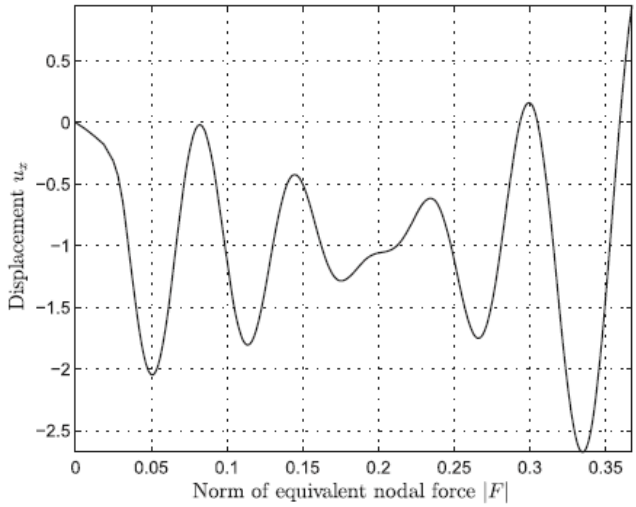


Figure 10.12 Twisting column example: curve displaying the norm of the equivalent nodal force vs the *OX* component of the displacement at node $[1, 0.5, 6.5]$.

10.12.6 Elasto-Plastic Cantilever

This example shows the elastic and elasto-plastic behavior of a cantilever undergoing finite deformations. The geometry is shown in Figure 10.13(a), where 20, 2, and 8 `hexa8` elements are used in the X, Y, and Z directions respectively. The constants for material 17 are $\mu = 80\,769.23$ kN/mm^2 and $\lambda = 121\,153.86$ kN/mm^2, the yield stress and hardening parameters being $\tau_y = 2500.0$ kN/mm^2 and 1.0 corresponding to a value of Young's modulus of $210\,000$ kN/mm^2 and a Poisson's ratio of 0.3 respectively. The elastic analysis is carried out with an artificially high yield stress. In both elastic and elasto-plastic cases a variable arc length is used. Figures 10.13(b,d) show the load deflection behavior at the tip of the cantilever, while Figure 10.13(c) shows the development of plasticity at the support for a load of 788.18 N. Observe that Figure 10.13(c) shows coordinates and not the displacement at the loaded point. Also, note that an analysis will fail when the tangent matrix looses its positive definite nature for this material model. Essentially, the simple logarithmic stretch-based constitutive model is only valid for moderate deformations, even though displacements may be large.

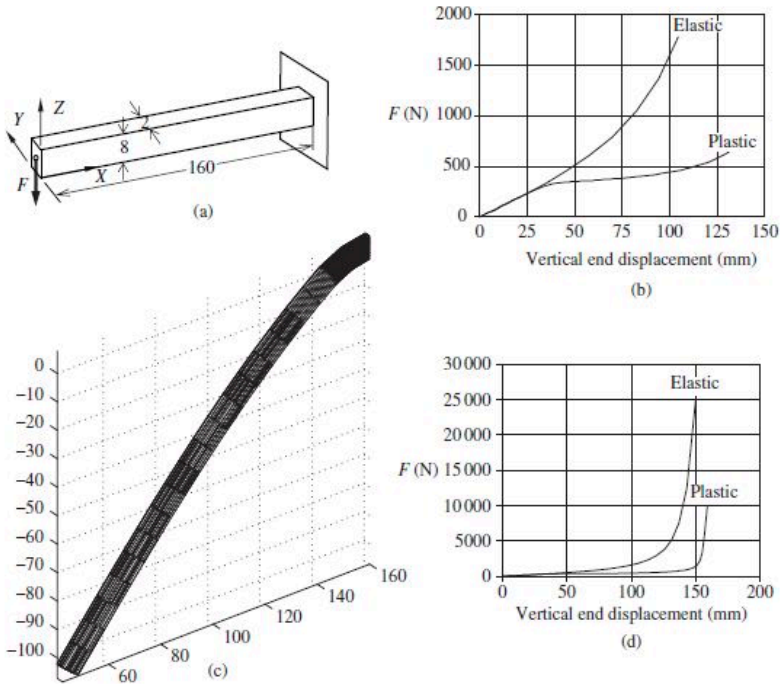


Figure 10.13 Large deflection elasto-plastic behavior of a cantilever: (a) Geometry; (b) Detail of force deflection behavior; (c) Deformation showing yielding at the support at a load of 788.18 N; (d) Overall force deflection behavior.

