The development of finite element software for creep damage analysis

Dezheng Liu

A thesis submitted to the University of Huddersfield in partial fulfillment of the requirements for the degree of Doctor of Philosophy

School of Computing and Engineering

University of Huddersfield

February 2015

Abstract

Creep deformation and failure in high temperature structures is a serious problem for industry and is becoming even more so under the current increasing pressures of power, economics and sustainability. Laboratory creep tests can be used in the description of creep damage behaviour; however, it's usually expensive and time-consuming. Thus, the computer-based finite element (FE) technique is considered here for both time and economic efficiency.

This project aims to develop an in-house FE software for creep damage analysis. A novel in-house FE software High Temperature Structural Integrity (HITSI) was developed through the use of Continuum Damage Mechanics (CDM) and finite element method (FEM) in conjunction with an advanced engineering computer programming language (Fortran 2003) based on an objected oriented programming (OOP) approach.

This research provides four main contributions. First, a critical review of the current state of obtaining the computational capability for creep damage analysis. This critical review presents the advantages through the use of in-house software in analysing creep damage behaviour and the state-of-the-art research advancements and technologies need to be involved in developing in-house software. Second, the proposed OOP approach in design and development of in-house FE software for creep damage analysis. Third, the prototyping and implementation of a practical in-house FE software HITSI for analysing creep damage behaviour. The general flow diagram and development strategy of HITSI were proposed. Fourth, the benchmark test of HITSI via the numerical investigation of creep damage behaviour of a Cr-Mo-V steam pipe weldment case. The efficiency of the integration algorithms (Euler and Runge-Kutta) and normalized Kachanov-Rabotnov creep damage constitutive equation was investigated and commented.

Generally, this project provides a novel in-house software prototype that allow the scientist to simulate the behaviour of creep damage in particular to analysis the evolution of creep damage in welds.

Acknowledgements

I am sincerely grateful to my supervisors Dr Qiang Xu, Professor Zhongyu Lu and Dr Donglai Xu, for their guidance, numerous discussions and continuous encouragement over the course of this research. I would also like to thank them for providing the opportunities for attending international conferences and meetings during this study.

I would like to thank Dr Steve Andrews and Dr Tatyana Karpenko-Seccombe for their proof reading of the thesis.

The financial support provided by author's parents is gratefully acknowledged; without my mother and father's financial support and encouragement, this thesis would not have been finished.

I would also like to express my sincere thanks to Qihua Xu who never give up providing supports to my study.

Partial finance support from the School Scholarship scheme is also appreciated.

Finally, the author would like to thank colleagues and friends in this research group for their kindly encouragement.

Declaration

This dissertation is submitted for the degree of Doctor of Philosophy at the University of Huddersfield. I declare that the work in this dissertation was carried out in accordance with the Regulations of the University of Huddersfield.

The first two years of this research were conducted at Teesside University under the supervision of Dr Qiang Xu, Prof Zhongyu Lu, and Dr Donglai Xu, while Dr Qiang Xu was working at Teesside University.

This work is original except where acknowledgment and references are made to the previous work. Neither this nor any substantially similar dissertation has been or is being submitted for a degree, diploma or other qualification at any other university.

List of Publications

1, Liu, D., Xu, Q., Lu, Z. and Xu, D. (2012) 'The review of computational FE software for creep damage mechanics', Advanced Materials Research, 510, pp. 495-499. ISSN 1662-8985

2, Liu, D., Xu, Q., Lu, Z. and Xu, D. (2012) 'Research in the development of computational FE software for creep damage mechanics', In: Proceedings of 18th International Conference on Automation and Computing (ICAC), Loughborough, UK, 8 September 2012, IEEE. ISBN 978-1-4673-1722-1

3, Liu, D., Xu, Q., Lu, Z., Xu, D. and Tan, F. (2013) 'The development of finite element analysis software for creep damage analysis', In: Proceedings of the 2013 World Congress in Computer Science and Computer Engineering and Application, Las Vegas, USA, 22-25 July 2013, CSREA Press. ISBN 1-60132-238-0

4, Tan, F., Xu, Q., Lu, Z. and Liu, D. (2013) 'Practical guidance on the application of R-K integration method in finite element analysis of creep damage problem'. In: Proceedings of the 2013 World Congress in Computer Science and Computer Engineering and Application, Las Vegas, USA, 22-25 July 2013, CSREA Press. ISBN 1-60132-238-0

5, Liu, D., Xu, Q., Lu, Z., Xu, D. and Xu, Q. (2013) 'The techniques in developing finite element software for creep damage analysis', Advanced Materials Research, 744, pp. 199-204. ISSN 1662-8985

6, Liu, D., Xu, Q., Lu, Z., Xu, D. and Tan, F. (2013) 'The validation of computational FE software for creep damage mechanics', Advanced Materials Research, 744, pp. 205-210. ISSN 1662-8985

7, Tan, F., Xu, Q., Lu, Z., Xu, D. and Liu, D. (2013) 'The validation of computational software system for creep damage analysis', Advanced Materials Research, 744, pp. 449-454. ISSN 1662-8985

8, Liu, D., Xu, Q. and Lu, Z. (2013) 'Research in the development of finite element software for creep damage analysis', Journal of Communication and Computer, 2013 (10), pp. 1019-1030. ISSN 1548-7709

9, Liu, D., Xu, Q., Lu, Z. and Barrans, S. (2013) ‘The development of finite element software for creep deformation and damage analysis of weldment’, In: 6th International ‘HIDA’ Conference: Life/Defect Assessment & Failures in High Temperature Plant, Nagasaki, Japan, 2-4 December 2013, pp. 1-10.

Contents

List of Tables

List of Figures

List of Codes

List of Abbreviations

BM	Base Metal
CDM	Continuum Damage Mechanics
CG	Coarse Grain
ECCC	European Creep Collaborative Committee
EI	Explicit-Implicit
EPRI	Electric Power Research Institute
ETD	European Technology Development Ltd
FE	Finite Element
FEM	Finite Element Method
FG	Fine Grain
GCC	German Creep Committee
GE	General Electric
HAZ	Heat Affected Zone
H-D	Harper Dorn
HITSI	High Temperature Structural Integrity
IC	Inter-critical
MMRE	Mean Magnitude of Relative Error
N-H	Nabarro-Herring
ODE	Ordinary Differential Equations
OOP	Objected Oriented Programming
RHS	Right Hand Side
UMAT	User-defined Material Routine
UMIST	University of Manchester Institute of Science and Technology
WM	Weld Metal

Chapter 1 Introduction

1.1 Project Background

Creep in the temperature field of structural components has been of interest to engineers for over 100 years. Demands of thermal efficiency lead to an increase in the operating temperature of structural components so that such components experience more creep deformation and damage leading to rupture. The fields where the creep phenomenon has been of importance in the interpretation of the structural response are the design and construction of nuclear power plants, gas turbine engines, refining and chemical plants, heat exchangers and jet engines (Wilson and Korakianitis, 2014). For safe design and operation, as well as for better design and the development of new creep resistant components, current research institutes (such as General Electric (GE), Electric Power Research Institute (EPRI), University of Manchester Institute of Science and Technology (UMIST), European Technology Development Ltd (ETD), German Creep Committee (GCC) and European Creep Collaborative Committee (ECCC)) have utilized computer-based FE method or experimental method in the investigation of creep damage behaviour in structural components (Kim et al., 2002). However, some issues are still existing in current methods in the analysis of creep damage problem, and the issues can be classified as:

1) Experimental method in creep damage analysis is usually expensive and time-consuming, for example, the temperature field of structural components used in power plants and gas turbine engines are extremely expensive and are expected to last for 20 years or more, which makes experimental measurement unacceptable (Hyde et al., 1993).

2) The standard commercial FE software can only analyse the primary-secondary creep stage but are unable to simulate the tertiary stage where significant damage occurs since the CDM approach is not readily available in standard commercial FE software. The ECCC has reported that primary-secondary creep occupied only about 20% of the total specimen (low chrome alloy) life under the high stress level, whilst tertiary behaviour accounted for almost 80% of the test (Panait et al., 2010). Hence, the tertiary creep stage should be considered in the FE modelling.

3) Though there are a few research groups such those of Hayhurst et al. (1984), Becker et al. (1994) and Wong (1999) have reported the development and the use of their in-house FE software for creep damage analysis, the source codes of their in-house FE software have not published. On the other hand, the OOP approach has not been considered within their in-house FE software. It is a time-consuming task to maintain the FE codes for the analysis of high non-linearity creep damage problem, whereas the OOP approach can overcome limitations such as the complexity and unitary programming procedures of procedural programming at no additional cost (Mackie, 2008). It needs to consider efficiency of the programming and maintenance of FE codes when the development of in-house FE software is deployed.

To overcome the issues stated above, the development of a novel dedicated in-house FE software should be considered for its efficiency and functionality in creep damage analysis. CDM and FEM in conjunction with an advanced engineering computer programming language (NAG Fortran 2003) based on an OOP approach can provide a way to develop in-house FE software that allow the scientist to simulate the creep damage evolution and the lifetime of high temperature structural components. The main challenges in developing such FE software are dealing with the time dependent high non-linearity behaviour, the stress redistribution and the multi-material zones.

1.2 Aims and Objectives

This project aims to develop the in-house FE software HITSI to describe the creep damage behaviour of the temperature field of structural components and predict the lifetime of such structural components. The target estimation accuracy of HITSI in the prediction of creep failure time was designed based on the Mean Magnitude of Relative Error (MMRE) (Briand and Wieczorek, 2002), which is the most widely used evaluation criterion to assess the performance of software prediction models. According to the estimation in software engineering (Conte et al., 1986), HITSI considers relative error ≤ 0.25 as acceptable for effort prediction models. The HITSI will focus on modelling the creep damage fields in the design and safety evolution stages in particular to analysis the evolution of creep damage in welds. This software can also be used as a research platform for helping researchers to validate the new creep damage constitutive equations and numerical time integration methods.

It is envisaged that the research will contribute to the domain knowledge in creep deformation, creep fracture and the state-of-the-art research advancements and technologies in computational creep damage mechanics. Also it will contribute to the development strategy, specifications of FE technologies, programming processes and verification procedures in developing in-house FE software for creep damage analysis.

In order to fulfill above aims the specific objectives are detailed below, and include:

1) a thorough understanding of the mechanisms of creep, the requirements and measurement techniques in preparing for creep damage analysis, and to investigate the current state of how to achieve computational capability for creep damage analysis.

2) a general methodology for developing HITSI, and detail the specific techniques used in spatial discretisation by finite elements, element stiffness integration and assembly, solution of equilibrium equation and recovery of results at integrating points.

3) an integrated framework to encompass the development of HITSI for creep damage analysis and to develop this software in accord with the development strategy (linear elastic stage, non-linear elastic-plastic stage and creep damage stage).

4) a consistent strategy to validate HITSI for the analysis of creep damage behaviour, and to validate the FE codes in a step by step fashion according to the development strategy for consistency and integrity of this project.

5) a benchmark test of HITSI via the numerical investigation of creep damage behaviour of a Cr-Mo-V steam pipe weldment case, and to investigate the efficiency of the Euler and Runge-Kutta integration methods and the normalized Kachanov-Rabotnov creep damage constitutive equation.

6) user guidance for HITSI, and to develop the instructions and tutorials for users and researchers.

At present, all of the above six objectives are delivered in this thesis. After achieving the above six objectives, the in-house FE software HITSI will enable engineers and researchers to use computer-based FE method in modelling creep damage behaviour in

structural components. Here, some limitations in the use of HITSI should be mentioned and outlined as: 1) weldment structure such as butt-welded pipework which contains pipe intersections and branches cannot be modelled by HITSI unless the complex three-dimensional element types such as tetrahedron element are developed; 2) the FE codes in HITSI can be compiled by either the NAG Fortran compiler or Code::Blocks on Windows/ Linux systems; however, programs compiled with a 64-bit system will only run on 64-bit kernels with object sizes limited to 2 GB and program compiled with a 32-bit system will only run on x86 platform with object sizes limited to 2 GB.

1.3 Project Approach

This project started with an extensive literature review of the sate-of-the-art in mechanisms of creep deformation and creep fracture, knowledge-based weldment component, CDM approach, FE algorithm, OOP approach, current FE software applications and numerical integration scheme practices. Based upon the domain research outlined in literature review, the development of the in-house software HITSI will use the following development process:

1. **Initial Design**. The overall project framework will be produced that takes into account all the aspects of problem domains and requirements in developing in-house FE software for creep damage analysis. To address problems highlighted in Section 1.1, CDM and FEM in conjunction with Fortran 2003 programming language based on an OOP approach are used in the development of this FE software.

2. **Proof-of-Concept Development.** A proof-of-concept consideration was developed in order to understand implementation issues and develop solutions to those issues. This consideration also served as a demonstration of the design concepts and capabilities of the final system and allows various tests to evaluate the software system by researchers.

3. **Design Modification**. After analysing and evaluating the findings from the various tests, the design on both the conceptual model and the functionality of software was refined. In this project, it was originally only aiming at the development of a 2D (plane stress, plane strain and axisymmetric) version. With successful progress on 2D version and recognising the practical importance of a

3D version (a more general version), the 3D version of the software was proposed that needs to be developed.

4. **Development and Implementation.** The development of this in-house FE software will be undertaken with as much adherence to the design as possible, although further issues may arise that require modification to the design. It includes the linear elastic stage, the non-linear (single material and time independent) elastic-plastic stage and the time dependent creep deformation and creep damage stage. In this project, the existing standard FE subroutines (Smith and Griffiths, 2005) and a specific subroutine library (Feng Tan's FE library) were utilized in order to make the development of HITSI more efficient.

5. **Testing and Validation.** The developed software will be tested and validated to ensure its performance and efficiency satisfy the requirement this research. In this project, the FE simulated results from HITSI (uni-axial case) were compared with the theoretical results to demonstrate the validity of the FE program and a benchmark test of HITSI was performed via the numerical investigation of creep damage behaviour of a Cr-Mo-V steam pipe weldment case (multi-axial case). The expert in computational creep damage mechanics in the University of Huddersfield will be invited to assess the software to ensure that it meets the aim of the project and is of an appropriate level of quality.

1.4 Arrangements of the Thesis

Chapter 1 introduces the need for computational capability in creep damage analysis and the justification for the development of the in-house FE software HITSI. The aims and objectives are described. The research approaches are also demonstrated.

Chapter 2 presents a review of literature in the context of the various knowledge domains relating to this project. The contents of literature review mainly focuses on the mechanisms of creep deformation and creep fracture for understanding the creep behaviour, the development of the FEM based CDM approach for obtaining the computational capability for creep damage analysis, the characteristics of existing standard commercial and in-house FE software relative to computational creep damage mechanics and the preference for in-house software. The investigation of the numerical integration methods for the analysis of creep damage behaviour is also illustrated.

Chapter 3 focuses on developing a unified FE algorithm for the development of the in-house FE software HITSI and describing FE techniques that will be involved in this project. FE techniques such as spatial discretisation by finite elements; element stiffness integration; element stiffness assembly; solution of equilibrium equation and recovery of results at the integrating points are discussed. Then, the relevant existing standard FE subroutines (Smith and Griffiths, 2005) which can be used in the development of HITSI are introduced and demonstrated.

Chapter 4 presents the development of HITSI. This chapter starts with a development strategy for ensuring the programming of HITSI in a step by step fashion, as well as to be logical and efficient. Then, it moves to the actual programming stage. The development of HITSI contains the linear elastic stage, the non-linear (single material and time independent) elastic-plastic stage and the time dependent creep damage stage (plane stress, plane strain, axisymmetric and 3D version). Each development stage in turn contains the correlative FE programs (sub-knowledge bases) and the specific FE techniques and necessary functional extensions in each FE program are presented and discussed in this chapter.

Chapter 5 presents the validation of the FE codes for HITSI. The validation procedures are corresponding to the development stages from linear elastic stage to creep damage stage. The validation of each FE program is conducted through the comparisons between the FE simulated results from the uni-axial case and the correlative theoretical results. The details of each validation case are described. The results and discussions in each FE program validation are presented.

Chapter 6 presents the benchmark test of HITSI via the numerical investigation of creep damage behaviour of a Cr-Mo-V steam pipe weldment case. The 2.25Cr1Mo:0.5Cr0.5Mo0.25V steam pipe weldment physical case (Hall and Hayhurst, 1991) is modelled by HITSI and the FE simulated results are compared with the laboratory test (Coleman et al., 1985) and the results from the software Damage XX (Hall and Hayhurst, 1991), respectively. Furthermore, the efficiency of the Euler and Runge-Kutta integration methods and the normalized Kachanov-Rabotnov creep damage constitutive equation are investigated via this case study.

Chapter 7 focuses on the summary of this research and contribution to knowledge. A discussion for the future work is also included at the end of this chapter.

Chapter 2 Literature Review

2.1 Introduction

This chapter is a review of literature covering creep damage behaviours, the associated metallurgy, the FEM based CDM approach for creep damage analysis, the current state of how to obtain the computational capability for creep damage analysis and the numerical integration scheme used in the development of the in-house software HITSI.

The following specific areas of knowledge are considered in detail:

1) To review the mechanisms of creep deformation. The two main creep processes (dislocation creep and diffusion creep) involved in the mechanisms of creep deformation are examined to understand the nature of creep deformation behaviour.

2) To review the mechanisms of creep fracture in metals and alloys. Two fracture classes (creep failure at temperature above one third melting point and creep failure at temperature under one third melting point) involved in the mechanisms of creep failure are reviewed to understand the nature of creep failure behaviour.

3) To review the weldment components. The weldment zones, the creep properties in weldment material zones and the creep failure types of weldment components are reviewed.

4) To review the development of the FEM based CDM approach for creep damage analysis. The development of CDM for creep damage analysis is considered first; then, the existing FE algorithms are reviewed and the preference for the explicit FE algorithm is demonstrated. It further reviews the advantages of the OOP approach in programming FE software.

5) To review the existing standard commercial FE and in-house FE software. The advantages and disadvantages of existing FE software for creep damage analysis are commented upon and it concludes with a preference for in-house FE software.

6) To review the existing numerical integration schemes for the analysis of creep damage. The Euler and Runge-Kutta schemes are reviewed first concluding with a preference for the Runge-Kutta scheme.

2.2 Mechanisms of Creep Deformation in Metals and Alloys

Creep is defined as the time dependent plastic deformation of a material experiencing constant load. There are two main creep processes involved with the mechanism of creep deformation (Ashby and Brown, 1983). The first process is called dislocation creep, in which the factor controlling the creep rate is the ability of dislocations to glide. The second process is called diffusion creep, in which the factor controlling the creep rate is continuous annealing at high temperatures (Svensson and Dunlop, 1981). These two creep processes are inevitably interconnected, as they may both happen at the same time.

The mechanism of creep may be controlled through the diffusion of vacancies or by motion of dislocations and it depends on the different levels of temperature and stress. In order to identify the mechanism of creep, Gollapudi (2007) summarized the previous work and reported that the particular mechanism of creep can be identified through knowledge of the grain size exponent, the stress exponent and the activation energy.

2.2.1 Dislocation Creep

Dislocation creep is a process involving the motion of dislocations through the crystal lattice of the material (Poirier, 1985). Dislocation creep can exist in whole creep process stages and the deformation tends to dominate with differential stress levels on the material and relatively low temperatures. Dislocations may move from one slip plane to another, by the mechanism known as cross-slip, which allows dislocations a further degree of freedom (Poirier and Nicolas, 1976). In dislocation creep processes, Harper Dorn (H-D), Viscous glide and Dislocation climb are mechanisms of creep that fall under the category of dislocation based processes (Gollapudi et al., 2008).

The key features of the mechanisms of creep under the category of dislocation based processes can be classified as follows:

a) The Harper-Dorn creep mechanism: It was first proposed by Harper and Dorn (1957) and this mechanism can be rate controlling at intermediate temperatures. The creep rate may be controlled either by lattice diffusion or by dislocation core diffusion (Rollason, 1973). Moreover, the stress and creep rate are

independent of grain size and similar creep rates are observed both in polycrystals and single crystals (Cadek, 1988).

b) Viscous glide: This mechanism is usually exhibited by alloys, and the dislocation velocity in this case is controlled by the rate of migration of the solute atoms (Cottrell and Jaswon, 1949). In the viscous process, the dragging force is an outcome of solute atoms segregating to stacking faults and the ordering of the region surrounding a dislocation reduces the total energy of the crystal by pinning the dislocation (Gollapudi et al.2008).

c) Dislocation climb: The earliest model to describe creep by dislocation climb was proposed by Weertman (1955); this mechanism considers the creep processes to be a result of the glide and climb of dislocations. Later, another model that considered the non-conservative motion of dislocations was proposed by Barrett and Nix (1965). These two models are similar in the sense that the rate of climb of the edge jogs is dependent on the concentration gradient established by the climbing jogs. However, Viswanathan et al. (1999) indicated that these jogs in the above two models could be several times larger than atomic dimensions and a modified jogged screw model was proposed by Viswanathan et al. (1999) to depict the behaviour of creep.

There are two types of dislocation (Bauer, 1965): the edge dislocation in Figure 2.1 and the screw dislocation in Figure 2.2.

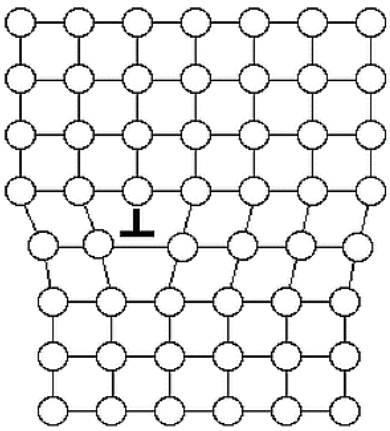


Figure 2.1: The edge dislocation (Bauer, 1965)

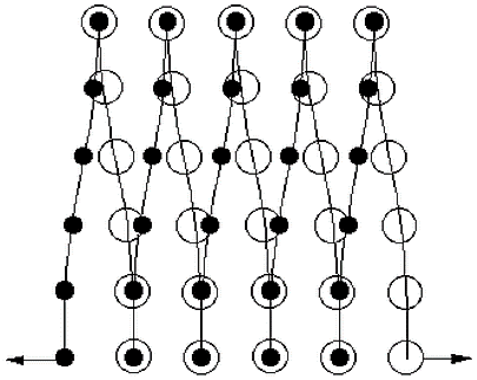


Figure 2.2: The screw dislocation (Bauer, 1965)

The edge dislocation and the screw dislocation can both contribute to the creep deformation. The schematic diagram of edge dislocation and screw dislocation are shown in Figure 2.1 and Figure 2.2, respectively. Edge dislocations form the edge of an extra layer of atoms inside the crystal lattice and they move in the direction of the Burgers vector (Callister, 2001), whereas screw dislocations form a line along which the crystal lattice jumps one lattice point and they move in a direction perpendicular to the Burgers vector (Poirier and Nicolas, 1976). Both edge dislocation and screw dislocation lines form a linear defect through the crystal lattice and the crystal can be intact on all sides of the line. When the distortion is spread over a large area, the movement of the dislocation becomes easier. Such dislocations can be called wide dislocations, and they normally exist in ductile metals.

Dislocation creep can be represented by Equation 2.1 and the secondary creep strain rate is dependent on the applied stress raised to a power n. This equation is known as the Norton Law (Norton, 1929).

$$\dot{\varepsilon}_s = K\sigma^n \tag{2.1}$$

Here $\dot{\varepsilon}_s$ is creep strain rate, K is a material constant, σ is stress and n is a variable known as the creep exponent. The variable n usually has a value between 1 and 10 (Norton, 1929).

2.2.2 Diffusion Creep

The process of diffusion creep was first considered for the deformation of crystalline solids by the diffusion of vacancies through their crystal lattice by Nabarro (1948) and Herring (1950). Later, Coble (1963) proposed that grain boundaries can also provide an alternative path for stress directed diffusional mass transport to take place. In diffusion

creep processes, Coble and Nabarro-Herring (N-H) are mechanisms of deformation that fall under the category of diffusion based processes (Gollapudi, 2007). The schematic diagrams of the mechanism of Nabarro-Herring (bulk diffusion) and Coble (grain boundary diffusion) are shown in Figure 2.3 and Figure 2.4, respectively (Goretta et al., 2001).

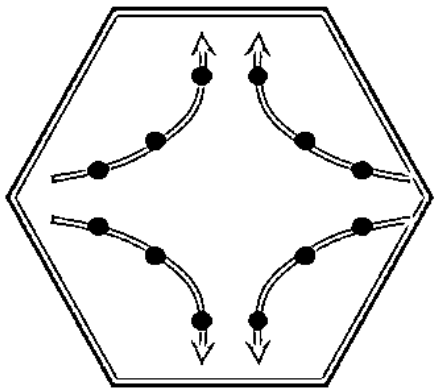


Figure 2.3: Nabarro-Herring diffusion (Goretta et al., 2001)

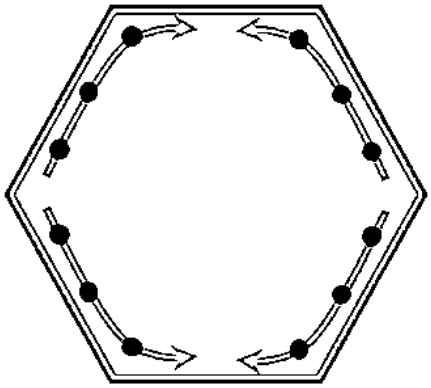


Figure 2.4: Coble diffusion (Goretta et al., 2001)

The key features of the mechanisms of creep in the category of diffusion based processes can be classified as follows:

a) Nabarro-Herring creep mechanism: It was first proposed by Nabarro (1948) and Herring (1950); and this mechanism considers the possibility of creep occurring by stress assisted diffusional mass transport through the lattice. The process of this mechanism is controlled by stress-directed atomic diffusion through the bulk of a metallic crystal. Atoms move through metallic crystals towards grain boundaries under tensile stress and conversely vacancies move towards grain boundaries under compressive stress.

b) Coble creep mechanism: The Coble creep is also called grain boundary diffusion and it was first proposed by Coble (1963); this mechanism considers that grain boundaries can also provide an alternative path for stress directed diffusional mass transport to take place based on the Nabarro-Herring creep mechanism.

Nabarro-Herring diffusion and Coble diffusion can contribute to the deformation of creep simultaneously. The diffusion of vacancies or the motion of atoms from one grain boundary to another could occur through the lattice (Nabarro-Herring) or via grain boundaries (Coble); however, with increasing temperature Nabarro-Herring creep has a greater tendency to become the rate controlling mechanism (Gollapudi, 2007).

2.3 Mechanisms of Creep Fracture in Metals and Alloys

Creep fracture is usually caused by the growth of nucleation and mutual connection of micro-cavities and micro-cracks (Riedel, 1987). With the continued growth of voids, creep cracks grow from the cusp and ultimately weaken the cross section to the point where failure occurs (Kun et al., 2003). The crystalline solids can fracture by one of several mechanisms. The following description of fracture mechanisms is in accordance with Ashby (1972), Frost and Ashby (1982) and Riedel (1987).

The fracture mechanism map as an effective way of representing the fracture model at any combination of stress and temperature was first proposed by Ashby (1977). The map indicates the different fracture mechanisms of creep operating in a material as a function of stress, temperature and grain size.

The fracture mechanism maps such as Ashby (1977), Ashby et al. (1979) and Krishnamohanrao et al. (1986) have been summarized by (Riedel, 1987). A schematic diagram of the fracture-mechanism map was proposed by Riedel (1987) and is shown in Figure 2.5.

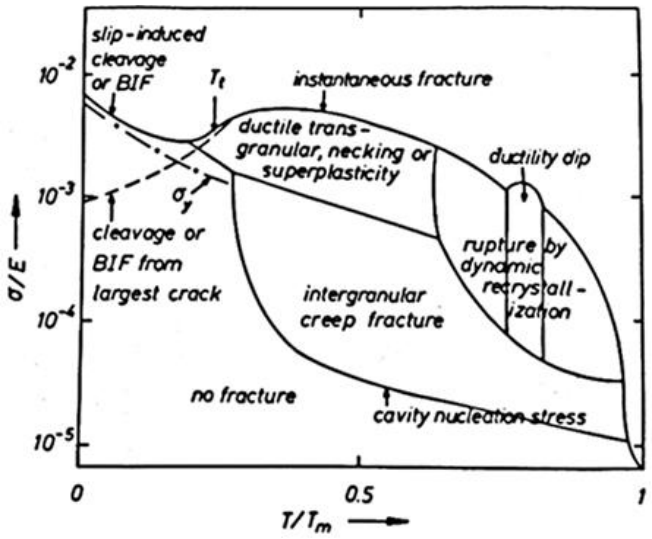


Figure 2.5: The schematic diagram of fracture mechanism map (Riedel, 1987)

The different regions are presented for a range of stress and temperature over which a specific mechanism is anticipated to be the principle process of creep (Liu, 2005). The fracture mechanisms can be described under the category of temperature level.

The mechanisms appearing on the fracture mechanism map in this category can be classified as follows (Ashby et al., 1979):

i. Creep fracture mechanism at temperatures above one third the melting temperature of the material.

ii. Creep fracture mechanism at temperatures below one third the melting temperature of the material.

2.3.1 Creep Fracture Mechanism at Temperatures above One Third Melting Point

In metals and alloys which creep at temperature above one third melting point, the key features of the fracture mechanisms under the category of temperatures beyond this point can be classified as follows:

a) Transgranular creep fracture: Transgranular creep fracture requires either that voids pre-exist or voids nucleate at inclusions that concentrate stress (Ashby et al., 1979). The size of voids grows by creep deformation around inclusions, elongating them in the direction where the stress is applied and the flow stress is determined by the strain rate, which is governed by the creep power-law (Hayhurst, 2006).

b) Intergranular creep fracture: Intergranular creep fracture is usually found at lower stresses and elevated temperatures. In this fracture mechanism, void growth by creep becomes so slow that fracture by grain boundary cavitation intervenes (Riedel, 1987). Garofalo (1965) reported that the shear deformation at grain boundaries observed in intergranular creep was much higher than that in transgranular creep. Creep void growth is controlled by dislocation creep at the primary-secondary creep stage when voids are small, whereas the diffusion creep also contributes to void and crack growth synchronously (Hayhurst, 2006).

c) Pure diffusional fracture: Pure diffusional fracture is usually found at lower stresses and high temperature. In this fracture mechanism, the stress is so low that the power-law creep can be negligible and the voids on the grain boundaries grow by the mechanism of diffusion alone (Hall, 1990). With the growth of the cavities, this type fracture will move to either intergranular fracture or transgranular fracture.

2.3.2 Creep Fracture Mechanism at Temperatures below One Third Melting Point

In metals and alloys which creep at temperature below one third melting temperature, the key features of the fracture mechanisms under this category can be classified as follows:

a) Cleavage: Cleavage creep fracture is usually found at low temperatures and high stresses. This fracture mechanism is usually initiated by plastic slipping or twinning, often where a slip band impinges on a coarse carbide particle (Riedel, 1987). The cracks concentrate stress, and the formation of cracks and their propagation, along certain crystallographic planes (Rice and Thomson, 1974).

b) Ductile failure: Ductile failure is usually found at low temperatures and the process of this fracture mechanism is similar to the transgranular creep fracture. Voids nucleate at inclusions and the plasticity promotes their growth. When voids grow big enough, they may coalesce and trigger the fracture of components. A new void is nucleated and it connects with other voids could result in fracture. Ductile fracture usually accompanies transgranular fracture. However, it may accompany intergranular fracture if the void density becomes higher in the boundaries (Hayhurst, 2006).

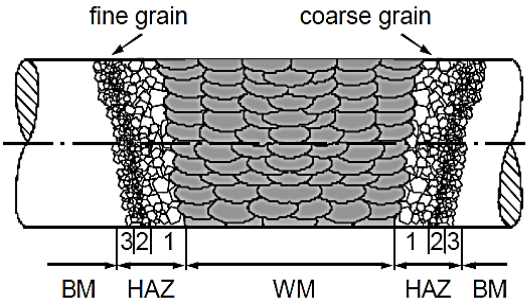
2.4 The Weldment Component

The weldment has been widely used in high temperature industry fields such as the construction of electrical power plant, gas turbine engines design, and refining and chemical plants design. The welding processes (Murti and Sundaresan, 1985) can provide a strong but straightforward and cost effective joint between components. This process can reduce the requirements for bolted flanges and seals; for example, fusion welding is the most important method for the design of high temperature power plant.

The main characteristic of the weldment component is the multi-material zones. ECCC (Holdsworth, 2008) and GCC (Kern et al., 2004) have reported that the creep damage in the welding area is usually more serious and the weldment response is further complicated due to the different base materials that are joined. Hence, the weldment zones, the microstructure and creep property behaviour in weldment zones and the creep rupture types of weldment are reviewed to understand the characteristic of weldment in this section.

2.4.1 Weldment Zones

The weldment can be divided into a number of different zones, which are weld metal, HAZ (coarse grain, fine grain and inter-critical) and parent material. A typical diagram (Klenk et al., 2003) is shown in Figure 2.6 to describe the different weldment zones.



Shortcut	Material
WM	Weld
CG	HAZ 1 (coarse grain)
FG	HAZ 2 (fine grain)
IC	HAZ 3 (intercritical)
BM	Parent

Figure 2.6: The diagram of five typical weldment zones (Klenk et al., 2003)

A weldment typically consists of five material zones, as illustrated in Figure. 2.6, the components in the diverse material zones of weldment are different (Klenk et al., 2003). The region close to the weld metal fusion boundary is called the heat affected zone (HAZ), which depends on the phase transformations, grain growth and refinement.

The evolution of microstructure will vary across the weldment as well as within the beads during welding, and the evolution is essentially controlled by the heat cycle that

the material experiences and the features of the material such chemical composition and microstructure of the base material (Hyde et al., 1999).

The stresses in cross-section are redistributed due to the thermal effect in the welding process (Hyde and Sun, 2002). Furthermore, the residual stresses can be reduced and the mechanical properties of the weldment constituents can be changed with a subsequent post weld heat treatment (Segle, 2002).

2.4.2 Creep Properties in Weldment Material Zones

Creep properties in weldment material zones vary with the type of microstructure. Furthermore, the microstructure of the weldment also varies with the welding processes; for example, the fusion between parent material and weld metal occurs with a heat treatment and the fusion becomes complex due to the changes of temperature during the welding processes (Hayhurst, 2006). The schematic diagram of temperature and grain growth relationship in a typical ferritic steel weld is shown in Figure 2.7 (Porter and Easterling, 1992).

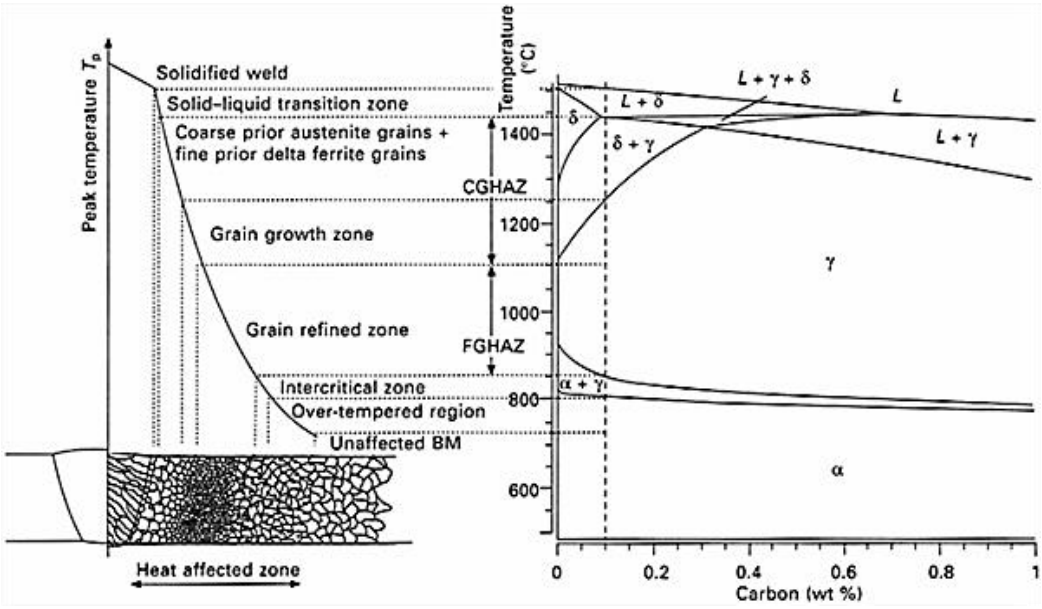


Figure 2.7: Schematic diagram of temperature and grain growth relationship in a typical ferritic steel weld (Porter and Easterling, 1992)

Segle (2002) has summarized the relationship between the type of microstructure and the material zones in weldment. Material properties such as tensile strength, yield strength, fatigue strength, hardening, fracture toughness, hardness, creep deformation rate, creep rupture strength and creep ductility vary with the type of microstructure (Coleman et al., 1998). According to the Figure 2.7, the material zones can be

distinguished across the weldment, starting from the centre of the weld: weld metal, fusion line, coarse grained HAZ, fine grained HAZ, inter-critical HAZ and base material (Segle, 2002).