10.13 Appendix: dictionary of main variables

The main variables of FLagSHyP are listed below, grouped according to their parent data structure. In addition, some other important variables are included.

`PRO.rest`	-	Restart file indicator: `.true.` if problem is restarted, `.false.` if problem is started from scratch
`PRO.title`	-	Title of the problem
`PRO.inputfile_name`	-	Name of input file
`PRO.outputfile_name`	-	Name of output file
`PRO.outputfile_name_flagout`	-	Name of single degree of freedom

		output file
`GEOM.ndime`	-	Number of dimensions
`GEOM.npoin`	-	Number of nodes in the mesh
`GEOM.x`	-	Current coordinates
`GEOM.x0`	-	Initial coordinates
`GEOM.Ve`	-	Initial elemental volume
`GEOM.V_total`	-	Total initial volume
`FEM.mesh.element_type`	-	Type of element
`FEM.mesh.n_nodes_elem`	-	Number of nodes per element
`FEM.mesh.n_face_nodes_elem`	-	Number of nodes per surface (line) element subjected to pressure loads
`FEM.mesh.nelem`	-	Number of elements
`FEM.mesh.connectivity`	-	Element connectivity
`FEM.mesh.connectivity_faces`	-	Connectivity of surface (line) elements subjected to surface loads
`FEM.interpolation.element.N`	-	Element shape functions
`FEM.interpolation.element.DN_chi`	-	Element shape function derivatives
`FEM.interpolation.boundary.N`	-	Boundary element shape functions
`FEM.interpolation.boundary.DN_chi`	-	Shape function derivatives of surface (line) elements subjected to pressure loads
`QUADRATURE.element.Chi`	-	Gauss point locations
`QUADRATURE.element.W`	-	Gauss point weights
`QUADRATURE.element.ngauss`	-	Number of Gauss points per element
`QUADRATURE.boundary.Chi`	-	Gauss point locations (boundary element)
`QUADRATURE.boundary.W`	-	Gauss point weights (boundary element)
`QUADRATURE.boundary.ngauss`	-	Number of Gauss points per boundary element
`KINEMATICS.DN_x`	-	Current Cartesian derivatives of shape functions
`KINEMATICS.Jx_chi`	-	Volume ratio (Jacobian) between current and nondimensional isoparametric domain
`KINEMATICS.F`	-	Deformation gradient tensor
`KINEMATICS.J`	-	Volume ratio (Jacobian) between current and initial domain ($J = \det \boldsymbol{F}$)
`KINEMATICS.b`	-	Left Cauchy–Green deformation tensor
`KINEMATICS.Ib`	-	First invariant of the left Cauchy–Green deformation tensor
`KINEMATICS.lambda`	-	Principal stretches
`KINEMATICS.n`	-	Principal directions
`MAT.matno`	-	Element material number identifier
`MAT.nmats`	-	Number of materials
`MAT.props`	-	Material properties: the first is always the initial density, the rest depend on the material type
`MAT.matyp`	-	Material number identifier
`MAT.n_nearly_incompressible`	-	Number of nearly incompressible materials
`BC.icode`	-	Nodal boundary condition identifier
`BC.freedof`	-	Nodal free degrees of freedom
`BC.fixdof`	-	Nodal fixed degrees of freedom
`BC.n_prescribed_displacements`	-	Number of prescribed displacements
`BC.presc_displacement`	-	Prescribed displacement vector
`BC.dofprescribed`	-	Prescribed degree of freedom
`LOAD.n_pressure_loads`	-	Number of pressure load surface elements
`LOAD.gravt`	-	Gravity vector