The main characteristics of creep properties in weldment material zones are reviewed in Table 2.1.

Table 2.1: The main characteristics of creep properties in weldment material zones

Weldment zone	**The main characteristics of creep properties**
Weld metal	The creep deformation rate is instability; the creep rupture strength, the creep ductility and rupture behaviour in weld metal are similar to that of the base material (Parker and Parsons, 1995)
Coarse grained HAZ	Lower creep deformation rate, higher creep rupture strength and lower creep ductility than that of the base material (Lee et al., 1989)
Fine grained HAZ	The fine-grained zone of HAZ contains higher density of dislocations than that of the base metal (Matsui et al., 2001). The creep deformation rate, creep rupture strength and creep ductility are similar to that of the base material (Parker and Parsons, 1995)
Inter-critical HAZ	Higher creep deformation rate, lower creep rupture strength and higher creep ductility than that of the base material (Segle, 2002)
Base (parent) material	Softening mechanism in this weld zone is known to be a process of creep cavitation and coarsening of the carbide precipitates (Parker and Parsons, 1995)

The different creep properties in weldment material zones will lead to the generation of stress redistribution when the weldment is set in operation. In the development of FE software for creep damage analysis of weldment, the multi-material zones program version should consider the material zones with different creep. Moreover, the stress and damage field variables should be updated at each iteration loop.

2.4.3 Weldment Failure Types

The failure types of weldment are related to both the range of microstructures developed during the welding processes and the effect of long-term, high temperature exposure on

micro-structural changes (Tu et al., 1994). The classification of cracking types in the weldment is presented in Figure 2.8 (Coleman and Kimmins, 1990).

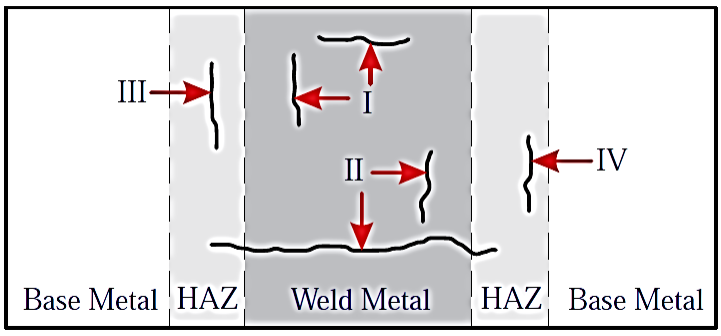


Figure 2.8: The classification of cracking types (Coleman and Kimmins, 1990)

The failure appears to be the result of the heterogeneous microstructure developed during the welding process leading to markedly different properties within the base material, HAZ and weld metal (Riedel, 1987). The direction and rate of the crack growth depend on factors such as stress level and stress state, the material properties in the cracked area and the ability to off-load the crack by stress redistribution (Segle, 2002). With the continued growth of voids, creep cracks grow from the cusp and ultimately weaken the cross section to the point where failure occurs (Kun et al., 2003). Coleman and Kimmins (1990) have classified the cracking types in weldment material zones.

Table 2.2: Cracking types in weldment material zones (Coleman and Kimmins, 1990)

Cracking type	**Positions in weldment zone**
Type I	in weld metal
Type II	in weld metal and adjacent HAZ
Type III	in coarse grained HAZ
Type IV	in inter-critical HAZ

Depending on the position in weldment material zones, cracking types are defined in Table 2.2 (Coleman and Kimmins, 1990). Type I and II cracking are often associated with the initiation of cracks in weld metal. The lower creep ductility in the coarse grained HAZ in conjunction with the lower creep deformation rate result in an enhancement in stress level due to stress redistribution during the operation; Type III cracking is generated by this evolution (Segle, 2002). Type IV cracking develops in the

inter-critical HAZ (Kimmins et al., 1996). The material in the inter-critical HAZ is typically characterised by a relatively lower creep strength (Viswanathan, 1989), higher minimum creep rate and creep ductility (Parker and Stratford, 1996). Type IV cracking is caused by additional loading or stress conditions being applied to the component which generate stresses normal to the Type IV zone. In operation, the additional stresses may be generated by the bending of pipes under their own weight and the constraint forces generated by brackets (Hayhurst, 2006).

In the development of FE software for the creep damage analysis of weldment, the creep deformation and damage are integrated with regard to time. Accuracy suffers and instability may occur if the time increment is too large because the evolution of cracking types in different weldment material zones is extremely sensitive and complex. Thus, the selection of the size of time step associated with an appropriate numerical integration method is very important. The creep damage increases monotonically with time until the damage increases from the initial zero value to the critical value. The element cannot then support any further load and as such is called a failed element. The program should remove the failed element to ensure the accuracy.

2.5 The Development of the FEM based CDM approach for Creep Damage Analysis

The FEM based CDM approach provides the possibility to model and analyse the creep damage behaviour in engineering structures. Becker et al. (2002) reported that the characteristics of the primary-secondary (steady state) creep deformation behaviour can be observed by experiment or simulated by standard commercial FE software in conjunction with a special user routine such as UMAT; however, the mathematical description of tertiary creep can only be described through the use of CDM. Here, the in-house FE software is developed for creep damage analysis through the use of CDM.

The specific theories in the development of the FEM based CDM approach software for creep damage analysis should be reviewed as follows:

i. The development of CDM for creep damage analysis.

ii. The FE algorithms used in developing FE software for creep damage analysis.

iii. The advantages of the OOP approach in programming FE software.

2.5.1 The Review of the CDM

The CDM approach is based on continuum mechanics where a damage parameter has been introduced by Kachanov (1958), Rabotnov (1969) and Murakami (1983). In creep CDM, the analysis of creep damage using the FEM based CDM approach has been used to predict the rupture lifetime of components and to investigate the initiation and growth of damage in structures. In FE applications, a damage parameter is defined that ranges from zero (no damage) to a critical damage value (full damage) and is then controlled throughout the creep processes. Creep failure time is defined as the time taken for the continuum damage level to move from no damage to full damage (Becker et al., 2002).

The CDM approach was first proposed by Kachanov (1958) and Rabotnov (1969), and was extensively developed by Hayhurst (1972), Hayhurst (1973), Leckie and Hayhurst (1977) and Hayhurst et al. (1984) for creep damage analysis.

The literature on CDM has now reached mature level. Initially, the CDM approach was developed for assessing the manufacture of components from single material; later on, it has been extensively used in the creep damage analysis of multi-material structures such as damage evolution in weldment. Other work on the description of creep damage evolution in weldment through the use of CDM was described by Riedel (1990), Hall and Hayhurst (1991), Wang and Hayhurst (1994), Murakami and Liu (1995) and Perrin and Hayhurst (1996b).

The advantage of CDM is the existence of a consistent derivation through the creep damage processes. The feature of CDM approach is the material gets damaged does not essential has to be understood in detail and the damage parameter can assess the damage level of creep (Penny and Marriott, 1995). The FEM combined with CDM approach has been demonstrated by Hall and Hayhurst (1991), Hall et al. (1996) and Hyde et al. (2000) to be an efficient method in developing in-house FE software for assessing the creep damage behaviour of the structural components.

2.5.2 The Review of FE Algorithm

FEM is a computer-aided engineering technique for obtaining approximate numerical solutions to boundary value problems which predict the response of physical systems subjected to external loads (Szabo and Babuška, 1991). In the development of FE software for creep damage analysis, the key challenge is dealing with the highly non-linear behaviour of creep. Factors such as the material inhomogeneity, the stress

redistribution due to tertiary creep and the multi-axial stress rupture criterion lead to a high non-linearity in creep damage analysis through the use of FEM based technique. Thus, numerical FE solution procedures to solve non-linear initial-boundary value problems must be developed in programming such software for creep damage analysis.

It should be noted that many numerical techniques such those of Lemaitre (1985), Chen and Hsu (1988), Krishnaswamy et al. (1995), Lemaitre and Desmorat (2005) and Cao et al. (2008) have previously been presented; however, all approaches can be fundamentally classified as the explicit FE algorithm, the implicit FE algorithm and the mixed explicit-implicit (EI) FE algorithm.

The key features of the FE algorithms in developing numerical FE solution procedures for the creep damage problem can be classified as follows:

a) The explicit FE algorithm: The explicit algorithm (Zienkiewicz and Cheung, 1967) involves an explicit relationship between increments of stress and increments of strain. The explicit procedure is based on the implementation of an explicit integration rule together with the use of diagonal element mass matrices. The equation of motion for the body is integrated using an explicit central difference integration rule (Sun et al., 2000). Internal and external forces are summed at each node point for all elements and this process is repeated at each iteration step. The main advantage of the explicit FE algorithm is that the analysis of non-linear problems through the use of FEM can be carried out element-wise and no global system storage is necessary. However, the stable time steps may need to be very small to avoid a potential loss of stability (Smith et al., 2013).

b) The implicit FE algorithm: The implicit algorithm (Lemaitre, 1985) assumes that the state of damage of the structure does not influence the state of stress or strain. The implicit procedure uses an automatic increment strategy based on the success rate of a full Newton iterative (Lemaitre, 1972). The main advantage of this algorithm is that the time step size can be selected by user. However, a large numerical effort is required to form, store and factorize the stiffness matrix. The local instabilities make force equilibrium difficult to achieve and as a result the unconditionally stable implicit method will encounter some difficulties in analysing the complicated FE model (Rebelo et al., 1992).

c) The mixed explicit-implicit FE algorithm: The mixed explicit-implicit algorithm is a methodology to combine explicit and implicit linear integration approaches based on element-wise stability considerations (Fierz et al., 2011). The improvements have been shown by Chen and Hsu (1988) to achieve stability and accuracy of results by using the mixed explicit-implicit algorithm. With the mixed explicit-implicit algorithm, much larger time step sizes can be employed with only slightly more computational effort than for the explicit scheme. However, the disadvantage of this algorithm for use in FEM for the highly non-linear problems is much more complexity and extra development work.

In FEM for highly non-linear dynamic problems, an explicit algorithm, which is conditionally stable, is the most adapted (Noels et al., 2004). The key feature of the analysis of creep damage problems through the use of FEM is dealing with the highly non-linear behaviour of creep. Especially in the tertiary creep stage, a large number of iterations are usually needed to achieve the stability and the accuracy of the FE solutions. In creep CDM, a damage parameter is defined to represent the continuum damage level from no damage to full damage and the size of time step is usually very small by comparison with the failure time of the components when describing the different damage levels in engineering structures. Thus, the many iterations will result in the use of a very large numbers of simultaneous equations for the solution of creep damage problem. In order to reduce the large storage demands and improve the efficiency of computational capability, the explicit FE algorithm is adapted in the development of HITSI for creep damage analysis.

2.5.3 The Review of OOP Approach

OOP approach is a relatively new philosophy of programming which using data structures consisting of data fields and methods together with their interactions to improve the overall quality of computer applications such as simulation programs, operating systems and graphical user interfaces (Machiels and Deville, 1997). In computer-based FE simulation, the first application of OOP to the FEM appeared at the end of the 1980s with the work by Rehak et al. (1989) and Forde et al. (1990). The authors abstracted out the essential components such as the element, the node, the boundary conditions and the loads information of the FEM as the basic objects of an OOP environment. Later on, Zimmermann et al. (1992), Miller et al. (1993) and Lages et al. (1999) described in detail the fundamental aspects relating application of OOP

techniques to implementation of the FEM. They also presented OOP architecture for use in non-linear dynamic FE analyses.

By comparison with the typical FE program developed based on the procedure-oriented approach, the FE program developed based on OOP approach has obviously advantages (Archer, 1996). The basis of OOP approach is abstraction. The application of OOP philosophy in the development of FE software can make the programming more flexible. Generally, the reasons for choosing OOP approach can be summarised as:

1) To apply a new algorithm or a new kind of element in the FE software system may easier because the alteration of one subroutine will not affect the whole program.

2) The efficiency in maintenance the FE program can be significantly improved because it allows the reusability of the FE codes.

3) The integrity and determination of the data structures are assured; thus it is easier to modify the existing FE codes and to extend the FE codes to adapt them for new uses, models and solution procedures.

It is noted that most of the relevant publications focus on computational aspects associated with OOP, rather than on actual engineering applications. The approach adopted in this project consists of employing OOP as a programming approach, which plays an important role in the development of in-house FE software HITSI for creep damage analysis. Although only a few research groups such as Hayhurst et al. (2009) and Hyde et al. (2000) had developed non-linear FE software in creep damage analysis, the OOP approach was not mentioned. Consequently, to develop in-house FE software based on OOP for creep damage analysis is still a new area, no distributed FE system based on OOP has been built for creep damage analysis so far.

The purpose of the use of OOP in current research is to make the software HITSI more flexible for further expansion, with low computing cost and high computing performance. For instance, in the FE module the different types of the creep damage constitutive equation are programmed based on the OOP approach. In this module, the data types such as variables and arrays applied to the data structure are defined to inherit characteristics among the different subroutines. With OOP, the developer can simply create a new creep damage constitutive equation subroutine that inherits many of its

features from existing subroutines and it can make this FE module much easier to modify.

2.6 Current FE software for Creep Damage Analysis

Computational creep damage mechanics have been developed and used to understand the creep deformation, creep damage evolution and creep rupture. The computational capability can only be obtained by the development and the application of special user routines either in conjunction with standard commercial software (such as ABAQUS or ANSYS) or with dedicated in-house FE software for creep damage analysis, each of which has its own advantages and disadvantages.

In FE modelling creep damage behaviours, one of the tasks behind the simulation of creep rupture process is to permit the removal of failed elements from the boundary-value problem as soon as the strength vanishes at the end of strain softening process. The failed element removal technique should be considered in the accurate simulation of creep damage behaviour because the creep rupture process includes the contact and impact of fragments that causes the dissipation of kinetic energy (Vignjevic et al., 2004). By using this technique it can be avoid the excessive distortion of the elements which may causes termination during the solution (Hayhurst et al., 1995).

A review on current state of the computational FE software for creep damage analysis is presented as follows:

i. The review of industrial standard commercial FE software for creep damage analysis

ii. The review of dedicated in-house FE software for creep damage analysis

iii. The preference for the development of dedicated in-house FE software

2.6.1 The Review of Industrial Standard FE Software

The current industrial standard commercial FE software is not able to provide the creep damage analysis capability; however, it can be expanded with the development and use of special user routines to achieve such computational capability.

The applications and characteristics of the most popular standard commercial FE software are summarized in Table 2.3.

Table 2.3: The industrial standard FE software

Standard FE software	Samples of application	Observation and Comment
ABAQUS	Benchmarks for FE analysis of creep CDM (Becker et al., 1994) Numerical investigation on the creep damage induced by void growth in HAZ of weldments (Yu et al., 2009)	User must develop a user routine to incorporate into ABAQUS such as ABAQUS-UMAT for the analysis of creep damage behaviour (Becker et al., 1994). It can access to a wide range of element types, material models and other facilities such as efficient equation solvers, which are not normally available in in-house FE codes. It does not currently permit the removal of failed elements from the boundary-value problem during the solution process (Mustata et al., 2006). CDM has not been incorporated in this FE software; it can analyse the primary-secondary creep stage but is unable to simulate the tertiary creep stage where significant damage occurs (Moberg, 1995).
ANSYS	On the accuracy of creep damage predictions in thin walled structures using the FEM (Altenbach et al., 2000) Simulation of early age concrete creep stress based on ANSYS (Li and Wu, 2008)	User must develop a user routine to incorporate into ANSYS for the analysis of creep damage behaviour. The integration scheme implemented in the user routine should be stable to ensure the overall numerical stability. It does not currently permit the removal of failed elements from the boundary-value problem during the solution process. CDM has not been incorporated into this FE software as a result of the lack of the consideration of stress redistribution due to tertiary creep. (Yao et al, 2007).

MSC.Marc software	A size-dependent crystal plasticity FE model for creep and load shedding in polycrystalline titanium alloys (Venkatramani et al., 2007) Case studies of reliability analysis by stochastic methodology in BGA creep analysis (Sasaki, et al., 2005)	Marc can simulate the response of the components under static, dynamic and multi-physics loading conditions. User must develop a user routine to incorporate into Marc for the analysis of creep damage behaviour (Venkatramani et al., 2007) . Stress redistribution due to tertiary creep and the multi-axial stress rupture criterion have not been considered because of the lack of CDM. It does not currently permit the removal of failed elements.
RFPA2D-Creep	Numerical Simulation on Floor Heave Mechanism of Roadway (Junhai, 2010) Numerical Test Study on the Mechanical Behaviour of Rock Creep Fracture (Yuan et al., 2012)	The development direction of RFPA2D-Creep is the analysis of creep behaviour in the structural analysis of soils and rocks. However, the creep behaviour of metallography such as creep damage in welds is not considered (Yuan et al., 2012). It does not currently permit the removal of failed elements from the boundary-value problem during the solution process. CDM has not been incorporated in this FE software.

2.6.2 The Review of in-house FE software

The characteristics of the main in-house FE software have been summarized in Table 2.4.

Table 2.4: The main in-house FE software

FE software	Characterization	Observation and Comment
FE-DAMAGE	FE-DAMAGE was written in FORTRAN and it was developed by Hyde's research group at University of Nottingham and (Becker et al., 1994). The CDM is incorporated in this in-house software.	The source codes of this FE software have not published. The OOP approach is not mentioned in this FE software and it could be used in future.
DAMAGE XX	DAMAGE XX (2D) was developed by Hayhurst's research group at UMIST. The CDM is incorporated in this in-house software. It incorporates the physics of the creep deformation and rupture of individual phases of the weld materials (Hayhurst et al., 1984).	The source codes of this FE software have not published. This solver requires a huge computer resource (Hayhurst et al., 2005). According to Ling et al. (2000), the fourth order Runge-Kutta integration scheme used in this solver might be incorrect. The OOP approach is not mentioned in this FE software.
DNA	DNA (2D) was developed by Voyiadjis's research group at Louisiana State University. The CDM is incorporated in this in-house software. It includes both the elastic and plastic analysis of materials incorporating damage effects (Kattan and Voyiadjis, 2002).	The function of this software is limited to plastic deformation and damage in ductile materials. Voyiadjis has reported that this solver can be extended for the analysis of the creep problem; however, such a function has not yet been incorporated in this FE program. It is a 32-bit DOS executable file which can only run under the Windows 95/98/NT operating system. The number of nodes in a problem must not exceed 3000, the number of elements in a problem must not exceed 400 (Kattan and Voyiadjis, 2002).

DAMAGE XXX	DAMAGE XXX (3D) was developed based on DAMAGE XX (2D) by Hayhurst's research group at UMIST. The CDM is incorporated in this in-house software. It is running on parallel computer parallel architectures (Hayhurst et al., 2009).	The source codes of this FE software have not published. The technique for different loading and operating conditions as well as the new RAM-based numerical technique for solving a large set of simultaneous equations should be developed to cope with more complex geometries such as butt-welded pipes in power generation (Wong, 1999). According to Ling et al (2000), the fourth order Runge-Kutta integration scheme used in this solver might be incorrect. The OOP approach is not mentioned in this FE software.

2.6.3 Why choose in-house FE Software

The current industrial standard commercial FE software in conjunction with the development and the application of special user subroutines can produce the computational capability for creep damage analysis. The advantages of the analysis of creep damage problem in this way can be summarized as: 1) it can access a wide range of element types, material models and other facilities such as efficient equation solvers, which are not normally available to in-house FE codes; 2) the development work needed through the use of industrial standard FE software is less than that for the development of in-house FE software for creep damage analysis (Gorash et al., 2008). However, the author still prefers the development of dedicated in-house FE software for creep damage analysis for the following reasons:

1) Computational capability such as CDM is not readily available in commercial general-purpose standard FE software, but it can be incorporated in in-house FE codes.

2) The industrial standard commercial FE software does not currently permit the removal of the failed elements from the boundary-value problems during the solution process, but this function can be achieved through the development of in-house FE software.

3) The industrial standard FE software makes no allowance for the stress redistribution due to tertiary creep and the multi-axial stress rupture criterion in the region of the welds; however, they can be considered through the use of in-house FE software.

Thus, this research project is conducted through the development of dedicated in-house FE software for creep damage analysis and the author still believes that there are advantages and merits in developing and using in-house FE software for creep damage analysis.

In order to advance knowledge from the investigation of existing standard commercial and in-house FE software in creep damage analysis, some innovative ideas and solutions relative to the computer-based FE modelling creep damage behaviour have proposed. The following paragraphs provide a brief summary for the innovative ideas and solutions in this project.

1) To apply the OOP approach in design and development of in-house FE software for creep damage analysis.

2) To provide a novel in-house software which includes different creep damage constitutive equation types and different numerical time integration methods for user.

2.7 Numerical Integration Scheme for Creep Damage Problem

In FEM for creep damage problems, the resulting equations are highly non-linear and stiff in nature (Zienkiewicz and Cormeau, 1974). The nature of creep damage analysis is time dependant and the field variables such as stress, strain, and creep damage variables need to be updated where an integration scheme needs to be implemented. The stability and accuracy of the FE solution critically depends on the selection of the time step size associated with an appropriate integration method (Tu et al., 2004). Thus, the numerical integration method should be investigated in the development of in-house FE software for the analysis of creep damage behaviour.

The numerical time integration methods that are reviewed in this section can be classified as: 1) Euler scheme; 2) Runge-Kutta scheme. The Runge-Kutta integration scheme can be subdivided into the classical 4^{th} order Runge-Kutta integration method,

the Runge-Kutta-Merson integration method and the Runge-Kutta-Fehlberg integration method.

Finally, it is noted that the Runge-Kutta scheme has obvious advantages in the analysis of creep damage problem in comparison with the Euler integration method.

2.7.1 The Review of Existing FE Integration Method

The characteristics of existing FE integration methods for creep problem have been summarized in Table 2.5.

Table 2.5: The review of existing FE integration method for creep problem

Integration method	**Characterization**	**Observation and Comment**
Euler method	The Euler method is a first order numerical procedure for solving ordinary differential equations with a given initial value. The Euler method (James et al., 1985) can be regarded as the 1st order Runge-Kutta method and can be divided into forward Euler's method and backward Euler's method.	The Euler method required extremely small time steps to ensure the convergence of iterations and accuracy of calculations in creep fracture problem (Cormeau, 1975). Due to the high concentration of creep strain that exists near the crack tip, the use of the Euler method for creep damage simulation is relatively uneconomic (Ling et al., 2000).
Classical 4th order Runge-Kutta	The 4th order Runge-Kutta method (Zolochevsky et al., 2009) means four evaluations of functions per time step are required. For the stability consideration, it is usual to have some means of controlling the time step in order to obtain the efficiency.	In terms of creep mechanics, the rate of stress redistribution greatly differs throughout the engineering structures and usually require a small time step to achieve accurancy. Accoring to Zolochevsky et al. (2009), the 4th order Runge-Kutta method in creep analysis has the advantage that it minimises the requirement of extra storage. Furthermore, the amount of round-off error can be reduced through use of this method.

Runge-Kutta-Merson method	The Runge-Kutta-Merson method (Christiansen, 1970) is a five-stage Runge–Kutta method with fourth-order accuracy. This method only requires a single start value (Hall, 1990); and gives an automatic and rapid way for determining the step length to be used in order to obtain a predetermined accuracy.	This method only needs to compute five estimates in the next step; an estimate of the local error is then available from a weighted sum of the individual estimates (Ling et al., 2000). In creep damage analysis, this method provides an easily calculable local truncation error estimate, which can form the basis for time step selection with a time step control technique. Moreover, the Runge-Kutta-Merson method provides reasonable solution accuracy and stability with a low computational overhead (Hayhurst et al., 1984).
Runge-Kutta-Fehlberg method	The Runge-Kutta-Fehlberg method (Bose, 2009) uses six evaluations of functions per time step. This method couples the 4^{th} and 5^{th} order Runge-Kutta-methods; and the advantage is that only six evaluations are required at each step in order to achieve approximations from both the 4^{th} and 5^{th} order Runge-Kutta-methods.	In the Runge-Kutta-Fehlberg method, one extra calculation can estimate and control the error in the solution process with a higher order embedded method; thus, it can determine the choice of an adaptive step size automatically (Bose, 2009). Ling et al. (2000) commented that using the Runge–Kutta–Fehlberg method needs a larger amount of computation compared to the Runge-Kutta-Merson method because of the large number of iterations required in the analysis of creep damage through the use of FEM.

2.7.2 Why the Runge-Kutta Scheme

The well-known Euler scheme is only conditionally stable and the stability condition is rather stringent. Although improved versions have been developed by Zienkiewicz and Cormeau (1974), Krishnaswamy et al. (1995) and Cao et al. (2008), it still requires

extremely small time steps to ensure the convergence of iterations and accuracy of calculations. In creep damage analysis, the Runge-Kutta scheme has obvious advantages:

1) With the Runge-Kutta scheme, large time steps can be employed with only slightly more computational effort than for the Euler scheme. The saving in total computation time can be considerable (Ling et al., 2000).

2) An improvement in stability and accuracy of results can be obtained through the use of the Runge-Kutta scheme as demonstrated by Hayhurst and Henderson (1977).

3) The high efficiency is even more pronounced for large-scale problems where many elements and nodes are involved (Ling et al., 2000).

4) The Runge-Kutta scheme is particularly suitable for the creep damage analysis, where higher concentrations of creep strain exist near the crack tip (Hayhurst et al., 1984).

It is noted that the Euler integration, the classical 4^{th} order Runge-Kutta integration, the Runge-Kutta-Merson integration and the Runge-Kutta-Fehlberg integration methods have been programmed by the author's colleague Feng Tan and they have been incorporated with the creep damage constitutive equation in the subroutine library for the in-house FE software HITSI. More details about the numerical integration method's subroutine will be reported in Chapter 3.

2.8 Summary

This chapter gives a brief overview and discussion on the problem domains relating to this project. The mechanisms of creep deformation and creep fracture in metals and alloys are reviewed to understand the nature of the creep damage problem. The creep damage behaviour in weldment component has been identified.

It also illustrates why this project needs to be done and why new techniques need to be involved. The current state of how to achieve the computational capability for creep damage analysis and why the in-house FE software should be developed have been demonstrated. It further reports on the techniques such as CDM, FE algorithm, OOP approach and numerical integration schemes that need to be involved in this project.

The author acknowledges that some specific knowledge in this chapter has been published in (Liu et al., 2012a) during the early stage of this research.

Chapter 3 Finite Element Method

3.1 Introduction

This chapter reports the general methodology, the FE algorithm, the specific FE theory and the existing standard FE subroutines for the development of HITSI for creep damage analysis. The fundamental FE procedures used in analysing the structural problem though the FEM can be summarized as the mesh discretization of the structure, element stiffness assembly, the solution of the equilibrium equation and recovery of results at the integrating points.

The specific requirements for this chapter include:

1) To consider the general methodology in the development of the in-house FE software HITSI for creep damage analysis.

2) To report the FE algorithm used in developing HITSI for creep damage analysis. The general FE algorithm, the creep damage constitutive equation, the numerical integration method and the explicit stress update FE algorithm that are used in developing HITSI are demonstrated.

3) To report on the finite elements (the mesh discretization of a continuous domain) in the development HITSI for creep damage analysis. Theoretical knowledge of the derivation of constitutive equations for the 2D (plane stress, plane strain and axisymmetric) and 3D element type are investigated, and the existing FE standard subroutines to set up element data are reported.

4) To report the element stiffness assembly method in developing HITSI for creep damage analysis. Theoretical knowledge of element assembly is presented firstly; then, the existing standard FE subroutines for the element stiffness assembly are reported.

5) To report the solution of the equilibrium equation and recovery of results at the integrating points methods in the development of HITSI for creep damage analysis. Theoretical knowledge of the solution of the equilibrium equation and integrating point result recovery is presented first followed by the relevant existing standard subroutines FE subroutines.

3.2 The General Methodology Consideration for the Development of HITSI

The general methodology used in developing in-house FE software for creep damage analysis can be divided into the following four parts: 1) Planning; 2) Programming; 3) Validation; 4) Software maintenance.

- Planning: The objective of this project is the development of FE software for creep damage analysis. A mathematical model of the creep damage behaviour should be formulated including the material independent equations, constitutive (evolution) equations as well as initial and boundary conditions (Ralph and Wand, 2009) . The use of the CDM approach and FEM in conjunction with an advanced engineering computer programming language (Fortran 2003) based on the OOP approach is planned in this project to develop HITSI. The existing standard FE subroutines adopted from Smith and Griffiths (2005) and a specific subroutine library provided by the author's colleague Feng Tan should be utilized in developing HITSI for efficiency.

- Programming: The development strategy and the general flow diagram for the development of HITSI have been developed for the description of the FE procedures in creep damage analysis. The development of HITSI as conducted includes the linear elastic stage, the non-linear (single material and time independent) elastic-plastic stage and the time dependent creep deformation and creep damage stage. Moreover, the 2D (plane stress, plane strain and axisymmetric) version and 3D version FE programs should be developed within the characteristics of updating stress and damage field variables, multi-materials zones and failed elements removal.

- Validation: The validation of the FE codes for HITSI has been developed and it can be divided into two parts: 1) the FE simulated results from HITSI (uni-axial case) are compared with the theoretical results to demonstrate the validity of the FE program; 2) a benchmark test of HITSI via the numerical investigation of creep damage behaviour of a Cr-Mo-V steam pipe weldment case (multi-axial case) and the computational results from HITSI are compared with existing results to demonstrate the validity of HITSI.

- Software maintenance: Maintaining and enhancing software is necessary to cope with newly discovered faults or requirements (Keates et al., 2000). User guidance of the in-house software HITSI has been developed and the instructions have been prepared with pseudo code to improve the readability and sustainability for later development by new software engineers.

3.3 The FE Algorithm for the Development of HITSI

The numerical FE algorithm used in the development of in-house FE software for creep damage can be divided into the following four parts: 1) the general FE algorithm; 2) the creep damage constitutive equation; 3) the numerical integration method; 4) the stress update FE algorithm.

3.3.1 The General FE Algorithm

The explicit FE algorithm is used in the development of HITSI and the advantages of the use of this algorithm in creep damage analysis have been reviewed in Chapter 2.

The computational solution with FEM for creep damage starts by solving the boundary value problem; it uses the initial elastic stresses to substitute into the creep damage constitutive equation, and the creep damage and strain fields are integrated with respect to time (Ling et al., 2000). Here, assuming the total strain ε in FE program can be partitioned into the elastic strain and creep strain, the total strain increment can be expressed as:

$$\Delta\varepsilon = \Delta\varepsilon^{e} + \Delta\varepsilon^{c} \tag{3.1}$$

Where the $\Delta\varepsilon$, $\Delta\varepsilon^{e}$and $\Delta\varepsilon^{c}$ are increments in total, elastic and creep strain components, respectively (Ling et al., 2000).

The stress increment is related to the elastic and creep strain increments by:

$$\Delta\sigma = D(\Delta\varepsilon - \Delta\varepsilon^{c}) \tag{3.2}$$

Where D is the stress-strain matrix and contains the elastic constants.

The stress increments are related to the incremental displacement vector Δu by:

$$\Delta\sigma = D(B\Delta u - \Delta\varepsilon^{c}) \tag{3.3}$$

Where B represents the strain-displacement matrix, and the equilibrium equation to be satisfied any time can be expressed by:

$$\int vB^T \Delta\sigma \, dv = \Delta R \tag{3.4}$$

Where ΔR is the vector of the equivalent nodal mechanical load and v is the element volume. Combining Equation 3.3 and Equation 3.4:

$$\int vB^T D(B\Delta u - \Delta\varepsilon^c) \, dv = \Delta R \tag{3.5}$$

The ΔR is used to update the loads applied to the structure of the FE model.

3.3.2 The Creep Damage Constitutive Equation

The computational capability relies on the availability of a computational tool and a set of creep damage constitutive equations that can depict the complex creep phenomena. The use of the creep damage constitutive equation is proposed to depict the behaviour of material during creep damage (deformation and rupture) process, and especially for predicting the lifetime of the material. The creep damage constitutive equation is a key part programming and the accuracy of the lifetime prediction depends on such an equation.

The most popular creep damage constitutive equations have been programmed and included in Feng Tan's subroutine library. This subroutine library has been utilized in the development of HITSI and contains constitutive equation subroutines for the Kachanov-Rabotnov-Hayhurst, the Kachanov-Rabotnov and the Kachanov-Rabotnov-Hayhurst-Xu models. The creep damage constitutive equations are presented as follows:

a) Kachanov-Rabotnov-Hayhurst equation:

The Kachanov-Rabotnov-Hayhurst equation (Perrin and Hayhurst, 1996a) is well-known and is widely used in creep damage analysis and includes both uni-axial and multi-axial forms (Perrin and Hayhurst, 1996a).

1. The uni-axial form:

$$\dot{\varepsilon} = A \sinh\left(\frac{B\sigma(1-H)}{(1-\varphi)(1-\omega)}\right) \tag{3.6}$$

$$\dot{H} = \frac{h}{\sigma}\left(1 - \frac{H}{H^*}\right)\dot{\varepsilon} \tag{3.7}$$

$$\dot{\varphi} = \frac{K_C}{3}(1-\varphi)^4 \tag{3.8}$$

$$\dot{\omega} = C\dot{\varepsilon}^{*} \tag{3.9}$$

Where *A, B C, h, H** and K_C are material parameters. H (0 < H < H*) indicates strain hardening during primary creep, φ (0< φ < 1) describes the evolution of spacing of the carbide precipitates (Perrin and Hayhurst, 1996a).

2. The multi-axial form:

$$\dot{\varepsilon_{ij}} = \frac{3S_{ij}}{2} Asinh(\frac{B\sigma_e(1-H)}{(1-\varphi)(1-\omega)}) \tag{3.10}$$

$$\dot{H} = \frac{h}{\sigma_e}\left(1 - \frac{H}{H^*}\right)\dot{\varepsilon_e} \tag{3.11}$$

$$\dot{\varphi} = \frac{K_C}{3}(1-\varphi)^4 \tag{3.12}$$

$$\dot{\omega} = C\dot{\varepsilon_e}\langle\frac{\sigma_1}{\sigma_e}\rangle^{\nu} \tag{3.13}$$

Where S_{ij} is the deviator stress tensor, σ_e is the Von Mises stress, σ_1 is the maximum principal stress and ν is the stress state index defining the multi-axial stress rupture criterion (Perrin and Hayhurst, 1996a)

b) Kachanov-Rabotnov equation:

The Kachanov-Rabotnov equation (Rabotnov, 1969) is used in the validation of the in-house FE codes and the benchmark test of HITSI via the numerical investigation of creep damage behaviour of a Cr-Mo-V steam pipe weldment case. The Kachanov-Rabotnov equation also contains uni-axial and multi-axial forms.