`LOAD.pressure`	-	Value of pressure in surface element
`LOAD.pressure_element`	-	Element subjected to pressure
`GLOBAL.nominal_external_load`	-	Nominal external load (without pressure)
`GLOBAL.Residual`	-	Global residual vector
`GLOBAL.external_load`	-	Current load vector
`GLOBAL.nominal_pressure`	-	Equivalent nodal force due to pressure
`GLOBAL.T_int`	-	Equivalent internal force vector
`GLOBAL.K`	-	Global stiffness matrix
`GLOBAL.R_pressure`	-	Equivalent internal force vector due to pressure
`GLOBAL.K_pressure`	-	Global stiffness matrix due to pressure
`PLAST.yield.f`	-	Yield surface value
`PLAST.yield.Dgamma`	-	Incremental plastic multiplier
`PLAST.yield.nu_a`	-	Direction vector
`PLAST.trial.lambdae`	-	Trial elastic stretches
`PLAST.trial.tau`	-	Trial Kirchhoff stress tensor
`PLAST.trial.n`	-	Trial eigenvectors
`PLAST.UPDATED.invCp`	-	Updated plastic right Cauchy–Green tensor
`PLAST.UPDATED.epbar`	-	Updated equivalent plastic strain
`PLAST.stress.Cauchy`	-	Cauchy stress tensor
`PLAST.stress.Cauchyaa`	-	Cauchy principal stresses
`CON.nincr`	-	Number of load increments
`CON.incrm`	-	Current load increment
`CON.xlmax`	-	Maximum load parameter
`CON.dlamb`	-	Incremental load value
`CON.miter`	-	Maximum number of iterations per increment
`CON.niter`	-	Current iteration per increment
`CON.cnorm`	-	Convergence criterion for Newton-Raphson
`CON.searc`	-	Line search parameter
`CON.msearch`	-	Maximum number of line search iterations
`CON.incrm`	-	Current load increment
`CON.xlamb`	-	Current load parameter
`CON.ARCLEN.farcl`	-	Logical fixed arc length indicator
`CON.ARCLEN.arcln`	-	Arc length parameter (0.0 means no arc length)
`CON.ARCLEN.itarget`	-	Target iteration/increment for variable arc length option
`CON.ARCLEN.iterold`	-	Number of iterations in previous increment for arc length
`CON.ARCLEN.xincr`	-	Total displacement over the load increment
`CON.ARCLEN.afail`	-	Logical arc length failure indicator
`CON.OUTPUT.incout`	-	Output counter
`CON.OUTPUT.nwant`	-	Single output node
`CON.OUTPUT.iwant`	-	Output degree of freedom
`eta`	-	Scaling factor or displacement factor
`eta0`	-	Previous value of the parameter `eta`
`rtu`	-	Current dot product of $\mathbf{R}$ by $\mathbf{u}$
`rtu0`	-	Initial dot product of $\mathbf{R}$ by $\mathbf{u}$
`displ`	-	Newton–Raphson displacement vector $\mathbf{u}$
`dispf`	-	load component of the displacement vector $\mathbf{u}_F$
`rnorm`	-	Current residual norm
`c`	-	Fourth-order tensor

`indexi`	-	Vector storing the row entry of the stiffness matrix coefficient
`indexj`	-	Vector storing the column entry of the stiffness matrix coefficient
`counter`	-	Counter to travel through the vectors storing the stiffness matrix coefficients
`global_stiffness`	-	Vector storing each coefficient of the stiffness matrix
`mu`	-	μ coefficient (material parameter)
`lambda`	-	λ coefficient (material parameter)
`kappa`	-	κ coefficient (material parameter)
`H`	-	strain hardening (material parameter)

10.14 Appendix: constitutive equation summary

To facilitate the understanding of the implementation of the various constitutive equations, the boxes below summarize the required constitutive and kinematic equations for each material type. These equations are presented here in an indicial form to concur with the code.

Material 10.1: Three-dimensional or plane strain compressible neo-Hookean

$$\sigma_{ij} = \frac{\mu}{J}(b_{ij} - \delta_{ij}) + \frac{\lambda}{J}(\ln J)\delta_{ij} \tag{6.29}$$

$$\mathfrak{c}_{ijkl} = \lambda'\delta_{ij}\delta_{kl} + \mu'\left(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}\right) \tag{6.40}$$

$$\lambda' = \frac{\lambda}{J}; \qquad \mu' = \frac{\mu - \lambda \ln J}{J} \tag{6.41}$$

Material 10.2: One-dimensional stretch-based hyperelastic plastic

$$\lambda_{e,n+1}^{\text{trial}} = \frac{l_{n+1}}{l_{p,n}}\,; \quad J = \left(\lambda_{e,n+1}^{\text{trial}}\right)^{(1-2\nu)} \tag{3.67, 3.66}$$

$$\varepsilon_{e,n+1}^{\text{trial}} = \ln \lambda_{e,n+1}^{\text{trial}} \tag{3.68a,b,c}$$

$$\tau_{n+1}^{\text{trial}} = E\,\varepsilon_{e,n+1}^{\text{trial}} \tag{3.77a,b}$$

$$f\left(\tau_{n+1}^{\text{trial}}, \bar{\varepsilon}_{p,n}\right) = \left|\tau_{n+1}^{\text{trial}}\right| - \left(\tau_y^0 + H\bar{\varepsilon}_{p,n}\right) \tag{3.78}$$

$$\Delta\gamma = \begin{cases} \dfrac{f\left(\tau_{n+1}^{\text{trial}}, \bar{\varepsilon}_{p,n}\right)}{E+H} & \text{if } f\left(\tau_{n+1}^{\text{trial}}, \bar{\varepsilon}_{p,n}\right) > 0 \\ 0 & \text{if } f\left(\tau_{n+1}^{\text{trial}}, \bar{\varepsilon}_{p,n}\right) \le 0 \end{cases} \tag{3.85}$$

$$\tau_{n+1} = \tau_{n+1}^{\text{trial}} - E\Delta\gamma \ \ \text{sign}(\tau_{n+1}) \tag{3.81}$$

$$\textbf{If} \quad f > 0 \quad \frac{d\,\tau_{n+1}}{d\,\varepsilon_{n+1}} = \frac{EH}{E+H} \tag{3.90}$$

$$\textbf{Else} \qquad \frac{d\,\tau_{n+1}}{d\,\varepsilon_{n+1}} = E \tag{3.15}$$

Material 10.3: Three-dimensional or plane strain hyperelasticity in principal directions

$$\sigma_{\alpha\alpha} = \frac{2\mu}{J}\ln\lambda_\alpha + \frac{\lambda}{J}\ln J \tag{6.94}$$

$$\sigma_{ij} = \sum_{\alpha=1}^{3} \sigma_{\alpha\alpha} T_{\alpha i} T_{\alpha j}; \qquad (T_{\alpha i} = \boldsymbol{n}_\alpha \cdot \boldsymbol{e}_i) \tag{6.81}$$

$$\begin{aligned} \mathfrak{c}_{ijkl} = &\sum_{\alpha,\beta=1}^{3} \frac{\lambda + 2(\mu - J\sigma_{\alpha\alpha})\delta_{\alpha\beta}}{J}\, T_{\alpha i} T_{\alpha j} T_{\beta k} T_{\beta l} \\ &+ \sum_{\alpha,\beta=1}^{3} \mu_{\alpha\beta}\ (T_{\alpha i} T_{\beta j} T_{\alpha k} T_{\beta l} + T_{\alpha i} T_{\beta j} T_{\beta k} T_{\alpha l}) \end{aligned} \tag{6.90, 6.95}$$

$$\mu_{\alpha\beta} = \frac{\sigma_{\alpha\alpha}\lambda_\beta^2 - \sigma_{\beta\beta}\lambda_\alpha^2}{\lambda_\alpha^2 - \lambda_\beta^2}; \quad \text{if } \lambda_\alpha \neq \lambda_\beta \quad \text{or}$$
$$\mu_{\alpha\beta} = \frac{\mu}{J} - \sigma_{\alpha\alpha} \ \text{ if } \lambda_\alpha = \lambda_\beta \tag{6.90, 6.91, 6.95}$$

Material 10.4: Plane stress hyperelasticity in principal directions

$$\gamma = \frac{2\mu}{\lambda + 2\mu} \tag{6.116a,b}_{\text{b}}$$

$$\bar{\lambda} = \gamma\lambda \tag{6.116a,b}_{\text{a}}$$

$$J = j^\gamma\,; \qquad (J = dv/dV;\ j = da/dA) \tag{6.117}$$

$$\sigma_{\alpha\alpha} = \frac{2\mu}{J}\ln\lambda_\alpha + \frac{\bar{\lambda}}{J}\ln j \tag{6.118}$$

$$\sigma_{ij} = \sum_{\alpha=1}^{2} \sigma_{\alpha\alpha} T_{\alpha i} T_{\alpha j}; \qquad (T_{\alpha i} = \boldsymbol{n}_\alpha \cdot \boldsymbol{e}_i)$$