1. The uni-axial form:

$$\dot{\varepsilon} = K'\left(\frac{\sigma}{1-\omega}\right)^{n} \tag{3.14}$$

$$\dot{\omega} = M'\frac{\sigma^{\nu}}{(1-\omega)^{\Phi}} \tag{3.15}$$

Where *K'*, *M '*, *n*, *ν* and *Φ* are material constants (Hall, 1990).

2. The multi-axial form:

$$\dot{\varepsilon_{ij}} = \frac{3}{2}\frac{K'{\sigma_e}^{n-1}}{(1-\omega)^n}\, S_{ij}\, f(t) \tag{3.16}$$

$$\dot{\omega} = M'\frac{\Delta\, x(\sigma_{ij})}{(1-\omega)^{\Phi}}\; f(t) \tag{3.17}$$

Where $f(t)$ represents the primary creep region and is taken as $f(t) = t^m$ and $x = \nu(m+1)$ to represent the time scale modification (Hall, 1990).

c) Kachanov-Rabotnov-Hayhurst-Xu equation:

The Kachanov-Rabotnov-Hayhurst-Xu equation (Xu, 2001) is based on the Kachanov-Rabotnov-Hayhurst equation. Its uni-axial form is the same as the uni-axial form of the Kachanov-Rabotnov-Hayhurst equation; however, the improvement of the Kachanov-Rabotnov-Hayhurst-Xu equation is that the effect of states of stress is considered in its multi-axial form.

1. The multi-axial form:

$$\dot{\varepsilon_{ij}} = \frac{3S_{ij}}{2} A sinh(\frac{B\sigma_e(1-H)}{(1-\varphi)(1-\omega)}) \tag{3.18}$$

$$\dot{H} = \frac{h\dot{\varepsilon}_e}{\sigma_e}\left(1-\left(\frac{H}{H^*}\right)\right) \tag{3.19}$$

$$\dot{\varphi} = \frac{K_C}{3}(1-\varphi)^4 \tag{3.20}$$

$$\dot{\omega} = CN\dot{\varepsilon}_e f_2 \tag{3.21}$$

$$\dot{\omega}_d = CN\dot{\varepsilon}_e f_1 \tag{3.22}$$

$$f_1 = \left(\frac{2\sigma_e}{3S_1}\right)^a exp\left\{b\left[\frac{3\sigma_m}{S_S}-1\right]\right\} \tag{3.23}$$

$$f_2 = exp\left[p\left(1-\frac{\sigma_1}{\sigma_{vm}}\right)+q\left(\frac{1}{2}-\frac{3\sigma_m}{2\sigma_{vm}}\right)\right] \tag{3.24}$$

Where f_1 and f_2 are functions of stress states. The function f_2 is introduced to depict the effect of states of stress on the damage evolution. The additional function f_1 is introduced to better represent phenomenologically the coupling between damage and tertiary deformation and creep rupture (Xu, 2001).

3.3.3 The Numerical Integration Method

The FE solution depends critically on the size selection of the time steps associated with an appropriate integration method. The most popular integration methods have been programmed and included in Feng Tan's subroutine library utilized in the development of HITSI. This library contains integration subroutines for the Euler, the classical 4th order Runge-Kutta, the Runge-Kutta-Merson and the Runge-Kutta-Fehlberg methods. The algorithms of the integration methods are presented as follows:

a) Euler integration method (Cormeau, 1975):

$$y_{i+1} = y_i + f(x_i, y_i)\,\Delta t \tag{3.25}$$

b) Classical 4th order Runge-Kutta integration method (Zolochevsky et al., 2009):

$$k_1 = f(x_i, y_i) \tag{3.26}$$

$$k_2 = f\left(x_i + \frac{1}{2}\Delta t, y_i + \frac{1}{2}k_1\right) \tag{3.27}$$

$$k_3 = f\left(x_i + \frac{1}{2}\Delta t, y_i + \frac{1}{2}k_2\right) \tag{3.28}$$

$$k_4 = f(x_i + \Delta t, y_i + k_3) \tag{3.29}$$

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\Delta t \tag{3.30}$$

c) Runge-Kutta-Merson integration method (Christiansen, 1970):

$$k_1 = f(x_i, y_i) \tag{3.31}$$

$$k_2 = f\left(x_i + \frac{1}{3}\Delta t, y_i + \frac{1}{3}k_1\right) \tag{3.32}$$

$$k_3 = f\left(x_i + \frac{1}{3}\Delta t, y_i + \frac{1}{6}(k_1 + k_2)\right) \tag{3.33}$$

$$k_4 = f\left(x_i + \frac{1}{2}\Delta t, y_i + \frac{1}{8}(k_1 + 3k_3)\right) \tag{3.34}$$

$$k_5 = f\left(x_i + \Delta t, y_i + \frac{1}{2}(k_1 - 3k_3 + 4k_4)\right) \tag{3.35}$$

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 4k_4 + k_5)\Delta t \tag{3.36}$$

d) Runge-Kutta-Fehlberg integration method (Bose, 2009):

$$k_1 = f(x_i, y_i) \tag{3.37}$$

$$k_2 = f\left(x_i + \frac{1}{4}\Delta t, y_i + \frac{1}{4}k_1\right) \tag{3.38}$$

$$k_3 = f\left(x_i + \frac{3}{8}\Delta t, y_i + \frac{3}{32}k_1 + \frac{9}{32}k_2\right) \tag{3.39}$$

$$k_4 = f\left(x_i + \frac{12}{13}\Delta t, y_i + \frac{1932}{2197}k_1 - \frac{7200}{2179}k_2 + \frac{7296}{2179}k_3\right) \tag{3.40}$$

$$k_5 = f\left(x_i + \Delta t, y_i + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right) \tag{3.41}$$

$$k_6 = f\left(x_i + \frac{1}{2}\Delta t, y_i - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right) \tag{3.42}$$

$$y_{i+1} = y_i + \left(\frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 - \frac{2}{55}k_6\right)\Delta t \tag{3.43}$$

3.3.4 The Stress Update FE Algorithm

Creep deformation can be regarded as a time-related plastic deformation and the process of the creep damage is extremely non-linear and transient. The creep damage field variables such as creep strain, damage and stress should be updated with the time integration. The explicit FE algorithm, which has been demonstrated to have obvious advantages in dealing with highly non-linear creep damage problems in Chapter 2, is adapted for the development of a stress update FE algorithm in HITSI.

It is noted that Hayhurst et al. (2005) and Becker et al. (1994) had developed in-house FE software for creep problems. Hall et al. (1991) have reported their stress update FE algorithm in the FE software DAMAGE XX; later on, Smith and Griffiths (2005) presented the stress update FE algorithm for elastic-plastic problem. Here, the explicit stress update FE algorithm for HITSI is developed based on the studies of Hall and Hayhurst (1991) and Smith and Griffiths (2005). The loop of the stress update FE algorithm for HITSI is shown in Figure 3.1.

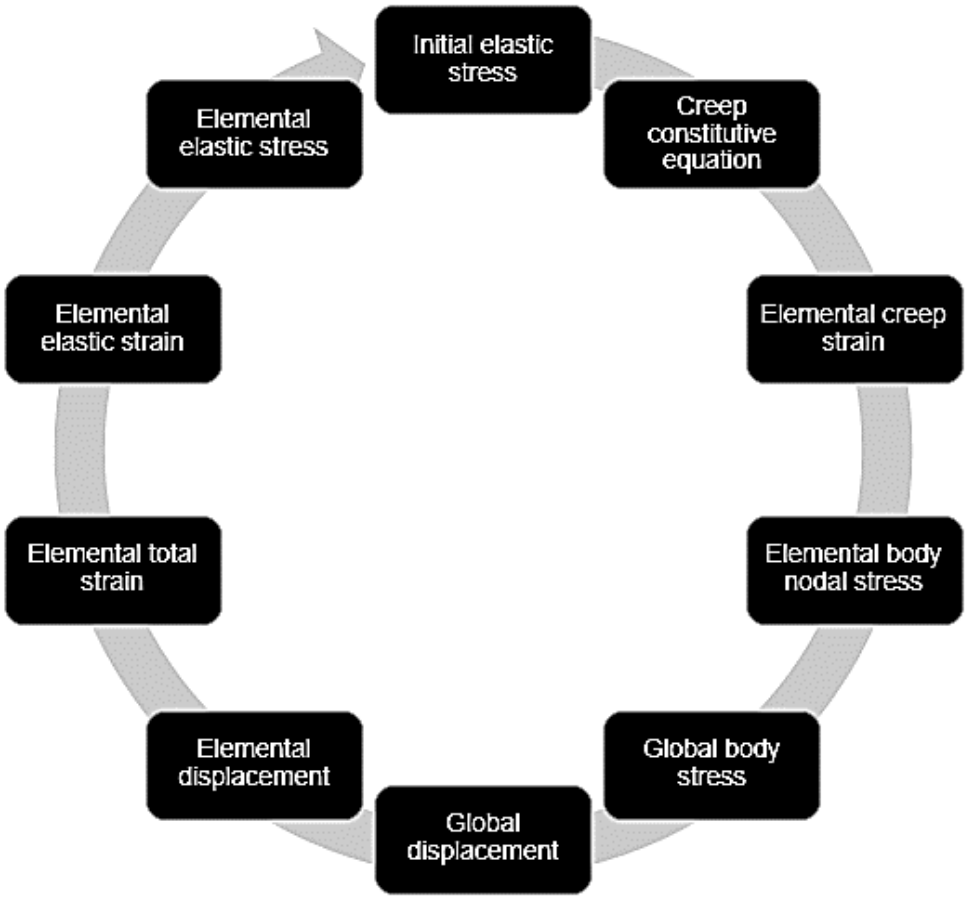


Figure 3.1: The loop of the stress update FE algorithm for HITSI

In this FE algorithm, the loads vector consists of external applied loads and self-equilibrating "body loads" at each time iteration in the program. Thus, the body loads have the effect of redistributing stress within the system and the stress in the system should therefore be updated.

The process of the stress update FE algorithm can be summarized as follows:

1) Substituting the initial elastic stress into the creep damage constitutive equation and calculating the elemental creep strain.

2) Substituting the elemental creep strain into the stress-strain matrix and calculating the elemental body nodal stress.

3) Assembling the elemental body nodal stress into the global body stress and solving the equilibrium equation.

4) Calculating the global displacement and extracting the elemental displacement from the global displacement vector.

5) Substituting the elemental displacement into the strain-displacement matrix and calculating the elemental elastic strain.

6) Substituting the elemental elastic strain into stress-strain matrix and calculating the elemental elastic stress.

7) Substituting the elastic stress into the creep damage constitutive equation to complete this loop; and to repeat the above steps for the next loop.

The FE formulations of the stress update FE algorithm for HITSI are presented as follows:

The principle of virtual work applied to the boundary value problem is given by:

$$P_{load} = [K_v] \times TOTD - P_c \tag{3.44}$$

Where P_{load} is the applied force vector; $[K_v]$ is the global stiffness matrix, which is assembled by the element stiffness matrices $[K_m]$; *TOTD* is the global vector of the nodal displacement and P_c is the global creep force vector.

$$[K_v] = \int\int [B]^T [D][B] dxdy \tag{3.45}$$

Here, $[B]$ and $[D]$ represent the strain-displacement and stress-strain matrices, respectively.

$$TOTD = [K_v]^T \times (P_{load} + P_c) \tag{3.46}$$

The initial P_c is zero and the Cholesky Method (Liu, 2005) is used for the inverse of the global stiffness matrix $[K_v]$. Given P_{load}, the elastic strain ε_{ek} and the elastic stress σ_{ek} for each element can be obtained by:

$$\varepsilon_{ek} = [B] \times ELD \tag{3.47}$$

$$\sigma_{ek} = [D] \times \varepsilon_{ek} \tag{3.48}$$

The element node displacement *ELD* can be found from the global displacement vector and the creep strain rate ε_{ckrate} for each element which can be obtained by substituting the element elastic stress into the creep damage constitutive equation. The creep strain can be calculated as:

$$\varepsilon_{ck(t+\Delta t)} = \varepsilon_{ck(t)} + \varepsilon_{ckrate} \times \Delta t \quad (3.49)$$

The nodal creep force vectors for each element are given by:

$$P_{ck} = [B]^T [D] \times \varepsilon_{ck} \quad (3.50)$$

The nodal creep force vector P_{ck} can be assembled into the global creep force vector P_c and the P_c is used to update Equation 3.44. Thus, the elastic strain can be updated:

$$\varepsilon_{totk} = [B] \times ELD \quad (3.51)$$

$$\varepsilon_{totk} = \varepsilon_{ek} + \varepsilon_{ck} \quad (3.52)$$

$$\varepsilon_{ek} = [B] \times ELD - \varepsilon_{ck} \quad (3.53)$$

Where the ε_{totk} and ε_{ck} represent the total strain and creep strain for each element, respectively; and the elastic strain ε_{ek} is used to update Equation 3.48.

3.4 Finite Elements

The FEM is a numerical method for solving partial differential equations by discretising these equations in their space dimensions (Smith et al., 2013). The finite elements (sub-domains) are dispersed by the mesh discretization of a continuous domain and such finite elements are connected with adjacent elements at their nodes. The generation of finite elements data is a process of generating the geometric data of the element and involves computing the coordinates of nodes, defining nodes connectivity and thus constructing the elements. The geometric features of generated elements influence the overall performance and accuracy of the FE analysis. Thus, the generation of finite elements data is one of the most important procedures in the development of FE software.

The characteristics of finite elements have already been described in many well-known FEM books such as Zienkiewicz and Taylor (2000), Zienkiewicz and Taylor (2005), Smith and Griffiths (2005), Oden and Reddy (2012) and Smith et al. (2013). Moreover, existing, and published, standard FE subroutines such as (Smith and Griffiths, 2005) have been utilized to set up element data in the development of HITSI.

This project covers the development of the in-house FE software HITSI for the analysis of 2D (plane stress, plane strain and axisymmetric) and 3D creep damage problems.

Thus, the constitutive equations for the 2D (plane stress, plane strain and axisymmetric) and 3D element type have been investigated and the existing FE standard subroutines to set up element data are reported.

3.4.1 The Characteristics of Finite Elements

The characteristics of finite elements are investigated to understand the relationship between the constitutive matrix and the problem types (plane stress, plane strain, axisymmetric, and three-dimensional). Furthermore, the definition of the analysis type in the development and the application of HITSI can also be achieved through the investigation of the characteristics of finite elements.

The generation of finite elements usually starts with the division of the body under consideration into small regions and such small regions are then subdivided into finite elements. The subdivision between regions should be located where there is a change in geometry or material properties (Cook, 2007). The generated elements that should be considered are those that can be joined together at nodes so that complete compatibility and equilibrium achieved (Smith et al., 2013).

In FEM, the derivation of formulas for the constitutive matrix to define the analysis of problem type may be summarized and presented as follows:

In a state of plane stress, the stresses in the structure must satisfy the following equilibrium equations (Zienkiewicz and Taylor, 2000):

$$\begin{cases} \dfrac{\partial \sigma_x}{\partial x} + \dfrac{\partial \tau_{xy}}{\partial y} + f_x = 0 \\ \dfrac{\partial \tau_{xy}}{\partial x} + \dfrac{\partial \sigma_y}{\partial y} + f_y = 0 \end{cases} \tag{3.54}$$

Where σ_x, σ_y and τ_{xy} are non-zero stress components; f_x and f_y are body forces, such as gravity forces, per unit volume (Zienkiewicz and Taylor, 2000).

For plane structures, the relationships between strain and displacement under the small strain and small rotation hypotheses can be written as (Smith and Griffiths, 2005):

$$\varepsilon_x = \frac{\partial u}{\partial x}, \quad \varepsilon_y = \frac{\partial v}{\partial y}, \quad \gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \tag{3.55}$$

In the FE method, the strain-displacement relationship can be re-written in matrix form (Smith and Griffiths, 2005):

$$\begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \begin{Bmatrix} u \\ v \end{Bmatrix} \tag{3.56}$$

Where u is the displacement in the x and v is the displacement in the y directions. According to the Hook's Law, the stress and strain relationship (Zienkiewicz and Taylor, 2000) can be presented by:

$$\boldsymbol{\sigma} = \mathbf{D}\boldsymbol{\varepsilon} \tag{3.57}$$

Where the stress and strain vectors are $\boldsymbol{\sigma} = \begin{Bmatrix} \sigma_{xx} & \sigma_{yy} & \tau_{xy} \end{Bmatrix}^T$ and $\boldsymbol{\varepsilon} = \begin{Bmatrix} \varepsilon_{xx} & \varepsilon_{yy} & \gamma_{xy} \end{Bmatrix}^T$, respectively. **D** is represented by:

$$\mathbf{D} = \begin{bmatrix} D_{11} & D_{12} & D_{13} \\ D_{21} & D_{22} & D_{23} \\ D_{31} & D_{32} & D_{33} \end{bmatrix} = \begin{bmatrix} \{D1\}^T \\ \{D2\}^T \\ \{D3\}^T \end{bmatrix} \tag{3.58}$$

The vectors $\{D1\}$, $\{D2\}$ and $\{D3\}$ are the first, second and third row of the matrix **D** respectively (Zienkiewicz and Taylor, 2005).
Thus:

$$\{\boldsymbol{\sigma}\} = \begin{Bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \tau_{xy} \end{Bmatrix} = [\mathbf{D}]\{\boldsymbol{\varepsilon}\} = \begin{Bmatrix} \{D1\}^T \{\boldsymbol{\varepsilon}\} \\ \{D2\}^T \{\boldsymbol{\varepsilon}\} \\ \{D3\}^T \{\boldsymbol{\varepsilon}\} \end{Bmatrix} \tag{3.59}$$

The derivation of element stiffness can be described by the energy approach. The expression for the total potential energy W_p in a linearly elastic body is shown by (Zienkiewicz and Taylor, 2005):

$$W_p = \int_V \left(\frac{1}{2}\boldsymbol{\varepsilon}^T \mathbf{E}\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}^T \mathbf{E}\boldsymbol{\varepsilon}_0 + \boldsymbol{\varepsilon}^T \boldsymbol{\sigma}_0 \right) dV - \int_V \mathbf{u}^T \mathbf{F} dV - \int_S \mathbf{u}^T \Phi dS - \mathbf{D}^T \mathbf{P} \tag{3.60}$$

Where $\mathbf{u}$ represents the displacement field; $\boldsymbol{\varepsilon}$ represents the strain field; $\mathbf{E}$ the elastic constants matrix (material property); $\boldsymbol{\varepsilon}_0$ and $\boldsymbol{\sigma}_0$ are initial strains and initial stresses; $\mathbf{F}$ is the body forces; $\boldsymbol{\Phi}$ is the surface tractions; $\mathbf{D}$ represents the vector of global nodal displacements and $\mathbf{P}$ is the loads (Zienkiewicz and Taylor, 2005).