(6.81)

$$\mathfrak{c}_{ijkl} = \sum_{\alpha,\beta=1}^{2} \frac{\bar{\lambda} + 2(\mu - J\sigma_{\alpha\alpha})\delta_{\alpha\beta}}{J} T_{\alpha i} T_{\alpha j} T_{\beta k} T_{\beta l} + \sum_{\substack{\alpha,\beta=1 \\ \alpha\neq\beta}}^{2} \mu_{\alpha\beta} \left(T_{\alpha i} T_{\beta j} T_{\alpha k} T_{\beta l} + T_{\alpha i} T_{\beta j} T_{\beta k} T_{\alpha l} \right)$$

(6.90, 6.119)

$$\mu_{\alpha\beta} = \frac{\sigma_{\alpha\alpha}\lambda_\beta^2 - \sigma_{\beta\beta}\lambda_\alpha^2}{\lambda_\alpha^2 - \lambda_\beta^2}\ ; \quad \text{if } \lambda_\alpha \neq \lambda_\beta \quad \text{or} \quad \mu_{\alpha\beta} = \frac{\mu}{J} - \sigma_{\alpha\alpha} \text{ if } \lambda_\alpha = \lambda_\beta$$

(6.90, 6.91, 6.119)

$$h = \frac{HJ}{j}$$

(Exercise 4 of Chapter 4)

Material 10.5: Three-dimensional or plane strain nearly incompressible neo-Hookean

$$\bar{J} = \frac{v^{(e)}}{V^{(e)}} \tag{8.53a}$$

$$p = \kappa(\bar{J} - 1) \tag{8.55}$$

$$\bar{\kappa} = \kappa \frac{v^{(e)}}{V^{(e)}} \tag{8.64}$$

$$\sigma'_{ij} = \mu J^{-5/3} \left(b_{ij} - \frac{1}{3} I_{\boldsymbol{b}} \delta_{ij} \right) \tag{6.55}$$

$$\sigma_{ij} = \sigma'_{ij} + p\delta_{ij} \tag{5.49a,b(a))}$$

$$\hat{\mathfrak{c}}_{ijkl} = 2\mu J^{-5/3} \left[\frac{1}{6} I_{\boldsymbol{b}} \left(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk} \right) - \frac{1}{3} b_{ij}\delta_{kl} - \frac{1}{3}\delta_{ij} b_{kl} + \frac{1}{9} I_{\boldsymbol{b}} \delta_{ij}\delta_{kl} \right]$$

(6.59a)

$$\mathfrak{c}_{p,ijkl} = p \left(\delta_{ij}\delta_{kl} - \delta_{ik}\delta_{jl} - \delta_{il}\delta_{jk} \right)$$

(6.59b)

Material 10.6: Plane stress incompressible neo-Hookean (Exercise 1 of Chapter 6)

$$J = 1; \quad (J = dv/dV,\ j = da/dA)$$

$$\sigma_{ij} = \mu(b_{ij} - j^{-2}\delta_{ij}); \quad \left(j^2 = \det_{2\times 2} \boldsymbol{b} \right)$$

$$\mathfrak{c}_{ijkl} = \lambda' \delta_{ij}\delta_{kl} + \mu' \left(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk} \right)$$

$$\lambda' = \frac{2\mu}{j^2}$$

$$\mu' = \frac{\mu}{j^2}$$

$$h = \frac{H}{j}$$

Material 10.7: Nearly incompressible in principal directions

$$\bar{J} = v^{(e)}/V^{(e)} \tag{8.53a}$$

$$p = \frac{\kappa \ln \bar{J}}{\bar{J}} \tag{6.103}$$

$$\bar{\kappa} = \bar{J}\frac{dp}{d\bar{J}} = \frac{\kappa}{\bar{J}} - p \tag{8.62}$$

$$\sigma'_{\alpha\alpha} = -\frac{2\mu}{3J}\ln J + \frac{2\mu}{J}\ln\lambda_\alpha \tag{6.107}$$

$$\sigma'_{ij} = \sum_{\alpha=1}^{3} \sigma'_{\alpha\alpha} T_{\alpha i} T_{\alpha j}; \qquad (T_{\alpha i} = \boldsymbol{n}_\alpha \cdot \boldsymbol{e}_i) \tag{6.81}$$

$$\sigma_{ij} = \sigma'_{ij} + p\delta_{ij} \tag{5.49a,b)$_a$}$$

$$\mathfrak{c}_{p,ijkl} = p(\delta_{ij}\delta_{kl} - \delta_{ik}\delta_{jl} - \delta_{il}\delta_{jk}) \tag{6.59b}$$

$$\begin{aligned}\hat{\mathbf{c}}_{ijkl} = &\sum_{\alpha,\beta=1}^{3} \frac{2}{J}\left[(\mu - J\sigma'_{\alpha\alpha})\delta_{\alpha\beta} - \frac{1}{3}\mu\right] T_{\alpha i}T_{\alpha j}T_{\beta k}T_{\beta l} \\ &+ \sum_{\substack{\alpha,\beta=1 \\ \alpha\neq\beta}}^{3} \mu_{\alpha\beta}\,(T_{\alpha i}T_{\beta j}T_{\alpha k}T_{\beta l} + T_{\alpha i}T_{\beta j}T_{\beta k}T_{\alpha l})\end{aligned} \tag{6.110, 6.111}$$

$$\mu_{\alpha\beta} = \frac{\sigma'_{\alpha\alpha}\lambda_\beta^2 - \sigma'_{\beta\beta}\lambda_\alpha^2}{\lambda_\alpha^2 - \lambda_\beta^2}; \quad \text{if } \lambda_\alpha \neq \lambda_\beta \quad \text{or} \quad \mu_{\alpha\beta} = \frac{\mu}{J} - \sigma'_{\alpha\alpha} \text{ if } \lambda_\alpha = \lambda_\beta \tag{6.110, 6.91, 6.111}$$

Material 10.8: Plane stress incompressible in principal directions

$$\lambda \to \infty; \qquad \gamma = 0; \qquad \bar{\lambda} = 2\mu \tag{6.116a,b}$$

$$J = 1; \qquad (J = dv/dV; \qquad j = da/dA) \tag{6.117}$$

$$\sigma_{\alpha\alpha} = 2\mu\ln\lambda_\alpha + \bar{\lambda}\ln j \tag{6.118}$$

$$\sigma_{ij} = \sum_{\alpha=1}^{2} \sigma_{\alpha\alpha} T_{\alpha i} T_{\alpha j}; \quad (T_{\alpha i} = \boldsymbol{n}_\alpha \cdot \boldsymbol{e}_i) \tag{6.81}$$

$$\mathfrak{c}_{ijkl} = \sum_{\alpha,\beta=1}^{2} \left[\bar{\lambda} + 2(\mu - \sigma_{\alpha\alpha})\delta_{\alpha\beta}\right] T_{\alpha i}T_{\alpha j}T_{\beta k}T_{\beta l} + \sum_{\substack{\alpha,\beta=1 \\ \alpha\neq\beta}}^{2} \mu_{\alpha\beta}\ (T_{\alpha i}T_{\beta j}T_{\alpha k}T_{\beta l} + T_{\alpha i}T_{\beta j}T_{\alpha l}T_{\beta k})$$

(6.90, 6.119)

$$\mu_{\alpha\beta} = \frac{\sigma_{\alpha\alpha}\lambda_\beta^2 - \sigma_{\beta\beta}\lambda_\alpha^2}{\lambda_\alpha^2 - \lambda_\beta^2}; \qquad \text{if } \lambda_\alpha \neq \lambda_\beta \quad \text{or} \quad \mu_{\alpha\beta} = \mu - \sigma_{\alpha\alpha} \text{ if } \lambda_\alpha = \lambda_\beta$$

(6.90, 6.91, 6.119)

$$h = \frac{H}{j}$$

(Exercise 4 of Chapter 4)

Material 10.17: Three-dimensional or plane strain hyperelastic-plastic in principal directions

$$\bar{J} = v^{(e)}/V^{(e)} \tag{8.53a}$$

$$p \approx \kappa \frac{\ln \bar{J}}{\bar{J}} \tag{6.103}$$

$$\nu_\alpha^{n+1} = \frac{\tau_{\alpha\alpha}^{\prime trial}}{\sqrt{\frac{2}{3}}\|\boldsymbol{\tau}'^{\text{ trial}}\|} \tag{7.54a,b}$$

$$\Delta\gamma = \begin{cases} \dfrac{f(\boldsymbol{\tau}^{\text{trial}}, \bar{\varepsilon}_{p,n})}{3\mu + H} & \text{if } f(\boldsymbol{\tau}^{\text{trial}}, \bar{\varepsilon}_{p,n}) > 0 \\ 0 & \text{if } f(\boldsymbol{\tau}^{\text{trial}}, \bar{\varepsilon}_{p,n}) \leq 0 \end{cases} \tag{7.59}$$