Displacements within an element are interpolated from the element nodal displacement **d** with the shape function matrix **N** and can be obtained as:

$$\mathbf{u} = \mathbf{N}\mathbf{d} \tag{3.61}$$

Strains are obtained from displacements by Equation 3.56 and Equation 3.61.

$$\boldsymbol{\varepsilon} = \partial\mathbf{u} = \partial\mathbf{N}\mathbf{d} = \mathbf{B}\mathbf{d} \tag{3.62}$$

Substitution of the Equation 3.61 and Equation 3.62 into Equation 3.60:

$$W_p = \frac{1}{2}\sum_{n=1}^{Nr.El.}\mathbf{d}_n^T\mathbf{k}_n\mathbf{d}_n - \sum_{n=1}^{Nr.El}\mathbf{d}_n^T\mathbf{r}_{en} - \mathbf{D}^T\mathbf{P} \tag{3.63}$$

The element stiffness matrix and element equivalent nodal loads vector (Smith and Griffiths, 2005) are defined as:

$$\mathbf{k} = \int_{V_e}\mathbf{B}^T\mathbf{E}\mathbf{B}\,dV_e \tag{3.64}$$

$$\mathbf{r}_e = \int_{V_e}\mathbf{B}^T\mathbf{E}\boldsymbol{\varepsilon}_0\,dV_e - \int_{V_e}\mathbf{B}^T\boldsymbol{\sigma}_0\,dV_e + \int_{V_e}\mathbf{N}^T\mathbf{F}\,dV_e + \int_{S_e}\mathbf{N}^T\boldsymbol{\Phi}\,dS_e \tag{3.65}$$

Where V_e denotes the volume of an element and S_e its surface and in the surface integral the shape function matrix is evaluated on S_e (Smith and Griffiths, 2005).

Every degree of freedom in an element displacement vector **d** also appears in the vector of global displacement matrix, thus the global stiffness matrix and nodal force vector (Smith et al., 2013) can be defined as:

$$\mathbf{K} = \sum_{n=1}^{Nr.El.}\mathbf{k}_n \tag{3.66}$$

$$\mathbf{R} = \sum_{n=1}^{Nr.El}\mathbf{r}_{en} + \mathbf{P} \tag{3.67}$$

By combining Equation 3.66, Equation 3.67 and Equation 3.63, the relationship between the global stiffness matrix and nodal loads vector (Smith et al., 2013) can be shown to be:

$$\mathbf{KD} = \mathbf{R} \tag{3.68}$$

In this research, the formulation of finite elements characteristics for the plane stress, plane strain, axisymmetric and three-dimensional element types is investigated.

a) Plane stress:

In the plane stress problem, the condition prevails in a flat plate in the x and y plane, loaded only in its own plane and without z-direction restraint, so that $\sigma_x = \tau_{xy} = \tau_{zx} = 0$. Thus, the constitutive matrix (Oden and Reddy, 2012) is:

$$\mathbf{E} = \frac{E}{1-\nu^2}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & (1-\nu)/2 \end{bmatrix} \tag{3.69}$$

Where the *E* is the Young's modulus and ν is the Poisson's ratio.

b) Plane strain:

In the plane strain problem, the condition that prevails is defined as a deformation state in which w=0 everywhere and u and v are functions of x and y but not of z. Thus, a typical slice of, say, an underground tunnel that lies along the z axis might deform in essentially plane strain conditions. The constitutive matrix (Zienkiewicz and Taylor, 2005) is:

$$\mathbf{E} = \frac{E}{(1+\nu)(1-2\nu)}\begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & (1-2\nu)/2 \end{bmatrix} \tag{3.70}$$

Where the *E* is the Young's modulus and ν is the Poisson's ratio. If needed, σ_z can be obtained from the relationship $\varepsilon_z = 0 = (\sigma_z - \nu\sigma_y - \nu\sigma_x) / E$ after σ_x and σ_y are known.

c) *Axisymmetric:*

In the axisymmetric problem, the condition considers a constant value of displacement in the circumferential direction. The stress and strain components for the element are:

$$\boldsymbol{\sigma}^T = \begin{bmatrix} \sigma_r & \sigma_\theta & \sigma_z & \tau_{rz} \end{bmatrix} \tag{3.71}$$

$$\boldsymbol{\varepsilon}^T = \begin{bmatrix} \varepsilon_r & \varepsilon_\theta & \varepsilon_z & \gamma_{rz} \end{bmatrix} \tag{3.72}$$

Where the strains are defined as follows, with *u* and *w* being the displacements in the *r* and *z* directions respectively:

$$\varepsilon_r = \frac{\partial u}{\partial r},\ \varepsilon_\theta = \frac{u}{r},\ \varepsilon_z = \frac{\partial w}{\partial z},\ \gamma_{rz} = \frac{\partial u}{\partial z} + \frac{\partial w}{\partial r} \tag{3.73}$$

The constitutive matrix (Smith et al., 2013) is:

$$\mathbf{E} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 \\ \nu & 1-\nu & \nu & 0 \\ \nu & \nu & 1-\nu & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix} \tag{3.74}$$

Where the *E* is the Young's modulus and ν is the Poisson's ratio.

d) Three-dimensional:

The three-dimensional problem usually requires a larger total number of elements to obtain reasonable simulated results; hence a larger storage capacity is necessary (Zienkiewicz and Cheung, 1967). Equations 3.54 and 3.55 may be extended to the three-dimensional displacement components and the stress and strain components (Smith et al., 2013) for the element are then:

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_x & \sigma_y & \sigma_z & \tau_{xy} & \tau_{yz} & \tau_{zx} \end{bmatrix} \tag{3.75}$$

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_x & \varepsilon_y & \varepsilon_z & \gamma_{xy} & \gamma_{yz} & \gamma_{zx} \end{bmatrix} \tag{3.76}$$

Where the strains are defined as follows, with *u, v* and *w* being the displacements in the x, y and z directions respectively.

$$\varepsilon_x = \frac{\partial u}{\partial x} + \frac{1}{2}\left[\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial w}{\partial x}\right)^2\right] \tag{3.77}$$

$$\varepsilon_y = \frac{\partial v}{\partial y} + \frac{1}{2}\left[\left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 + \left(\frac{\partial w}{\partial y}\right)^2\right] \tag{3.78}$$

$$\varepsilon_x = \frac{\partial w}{\partial z} + \frac{1}{2}\left[\left(\frac{\partial u}{\partial z}\right)^2 + \left(\frac{\partial v}{\partial z}\right)^2 + \left(\frac{\partial w}{\partial z}\right)^2\right] \tag{3.79}$$

$$\gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} + \frac{\partial u}{\partial x}\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\frac{\partial v}{\partial y} + \frac{\partial w}{\partial x}\frac{\partial w}{\partial y} \tag{3.80}$$

$$\gamma_{xz} = \frac{\partial u}{\partial z} + \frac{\partial w}{\partial x} + \frac{\partial u}{\partial x}\frac{\partial u}{\partial z} + \frac{\partial v}{\partial x}\frac{\partial v}{\partial z} + \frac{\partial w}{\partial x}\frac{\partial w}{\partial z} \tag{3.81}$$

$$\gamma_{yz} = \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} + \frac{\partial u}{\partial y}\frac{\partial u}{\partial z} + \frac{\partial v}{\partial y}\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\frac{\partial w}{\partial z} \tag{3.82}$$

The constitutive matrix (Smith et al., 2013) is:

$$\mathbf{E} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)}\begin{bmatrix} 1, & \nu/(1-\nu), & \nu/(1-\nu), & 0, & 0, & 0 \\ & 1, & \nu/(1-\nu), & 0, & 0, & 0 \\ & & 1, & 0, & 0, & 0 \\ & SYM.... & & (1-2\nu)/2(1-\nu), & 0, & 0 \\ & & & & (1-2\nu)/2(1-\nu), & 0 \\ & & & & & (1-2\nu)/2(1-\nu) \end{bmatrix} \tag{3.83}$$

3.4.2 The Existing Standard FE Subroutines to Set up Element Data

In this research, the existing standard FE subroutines (Smith and Griffiths, 2005) have been utilized to set up element data. The limitation imposed by using these subroutines is that the mesh generated by the subroutines should satisfy the order of node and freedom numbering rule: the first node can be located at any corner, but subsequent corners and freedoms must follow in a clockwise sense. The introduction and function of these subroutines are summarized in Table 3.1.

Table 3.1: The existing standard FE subroutines to set up element data (Smith and Griffiths, 2005)

Subroutine name	Parameter index	Function
num_to_g	*num, nf, g*	This subroutine returns the element steering vector *g* from the element node *num* and the nodal freedom array *nf*
Geometry_3tx	*iel, nxe, aa, bb, coord, num*	This subroutine forms the coordinates *coords* and the element node vector *num* for a mesh of uniform 3-node triangles. *iel* is the element number. *nxe* is the number of columns of elements. *aa* and *bb* are the width and depth of element. It counts in the x-direction and local numbering is clockwise.
Geometry_6tx	*iel, nxe, aa, bb, coord, num*	This subroutine forms the coordinates *coords* and the element node vector *num* for a mesh of uniform 6-node triangles. *iel* is the element number. *nxe* is the number of columns of elements. *aa* and *bb* are the width and depth of element. It counts in the x-direction and local numbering is clockwise.
Geometry_15tyv	*iel, nye, width, depth, coord, num*	This subroutine forms the coordinates *coords* and the element node vector *num* for a mesh of uniform 15-node triangles. *iel* is the element number. *nxe* is the number of rows of elements. *width* and *depth* are the width and depth of element. It counts in the y-direction and local numbering is clockwise.
Geometry_4qx	*iel, nxe, aa, bb, coord, num*	This subroutine forms the coordinates *coords* and the element node vector *num* for a mesh of uniform 4-node quadrilaterals. *iel* is the element number. *nxe* is the number of columns of elements. *aa* and *bb* are the width and depth of element. It counts in the x-direction and local numbering is clockwise.

Geometry_8qx	*iel, nxe, aa, bb, coord, num*	This subroutine forms the coordinates *coords* and the element node vector *num* for a mesh of uniform 8-node quadrilaterals. *iel* is the element number. *nxe* is the number of columns of elements. *aa* and *bb* are the width and depth of element. It counts in the x-direction and local numbering is clockwise.
Geometry_9qx	*iel, nxe, aa, bb, coord, num*	This subroutine forms the coordinates *coords* and the element node vector *num* for a mesh of uniform 9-node quadrilaterals. *iel* is the element number. *nxe* is the number of columns of elements. *aa* and *bb* are the width and depth of element. It counts in the x-direction and local numbering is clockwise.
Geometry_8bxz	*iel, nxe, nze, aa, bb, cc, coord, num*	This subroutine forms the coordinates *coords* and the element node vector *num* for a mesh of uniform 8-node brick elements. *iel* is the element number. *nxe* is the number of columns of elements in x direction. *nze* is the number of columns of elements in z direction. *aa*, *bb* and *cc* are the width, depth and length of the element.
Geometry_20bxz	*iel, nxe, nze, aa, bb, cc, coord, num*	This subroutine forms the coordinates *coords* and the element node vector *num* for a mesh of uniform 20-node brick elements. *iel* is the element number. *nxe* is the number of columns of elements in x direction. *nze* is the number of columns of elements in z direction. *aa*, *bb* and *cc* are the width, depth and length of the element.

3.5 Element Stiffness Matrix Assembly

3.5.1 Assembly Procedures for the Element Stiffness Matrix

Assembly procedures (Smith et al., 2013) that are used in developing HITSI are based on direct summation with the use of the element connectivity array. The method of assembly of the global stiffness matrix from contributions of the element stiffness matrix can be expressed by the following procedures:

1. Looping the total number of degrees of freedom in the domain to set the global stiffness matrix array.
2. Looping the number of elements to generate the element connectivity array.
3. Looping the number of degrees of freedom per element to calculate element stiffness matrix.
4. Assembling the element stiffness matrix into the global stiffness matrix array in accord with the element connectivity array.
5. Stopping the loop of the number of degrees of freedom per element.
6. Stopping the loop of the number of elements.
7. Stopping the loop of the total number of degrees of freedom in the domain.

The element stiffness matrix (Smith et al., 2013) can be calculated from the following equations.

The element stiffness for a plane stress and plane strain problem:

$$[\boldsymbol{K_e}] = \int\int [\boldsymbol{B}]^{\boldsymbol{T}}[\boldsymbol{D}][\boldsymbol{B}]dxdy \tag{3.84}$$

Where **B** is the strain-displacement matrix and **D** is the stress-strain matrix.

The element stiffness for the axisymmetric problem:

$$[\boldsymbol{K_e}] = \int\int [\boldsymbol{B}]^{\boldsymbol{T}}[\boldsymbol{D}][\boldsymbol{B}]rdrdz \tag{3.85}$$

The element stiffness for the three-dimensional problem:

$$[\boldsymbol{K_e}] = \int\int\int [\boldsymbol{B}]^{\boldsymbol{T}}[\boldsymbol{D}][\boldsymbol{B}]dxdydz \tag{3.86}$$

Here, all element matrices are assembled fully in the full square global matrix. Since the global stiffness matrix is symmetric and sparse, these facts can be used to economize space and time in actual programming.

3.5.2 The Existing Standard FE Subroutines for the Element Stiffness Matrix Assembly

In this research, the existing standard FE subroutines (Smith and Griffiths, 2005) have been utilized to assemble the element stiffness matrix into the global stiffness matrix. The introduction and function of these subroutines are summarized in Table 3.2.

Table 3.2: The existing standard FE subroutines for the element stiffness matrix assembly (Smith and Griffiths, 2005)

Subroutine name	Parameter index	Function
formnf	*nf*	This subroutine returns the nodal freedom array *nf* from boundary conditions input of 0s and 1s.
formkb	*kb, km, g*	This subroutine returns the global full band matrix *kb* stored as a rectangle from the unsymmetrical element matrix *km* and steering vectors *g*.
fkdiag	*kdiag, g*	This subroutine returns the bandwidth *kdiag* for the rows of a skyline storage system from steering vectors *g*.
sample	*element, s, wt*	This subroutine returns the local coordinates *s* and weighting coefficients *wt* for the numerical integration of a finite element of type *element*.
deemat	*dee, e, v*	This subroutine returns the elastic stress-strain matrix *dee*. *e* and *v* are Young's modulus and Poisson's ratio.
shape_fun	fun, points, i	This subroutine returns the shape function *fun* at the i^{th} integrating point. *points* holds the local coordinates of the integrating points.

shape_der	der, points, i	This subroutine returns the shape function derivatives *der* at the i^{th} integrating point. *points* holds the local coordinates of the integrating points.
beemat	bee, deriv	This subroutine returns the strain-displacement matrix *bee* for shape function derivatives *der*.

3.6 Solution of Equilibrium Equation and Recovery of Results at Integrating Points

3.6.1 Solution Method

The solution of the equilibrium equation is performed after the element stiffness assembly. The key feature in FEM for creep damage analysis is dealing with the highly non-linear behaviour; the Newton-Raphson iterative method (Leonard, 1979) can be used in linearization of the non-linear problem. If the global stiffness matrix is assembled after the specification of boundary conditions, a typical global equilibrium equation (Smith et al., 2013) is given as:

$$[\boldsymbol{K}][\boldsymbol{U}] = [\boldsymbol{F}] \tag{3.87}$$

Where **K** is the global stiffness matrix, **U** is the global displacement matrix and **F** is the total loads.

Practical applications of the FEM lead to large systems of simultaneous linear algebraic equations. Two main methods that can be used in solving the equilibrium equations are: the direct solution method and the iterative solution method (Smith et al., 2013).

In the direct solution method, the Cholesky direct solution technique can be used to solve the sets of linear algebraic equations. In the iterative solution method, the Jacobi iteration, Gauss-Seidel and Conjugate Gradient iterative solution method can be used to solve the equations. Both the direct solution method and the iterative solution method have been implemented in FEM by many scientists; and the relevant standard FE subroutines (Smith and Griffiths, 2005) for the solution of equilibrium equation have been published and utilized in the development of HITSI.

Direct solution methods are generally used for problems of moderate size. For large problems, iterative methods require less computing time and hence they are preferable. Scientists can decide on the selection of the solution method for dealing with the non-linear problem according to the actual situation.

After the solution of the equilibrium equation, the nodal displacement can be computed and stored in the global displacement vector. The element nodal displacement can then be retrieved from the global displacement vector and element stiffness matrix re-computed. At this stage, the results such stress, strain and displacement of each element can be calculated and the results stored in the global result vector. The results at the integrating points can be retrieved from the global result vector.

3.6.2 The Existing Standard FE Subroutines in the Solution of the Equilibrium Equation

In this research, the existing standard FE subroutines (Smith and Griffiths, 2005) have been utilized to solve the equilibrium equation. The introduction and function of these subroutines are summarized in Table 3.3.