$$\ln \lambda_{e,\alpha}^{n+1} = \ln \lambda_{e,\alpha}^{trial} - \Delta\gamma\nu_\alpha^{n+1} \tag{7.45a,b}$$

$$\tau'_{\alpha\alpha} = \left(1 - \frac{2\mu\Delta\gamma}{\sqrt{2/3}\|\boldsymbol{\tau}'^{\text{ trial}}\|}\right)\tau_{\alpha\alpha}^{\prime trial} \tag{7.60}$$

$$\sigma'_{\alpha\alpha} = \frac{1}{J_{n+1}}\,\tau'_{\alpha\alpha} \tag{5.31a,b}_{\text{b}}$$

$$\sigma_{\alpha\alpha} = \sigma'_{\alpha\alpha} + p \tag{5.49a,b}_{\text{a}}$$

$$\sigma_{ij} = \sum_{\alpha=1}^{2} \sigma_{\alpha\alpha}T_{\alpha i}T_{\alpha j}; \quad (T_{\alpha i} = \boldsymbol{n}_\alpha \cdot \boldsymbol{e}_i) \tag{6.81}$$

$$\hat{\mathfrak{c}}_{ijkl} = \sum_{\alpha,\beta=1}^{3} \frac{1}{J}\,\mathfrak{c}_{\alpha\beta}\,T_{\alpha i}T_{\alpha j}T_{\beta k}T_{\beta l} - \sum_{\alpha=1}^{3} 2\sigma'_{\alpha\alpha}\,T_{\alpha i}T_{\alpha j}T_{\alpha k}T_{\alpha l} + \sum_{\substack{\alpha,\beta=1 \\ \alpha\neq\beta}}^{3} \frac{\sigma'_{\alpha\alpha}\left(\lambda_{e,\beta}^{\text{trial}}\right)^2 - \sigma'_{\beta\beta}\left(\lambda_{e,\alpha}^{\text{trial}}\right)^2}{\left(\lambda_{e,\alpha}^{\text{trial}}\right)^2 - \left(\lambda_{e,\beta}^{\text{trial}}\right)^2}\,(T_{\alpha i}T_{\beta j}T_{\alpha k}T_{\beta l}\ + T_{\alpha i}T_{\beta j}T_{\beta k}T_{\alpha l})$$

(7.62a)

$$f(\boldsymbol{\tau}^{\text{trial}}, \bar{\varepsilon}_{p,n}) \le 0 \tag{7.20a,b}$$

If $f > 0$

$$\mathbf{c}_{\alpha\beta} = \left(1 - \frac{2\mu\,\Delta\gamma}{\sqrt{2/3}\,\|\boldsymbol{\tau}'^{\,\text{trial}}\|}\right)\left(2\mu\delta_{\alpha\beta} - \frac{2}{3}\mu\right) - 2\mu\,\nu_\alpha\nu_\beta\left(\frac{2\mu}{3\mu + H} - \frac{2\mu\sqrt{2/3}\,\Delta\gamma}{\|\boldsymbol{\tau}'^{\,\text{trial}}\|}\right) \tag{7.67}$$

Else

$$\mathbf{c}_{\alpha\beta} = 2\mu\delta_{\alpha\beta} - \frac{2}{3}\mu \tag{7.64}$$

---

* Note that the arc length option in data item 13 cannot be used if nonzero prescribed displacements are employed.

† For material 17, only ρ, μ, λ, τ_y, and H are entered; κ is calculated by the program and stored in `MAT.props(4,im)`.

Bibliography

BATHE, K-J., *Finite Element Procedures in Engineering Analysis*, Prentice Hall, 1996.

BELYTSCHKO, T., LIU, W. K., and MORAN, B., *Nonlinear Finite Elements for Continua and Structures,* John Wiley & Sons, 2000.

BONET, J. and BHARGAVA, P., The incremental flow formulation for the analysis of 3-dimensional viscous deformation processes: Continuum formulation and computational aspects, *Int. J. Num. Meth. Engrg.*, **122**, 51–68, 1995.

BONET, J., GIL, A. J., and ORTIGOSA, R., A computational framework for polyconvex large strain elasticity, *Comput. Meths. Appl. Mech. Engrg.*, **283**, 1061–1094, 2015.

BONET, J., WOOD, R. D., MAHANEY, J., and HEYWOOD, P., Finite element analysis of air supported membrane structures, *Comput. Meths. Appl. Mech. Engrg.*, **190**, 579–595, 2000.

CRISFIELD, M. A., *Non-Linear Finite Element Analysis of Solids and Structures*, John Wiley & Sons, Volume 1, 1991.

ETEROVIĆ, A. L. and BATHE, K-L., A hyperelastic-based large strain elasto-plastic constitutive formulation with combined isotropic-kinematic hardening using logarithmic stress and strain measures, *Int. J. Num. Meth. Engrg.*, **30**, 1099–1114, 1990.

GONZALEZ, O. and STUART, A. M., *A First Course in Continuum Mechanics,* Cambridge University Press, 2008.

GURTIN, M., *An Introduction to Continuum Mechanics*, Academic Press, 1981.

HOLZAPFEL, G. A., *Nonlinear Solid Mechanics: A Continuum Approach for Engineering*, John Wiley & Sons, 2000.

HUGHES, T. J. R., *The Finite Element Method*, Prentice Hall, 1987.

HUGHES, T. J. R. and PISTER, K. S., Consistent linearization in mechanics of solids and structures, *Compt. & Struct.*, **8**, 391–397, 1978.

LUBLINER, J., *Plasticity Theory*, Macmillan, 1990.

MALVERN, L. E., *Introduction to the Mechanics of a Continuous Medium*, Prentice Hall, 1969.

MARSDEN, J. E. and HUGHES, T. J. R., *Mathematical Foundations of Elasticity*, Prentice Hall, 1983.

MIEHE, C., Aspects of the formulation and finite element implementation of large strain isotropic elasticity, *Int. J. Num. Meth. Engrg.*, **37**, 1981–2004, 1994.

ODEN, J. T., *Finite Elements of Nonlinear Continua*, McGraw-Hill, 1972. Also Dover Publications, 2006.

OGDEN, R. W., *Non-Linear Elastic Deformations*, Ellis Horwood, 1984.

PERIĆ, D., OWEN, D. R. J., and HONNOR, M. E., A model for finite strain elasto-plasticity based on logarithmic strains: Computational issues, *Comput. Meths. Appl. Mech. Engrg.*, **94**, 35–61, 1992.

REDDY, J. N., *An Introduction to Nonlinear Finite Element Analysis,* Oxford University Press, 2004.

SCHWEIZERHOF, K. and RAMM, E., Displacement dependent pressure loads in non-linear finite element analysis, *Compt. & Struct.*, **18**, 1099–1114, 1984.

SIMMONDS, J. G., *A Brief on Tensor Analysis*, Springer, 2nd edition, 1994.

SIMO, J. C., A framework for finite strain elasto-plasticity based on a maximum plastic dissipation and the multiplicative decomposition: Part 1. Continuum formulation, *Comput. Meths. Appl. Mech. Engrg.*, **66**, 199–219, 1988.

SIMO, J. C., Algorithms for static and dynamic multiplicative plasticity that preserve the classical return mapping schemes of the infinitesimal theory, *Comput. Meths. Appl. Mech. Engrg.*, **99**, 61–112, 1992.

SIMO, J. C. and HUGHES, T. J. R., *Computational Inelasticity*, Springer, 1997.

SIMO, J. C. and ORTIZ, M., A unified approach to finite deformation elastoplastic analysis based on the use of hyperelastic constitutive equations, *Comput. Meths. Appl. Mech. Engrg.*, **49**, 221–245, 1985.

SIMO, J. C. and TAYLOR, R. L., Quasi-incompressible finite elasticity in principal stretches. Continuum basis and numerical algorithms, *Comput. Meths. Appl. Mech. Engrg.*, **85**, 273–310, 1991.

SIMO, J. C., TAYLOR, R. L., and PISTER, K. S., Variational and projection methods for the volume constraint in finite deformation elasto-plasticity, *Comput. Meths. Appl. Mech. Engrg.*, **51**, 177–208, 1985.

SPENCER, A. J. M., *Continuum Mechanics*, Longman, 1980.

WEBER, G. and ANAND, L., Finite deformation constitutive equations and a time integration procedure for isotropic, hyperelastic-viscoplastic solids, *Comput. Meths. Appl. Mech. Engrg.*, **79**, 173–202, 1990.

ZIENKIEWICZ, O. C. and TAYLOR, R. L., *The Finite Element Method*, McGraw-Hill, 4th edition, Volumes 1 and 2, 1994.