Table 3.3: The existing standard FE subroutines for the solution of the equilibrium equation (Smith and Griffiths, 2005)

Subroutine name	Parameter index	Function
sparin	*kv, kdiag*	This subroutine returns the Cholesky factorized vector *kv* stored as a skyline. *kdiag* holds the locations of the diagonal terms.
spabac	*kv, loads, kdiag*	This subroutine returns solution loads which overwrite the Right Hand Side (RHS) by forward and back substitution on the Cholesky factorized vector *kv* stored as a skyline. *kdiag* holds the locations of the diagonal terms.
sparin_gauss	*kv, kdiag*	This subroutine returns the Gaussian factorized vector *kv* stored as a skyline. *kdiag* holds the locations of the diagonal terms.

spabac_guass	*kv, loads, kdiag*	This subroutine returns solution loads which overwrite the RHS by forward and back substitution on the Gaussian factorized vector *kv* stored as a skyline. *kdiag* holds the locations of the diagonal terms.
guass_band	*pb, work*	This subroutine returns the Gaussian factorized unsymmetrical full band matrix *pb* and array *work*.
banred	*kv, neq*	This subroutine returns the transformed *neq* of the real symmetric band matrix to tri-diagonal form by Jacobi rotations
bacsub	*kv, loads*	This subroutine returns the transformed *loads* of the real symmetric band matrix to tri-diagonal form by Jacobi rotations

3.7 Summary

This chapter analyzed the fundamental requirements for the development of in-house FE software for creep damage analysis, and proposed the general methodology considerations and the FE algorithm involved with the creep damage constitutive equation, numerical integration method and explicit stress update FE algorithm for the development of HITSI.

The methods such as the set of element data; the element stiffness assembly; the solution of the equilibrium equation and recovery of results at the integrating points have been stated and the relevant existing standard FE subroutines which can be used in the development of HITSI are reported.

The author acknowledges that some important achievements and findings in this chapter have been published by Liu et al. (2012b) and Liu et al. (2013a) at various stages in this research.

Chapter 4 Programming the Finite Element Codes for in-house Software HITSI

4.1 Introduction

This chapter reports the development of the in-house FE software HITSI for creep damage analysis. In the development of HITSI, some existing standard FE subroutines (Smith and Griffiths, 2005) have been adopted from the author's supervisor Dr. Qiang Xu and a specific subroutine library provided by the author's colleague Feng Tan. The standard FE subroutines (Smith and Griffiths, 2005) are used in HITSI for the set of element data, element stiffness integration and assembly, solution of equilibrium equation and result recovery at integrating points. The subroutine library provided by Feng Tan for HITSI contains subroutines for the creep damage constitutive equation, the time integration with time step control, a nodal force calculator for the axisymmetric FE program and a data transfer interface between the in-house FE software HITSI and the pre- and post-processor FE software FEMGV. The use of Smith's standard FE subroutines and Feng Tan's specific subroutine library was originally planned in this research to make the development work more efficient. This software is developed based on the explicit FE algorithm.

Creep deformation can be regarded as a time-related plastic deformation and the process of the creep damage is an absolutely transient problem. The general method for solving the creep damage problem is based on the iteration of the elastic solution, via the Newton-Raphson iterative method (Leonard, 1979). The existing published elastic FE program and elastic-plastic FE program (Smith and Griffiths, 2005) adopted from author's supervisor Dr. Qiang Xu were investigated and studied by the author at the beginning of this research to achieve familiarization with the FE program.

In order to develop the work in a step by step fashion, as well as to be logical and efficient, the development strategy has been divided into eight stages:

1) Planning stage: The general flow diagram for the development of HITSI has been developed to describe the FE procedures used in programming HITSI for creep damage analysis.

2) Linear elastic stage: Adoption and modification of the existing linear plane stress version elastic FE program (Smith and Griffiths, 2005) to become familiar with the structure of the FE program and the use of the existing standard FE subroutines.

3) Non-linear elastic-plastic stage: Adoption and modification of the non-linear (single material and time independent) axisymmetric version of the elastic-plastic FE program (Smith and Griffiths, 2005) to become familiar with the FE techniques used in dealing with non-linearity.

4) Plane stress creep damage stage: The plane stress version FE program for time dependent creep damage analysis has been developed. The stress and creep damage field variables are updated with the time integration. Here, the creep damage constitutive equation subroutines provided by Feng Tan have been utilized. The time-step control is integrated into the time integration subroutines and the time integration subroutines provided by Feng Tan have been utilized in developing this FE program.

5) Plane strain creep damage stage: The plane strain version FE program for time dependent creep damage analysis has been developed. The stress and creep damage field variables are updated with the time integration. Here, the effort will be focused on the application of the plane strain constitutive matrix in developing this FE program.

6) Axisymmetric creep damage stage: The axisymmetric version FE program for time dependent creep damage analysis has been developed. The stress and creep damage field variables are updated with the time integration. Here, a nodal force calculator provided by Feng Tan for the generation of the equivalent nodal loads information has been utilized and the axisymmetric constitutive matrix has been used in developing this FE program.

7) Three-dimensional creep damage stage: The three-dimensional version FE program for time dependent creep damage analysis has been preliminarily developed. The stress and creep damage field variables are updated with the time integration. Here, the effort will be focused on the application of the three-dimensional constitutive matrix in developing this FE program.

8) Multi-materials zone creep damage stage: The multi-materials zone version FE codes for time dependent creep damage analysis has been developed. Here, the effort will be focused on expanding the scope of the software's application such as for creep damage analysis in weldment (multi-materials). The multi-materials zone FE codes have been integrated into the plane stress, plane strain, axisymmetric and three-dimensional FE program for HITSI.

The in-house FE software HITSI has been developed and the current version includes four main programs (plane stress, plane strain, axisymmetric and three-dimensional) because of the different characteristics of the constitutive matrix. This chapter primarily consists of ten sections: 1) Introduction; 2) The flow diagram for the development of HITSI; 3) Adoption and modification of the linear elastic FE program; 4) Adoption and modification of the non-linear elastic-plastic FE program; 5) Development of the plane stress version creep damage FE program; 6) Development of the plane strain version creep damage FE program; 7) Development of the axisymmetric version creep damage FE program; 8) Development of the three-dimensional version creep damage FE program; 9) Development of the multi-materials version creep damage FE program; 10) Summary.

4.2 The Flow Diagram for the Development of HITSI

The general flow diagram for the development of HITSI has been developed for the description of the FE procedures in creep damage analysis. It is noted that Hyde's research group (Becker et al., 2002) and Hayhurst's research group (Hayhurst and Krzeczkowski, 1979) have reported the development and use of their in-house FE software for creep damage analysis. Hall et al. (1996) have reported the flow diagram of their in-house software DAMAGE XX for creep damage analysis.

The flow diagram of the development of HITSI for creep damage mechanics has been developed based on several previous works: (Becker et al., 2002), (Hayhurst and Krzeczkowski, 1979) and (Hall et al., 1996); the flow diagram for the development of HITSI is shown in Figure 4.1.

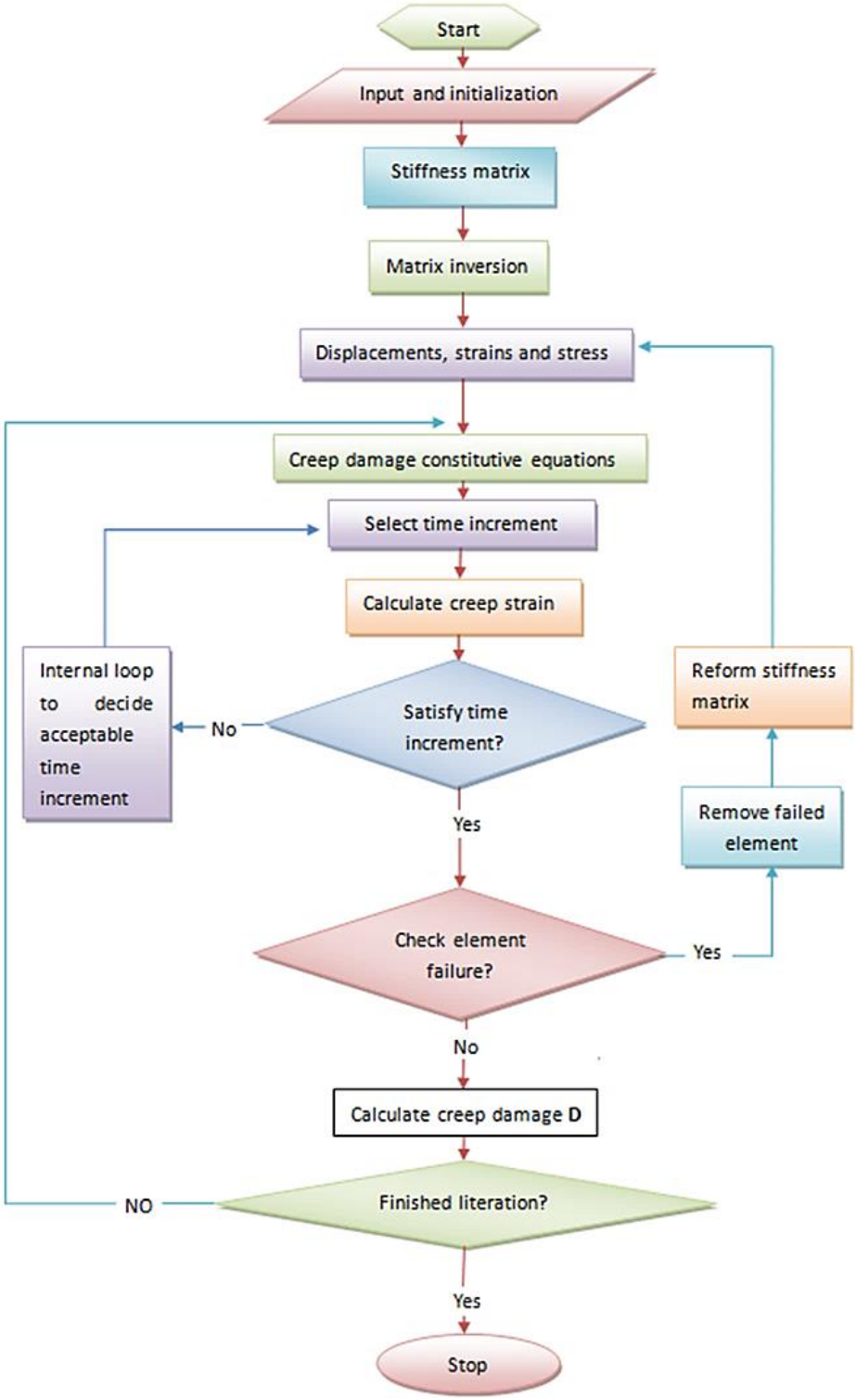


Figure 4.1: The flow diagram of the development of the in-house FE software HITSI

This flow diagram can be divided into five main aspects:

1. The mesh and element data information should be read by the program. The specifications, including nodes, elements, material properties, boundary conditions and the computational control parameters of the FE model, are input. The element stiffness matrices are assembled into the global matrix system and

the global nodal number information can be generated at this point. This corresponds to the development stages 1, 2 and 3.

2. The equilibrium equation should be solved after the input and the assembly of element stiffness matrix. The initial elastic stress and strain can be calculated. Each gauss point stress can be retrieved from the global result vector. These correspond to the development stages 2 and 3.

3. The creep damage constitutive equation should be embedded into the FE program. The creep strain rate and creep damage rate are integrated with regard to time and the time-step is controlled. Accuracy will suffer and instability may occur if the time increment is too large. The tolerance is pre-established; if the results are not satisfactory, the time increment is reduced by half of the previous time step iteration loop. These correspond to the development stages 4, 5 and 6.

4. The creep damage field variables such as creep strain, damage and stress should be updated with the time integration. Body loads are produced due to the creep deformation and these are added into the global loads vector for the stress updating. These correspond to the development stages 5, 6 and 7.

5. Stop execution and output results. The damage increases monotonically with time from the initial value zero to the critical value. The element cannot then support any further load and the Gaussian point in such an element has failed when damage value over the critical value. The program removes the failed element and the value of the element stiffness will be set to zero. Otherwise, the program calculates the creep damage until rupture occurs and the results output. This corresponds to the development stages 7 and 8.

4.3 Adoption and Modification of the Linear Elastic FE Program

4.3.1 The Structure of the Linear Elastic FE Program

Smith's linear elastic FE program: geotech / software / prog_fe / P50.F90 in (Smith and Griffiths, 2005) was adopted from the author's supervisor Dr. Qiang Xu for the plane stress of an elastic solid using uniform 3-node triangular elements numbered in the x direction. The new version elastic FE program has been developed based on modification of Smith's linear elastic FE program and additional modifications are

made in order to meet further development work. The modifications are summarized as follows:

- Single precision real variables and arrays are used in Smith's linear elastic FE program and subroutines; all single precision real variables and arrays are modified to double precision in the new version elastic FE program and subroutines.

- The mesh and element data information such as the element type, global coordinate information and element connection information used in Smith's linear elastic FE program are generated by his subroutines; the mesh and element data information used in new version elastic FE program are produced by the pre- and post-processor FE software FEMGV. The input method has been modified so that the new version elastic program can read the mesh and element data information directly.

- The output FE codes in Smith's linear elastic FE program are modified. The FE codes for the sequence and format of the results that are to be output are implanted in the new version elastic program to match the post-processing.

Some basic techniques for the development of FE software have been achieved through familiarization with Smith's elastic FE program. The techniques may be summarised as:

- The technique for reading the mesh and element data information.

- The technique for assembling the element stiffness matrix into the global stiffness system.

- The technique for integrating points to find nodal coordinates and the steering vector.

- The technique for factorising the global stiffness matrix and solving the equilibrium equation.

- The technique for recovering stresses at the central gauss point.

The subroutines in Tables 3.1, 3.2 and 3.3 can be used in this elastic FE program. The structure chart of the FE program for the analysis of linear elastic problem in Figure 4.2 corresponds to the development stages 1 and 2.

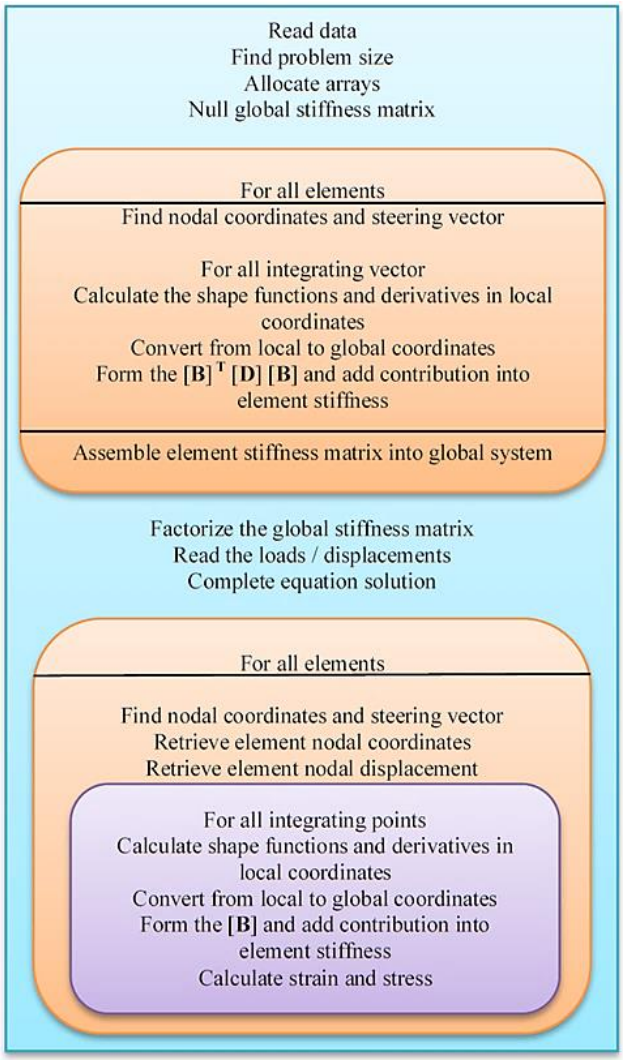


Figure 4.2: Structure chart of linear elastic FE program (Smith and Griffiths, 2005)

The structure chart in Figure 4.2 illustrates the sequence of the FE calculations for this program. The nodal coordinates, nodal numbering and boundary conditions can be obtained by the *read* statement after the declaration of the variables and the arrays. Then, the element stiffness matrix is integrated and assembled into the global stiffness matrix. Once all element stiffness matrices have been assembled, the equilibrium equation is solved. Lastly, the stress and strain at the integrating point can be calculated and recovered at this integration point. Specifications for programming the linear elastic FE program are illustrated in Section 4.3.2.

4.3.2 Specifications in Developing Linear Elastic FE Program

This new version linear elastic FE program is based on the modification of Smith's version linear elastic FE program: geotech / software / prog_fe / P50.F90 in (Smith and Griffiths, 2005). In this program, the variables and arrays are declared first; then, the program enters the "input and initialisation" stage. The declaration of variables and

arrays is summarized in Table 4.1 and Table 4.2, respectively. The FE codes of the declaration are presented in List 4.1.

Table 4.1: The declaration of variables in linear elastic FE program (Smith and Griffiths, 2005)

Variable name	Declaration
nels	number of elements
nce	number of elements in x direction
neq	number of degrees of freedom in mesh
nband	semi-bandwidth of grid
nn	number of nodes in the mesh
nr	number of restrained nodes
nip	number of integration points
nodof	number of degrees of freedom per node
nod	number of nodes per element
nst	number of stress terms
ndof	number of degrees of freedom per element
loaded_nodes	number of loaded nodes
i, k, iel	simple counters
ndim	number of dimensions
e	Young's modulus
v	Poisson's ratio
det	determinant of the Jacobian matrix
aa	the width of element
bb	the depth of element
element	element type

Table 4.2: The declaration of arrays in linear elastic FE program (Smith and Griffiths, 2005)

Array name	Declaration
kv	global stiffness matrix
loads	nodal loads and displacement
points	integrating point local coordinates

dee	stress strain matrix
coord	element nodal coordinates
jac	Jacobian matrix
der	shape function derivatives with respect to local coordinates
deriv	shape function derivatives with respect to global coordinates
weights	weighting coefficients
bee	strain displacement matrix
km	element stiffness matrix
eld	element nodal displacement
sigma	stress terms
g_coord	global nodal coordinates
nf	nodal freedom matrix
g	element steering vector
num	element node numbers vector
g_num	global element node number matrix
g_g	global element steering matrix

```
!------------------------------------------  Declaration  ----------------------------------------------------
!-----------------------------------Codes in linear elastic program---------------------------------------
99998 format(1X,I4)
integer::  nels,neq,nband,nn,nr,nip,nodof,nod, nst,ndof,i,k,iel,ndim,loaded_nodes
doubleprecision:: e,v
character(len=15) :: element
doubleprecision   ,allocatable :: kv(:),loads(:),points(:,:),dee(:,:),coord(:,:),          &
                                  jac(:,:), der(:,:),deriv(:,:), weights(:), bee(:,:),km(:,:),          &
                                  eld(:),sigma(:),g_coord(:,:)
 open (10,file='p1.dat',status='old',   action='read')
 open (11,file='p1.res',status='replace',action='write')
 read (10,*) element,nels,nn,nip,nodof,nod,nst,ndim
 ndof=nod*nodof
  allocate ( g_coord(ndim,nn),g_num(nod,nels),nf(nodof,nn), g_g(ndof,nels),       &
```

```
g(ndof),points(nip,ndim),weights(nip),coord(nod,ndim),          &
fun(nod), jac(ndim,ndim),dee(nst,nst),der(ndim,nod),            &
num(nod),km(ndof,ndof), eld(ndof),  bee(nst,ndof),              &
deriv(ndim,nod), sigma(nst),                                    &
tsigma(nst,nip,nels))
```

List 4.1: The FE codes of the declaration in linear elastic FE program (Smith and Griffiths, 2005)

After the declaration of variables and arrays in the linear elastic FE program, the program enters the element stiffness integration and assembly stage. Data concerning the mesh and its properties are presented together with the nodal freedom data. The total number of nodes and equations are read by main program. The elements are looped to generate the global array, which contains the element node numbers, the element nodal coordinates and the element steering vectors. The subroutine *formnf* is used to perform this task. The subroutine *sample* is called to return the local coordinates and weighting coefficients for integration. Subroutine *num_to_g* is used to find global coordinates and global node numbers.

In the element stiffness integration and assembly, subroutine *shape_der* is used to derive the shape functions with respect to the coordinates, subroutine *beemat* forms the strain-displacement matrix and subroutine *formkv* is used in assembling the element stiffness matrix into the global stiffness. The FE codes of element stiffness integration and assembly are presented in List 4.2.

```
!----------------------- Element stiffness integration and assembly ----------------------------
!-----------------------------------Codes in linear elastic program-----------------------------------------
do i=1, nn; read (10,*)k,g_coord(:,i); end do
do i=1, nels; read (10,*)k, g_num(:,i); end do
nf=1; read(10,*) nr ; if(nr>0) read(10,*)(k,nf(:,k),i=1,nr)
call formnf (nf); neq=maxval(nf); nband = 0
elements_1  : do iel =1,nels; num=g_num(:,iel); call num_to_g ( num , nf , g )
              g_g(:,iel)=g; if(nband<bandwidth(g))nband=bandwidth(g)
end do elements_1
dee=.0; dee(1,1)=e/(1.-v*v);dee(2,2)=dee(1,1);dee(3,3)=.5*e/(1.+v)
```

```
dee(1,2)=v*dee(1,1);dee(2,1)=dee(1,2); call sample(element,points,weights)
allocate( kv(neq*(nband+1)),loads(0:neq)); kv=0.0
elements_2: do iel = 1 , nels; num = g_num(:, iel); g = g_g( : , iel )
      coord = transpose(g_coord(:, num)) ; km=0.0
      gauss_pts_1: do i = 1 , nip
                  call shape_der(der,points,i) ; jac = matmul(der,coord)
                  det = determinant(jac); call invert(jac)
                 deriv = matmul(jac,der) ; call beemat (bee,deriv)
        km = km + matmul(matmul(transpose(bee),dee),bee) *det* weights(i)
    end do gauss_pts_1
   call formkv (kv,km,g,neq)
end do elements_2
```

List 4.2: The FE codes of element stiffness integration and assembly in elastic FE program (Smith and Griffiths, 2005)

The integration loop is entered after the assembly of the stiffness matrix. Subroutine *banred* and subroutine *bacsub* are called for the solution of the equilibrium equation. The stresses can be recovered at each gauss point through the use of subroutine *shape_der*, subroutine *invert* and subroutine *beemat*. The FE codes of the solution of the equilibrium equation and results recovery at the integrating points are presented in List 4.3.

```
!------------------- Solution of equilibrium equation and recover results -----------------------
!-----------------------------------Codes in linear elastic program-----------------------------------------
call banred(kv,neq) ;call bacsub(kv,loads)
nip = 1; deallocate(points,weights); allocate(points(nip,ndim),weights(nip))
call sample ( element , points , weights)
elements_3:do iel = 1 , nels
   num = g_num(: , iel);   coord =transpose( g_coord(: ,num) )
   g = g_g( : ,iel )   ;    eld=loads(g)
   gauss_pts_2: do i = 1 , nip
     call shape_der (der,points,i); jac= matmul(der,coord)
     call invert(jac) ;   deriv= matmul(jac,der)
```

```
    call beemat(bee,deriv); sigma = matmul (dee,matmul(bee,eld))
  end do gauss_pts_2
end do elements_3
```

List 4.3: The FE codes of the solution of equilibrium equation and results recovery at the integrating points in elastic FE program (Smith and Griffiths, 2005)

Adoption and modification of the existing linear elastic program is performed in order to obtain basic FE techniques for the development of HITSI for creep damage analysis. The validation of the new version elastic FE program is performed in Chapter 5.

4.4 Adoption and Modification of the Non-linear Elastic-plastic FE Program

4.4.1 The Structure of the Non-linear Elastic-plastic FE Program

The elastic-plastic FE program is developed as a further extension of the linear elastic version. Smith's version linear elastic-plastic FE program: geotech / software / prog_fe / P66.F90 in (Smith and Griffiths, 2005) is adopted from the author's supervisor Dr. Qiang Xu for calculating the axisymmetric 'un-drained' strain of an elastic-plastic solid using 8-node quadrilateral elements. This version elastic-plastic FE program is based on the modification of Smith's version elastic-plastic FE program and additional modifications are made in order to meet further development work. The modifications are summarized as follows:

- Single precision real variables and arrays are used in Smith's elastic-plastic FE program and subroutines; all single precision real variables and arrays are modified to double precision in this version elastic-plastic FE program and subroutines.

- The output FE codes in Smith's elastic-plastic FE program are modified. The FE codes for the sequence and format of the results that are to be output are implanted in this elastic-plastic program to match the post-processing.

The biggest difference between the linear elastic version FE program and elastic-plastic FE program is that the non-linear processes pose a very much greater analytical problem than do the linear processes. In practice, there is no direct method to solve the non-linear equation in mathematics. However, the Newton-Raphson iterative method

(Leonard, 1979) can be used in linearization of the elastic-plastic problem and such non-linear problem can be solved using this technique. A schematic diagram of the standard Newton-Raphson method is shown in Figure 4.3 and the modified Newton-Raphson method is shown in Figure 4.4.

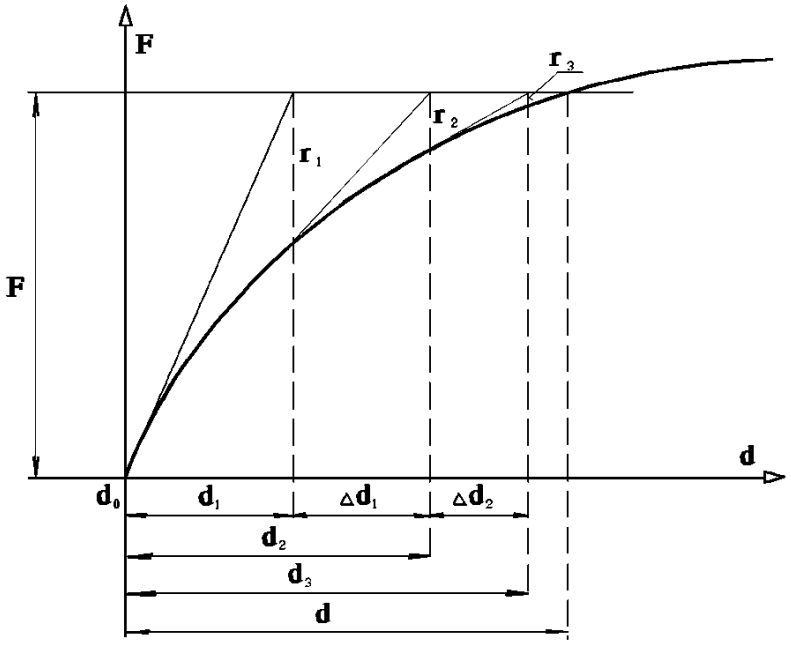


Figure 4.3: The Schematic of standard Newton-Raphson method (Copenhaver, 1980)

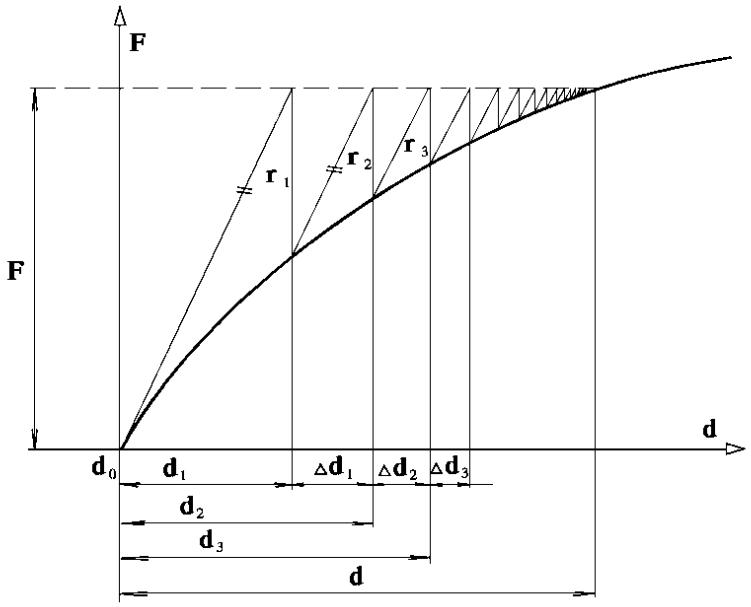


Figure 4.4: The Schematic of modified Newton-Raphson method (Copenhaver, 1980)

The standard Newton-Raphson method is usually called a constant stiffness method, in which non-linearity is caused by iteratively modifying the total loads vector (Smith et al., 2013). In the constant stiffness method, the global stiffness matrix is formed only

once and the subroutine *checon* is used to check plastic convergence in this FE program. The modified Newton-Raphson method is usually called a tangent stiffness method and the global stiffness matrix may be updated occasionally with fewer iterations per load step.

In this FE program, the subroutine *mocouf* together with subroutine *formm* and subroutine *mocouq* are used to check plastic convergence when using the tangent stiffness method. If small enough load steps are taken, the tangent stiffness method can save computing time because fewer iteration steps are needed in each load increment.

The creep deformation can be regarded as a time-related plastic deformation and the process of the creep damage is an absolutely transient problem. The general solution method of elastic-plastic and creep damage problems is very similar. Some techniques used in the development of HITSI may be obtained through familiarization with Smith's elastic-plastic FE program. The techniques have been summarized as:

1. The technique for adding the load or displacement increment loop
2. The technique for executing the plastic iteration loop
3. The technique for checking plastic convergence
4. The technique for checking whether yield is violated and update the gauss point stresses
5. The technique for computing the total body loads vector

The subroutines in Tables 3.1, 3.2 and 3.3 can be used in this FE program. The structure chart of the FE program for the analysis of the non-linear elastic-plastic solid problem is shown in Figure 4.5 correspond to the development stages 3 and 4.

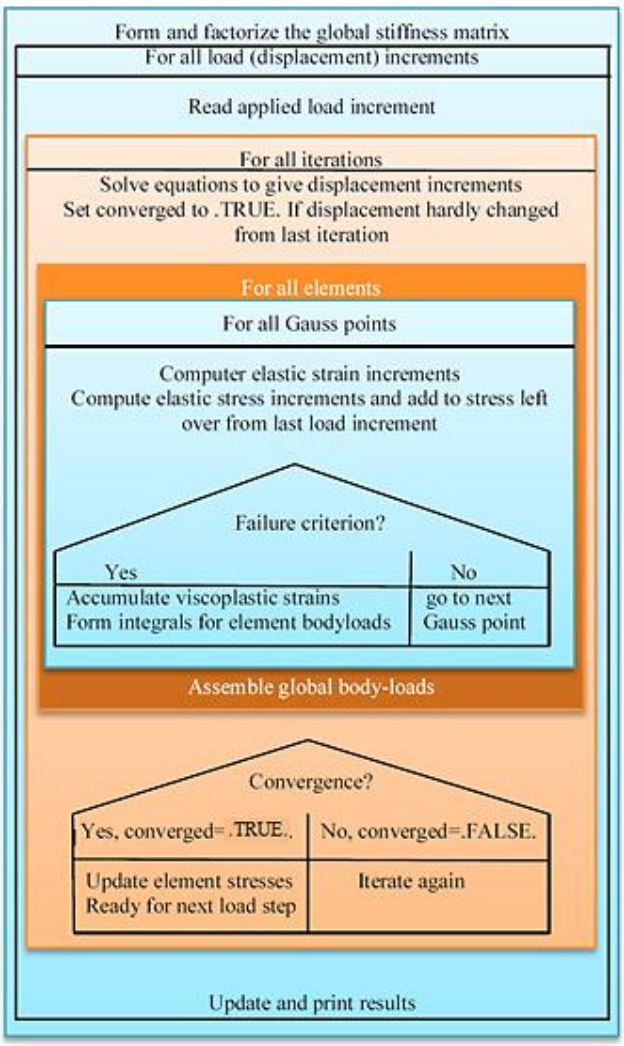


Figure 4.5: Structure chart of elastic-plastic FE program (Smith and Griffiths, 2005)

In a similar way to the linear elastic program the nodal coordinates, nodal numbering and boundary conditions can be obtained by the read statement after declaration of the variables and the arrays. Then, the element stiffness matrix is integrated and assembled into the global stiffness matrix. Once all element stiffness matrices have been assembled, the equilibrium equation is solved. Here, the difference between the linear elastic FE program and elastic-plastic FE program is that the load is variable and the solution of equilibrium equation is achieved based on the iterative method. Further specifications in developing the new version elastic-plastic FE program are illustrated in Section 4.4.2.

4.4.2 Specifications in Developing Non-linear Elastic-plastic FE Program

This non-linear elastic-plastic FE program is based on modification of Smith's version elastic-plastic FE program: geotech / software / prog_fe / P66.F90 in (Smith and Griffiths, 2005). In this FE program, the variables and arrays are declared first; then, the program enters the "input and initialisation" stage. The declaration of new variables and

arrays is summarized in Table 4.3 and Table 4.4, respectively. The FE codes of the declaration are presented in List 4.4.

Table 4.3: The declaration of new variables in elastic-plastic FE program (Smith and Griffiths, 2005)

New variable name	Declaration
nxe	number of elements in x direction
nye	number of elements in y direction
iters	the counters of plastic iterations
limit	plastic iteration ceiling
incs	number of load increments
converged	set to *.true.* if plastic iterations have converged
iy	simple counter
phi	friction angle
psi	dilation angle
dsbar	invariant
dq1, dq2, dq3	plastic potential derivative
lode_theta	lode angle
sigm	mean stress
pi	set to 3.1415
c	cohesion
dt	critical visco-plastic time step
snph	sine of phi
ptot	holds running total of applied pressure
tol	plastic convergence tolerance
presc	wall displacement increment
cons	consolidating stress
bulk	apparent fluid bulk modulus
radius	radius

Table 4.4: The declaration of new arrays in elastic-plastic FE program (Smith and Griffiths, 2005)

New array name	Declaration
bdylds	self-equilibrating global body loads
totd	holds running total of nodal displacement
evpt	holds running total of visco-plastic strains
oldis	nodal displacement from previous iteration
width	the width of the element
depth	the depth of the element
stress	stress term increment
storkv	holds augmented stiffness diagonal terms
eps	strain terms
bload	self-equilibrating element body loads
eload	integrating point contribution to *bload*
evp	plastic strain rate increment
devp	plastic force
m1, m2, m3	used to compute stress rate
flow	holds stress rate
tensor	holds running total of all integrating point stress terms
etensor	holds running total of all integrating point strain terms
pore	holds running total of all integrating point pore pressures
fun	shape function
no	fixed freedom numbers vector

```
!---------------------------------------- Declaration ----------------------------------------------
!--------------------------------Codes in linear elastic-plastic program----------------------------------
99998 format(1X,I4)
integer: :nels,nxe,nye,neq,nband,nn,nr,nip,nodof=2,nod=8,nst=4,ndof,                &
        i,j,k,iel,iters,limit,incs,iy,ndim=2,loaded_nodes
logical:: converged; character (len=15):: element='quadrilateral'
```

```
doubleprecision:: e,v,det,phi,c,psi,dt,f,dsbar,dq1,dq2,dq3,lode_theta,          &
sigm,pi,snph,bulk,cons,presc,ptot,radius,tol
doubleprecision ,allocatable:: kv(:),loads(:),points(:,:),bdylds(:),totd(:),    &
                  evpt(:,:,:),oldis(:),width(:),depth(:),stress(:),             &
                  dee(:,:),coord(:,:),jac(:,:),weights(:),storkv(:),            &
                  der(:,:),deriv(:,:),bee(:,:),km(:,:),eld(:),eps(:),           &
                  sigma(:),bload(:),eload(:),erate(:),g_coord(:,:),             &
                  evp(:),devp(:),m1(:,:),m2(:,:),m3(:,:),flow(:,:),             &
                 tensor(:,:,:),etensor(:,:,:),pore(:,:),fun(:)
integer, allocatable:: nf(:,:) , g(:), no(:) ,num(:), g_num(:,:) ,g_g(:,:)
open (10,file='p2.dat',status='old',action='read')
open (11,file='p2.res',status='replace',action='write')
read (10,*) phi,c,psi,e,v,bulk,cons, nels,nxe,nye,nn,nip
ndof=nod*nodof
allocate (nf(nodof,nn), points(nip,ndim),weights(nip),g_coord(ndim,nn),       &
          width(nxe+1),depth(nye+1),num(nod),evpt(nst,nip,nels),              &
          coord(nod,ndim),g_g(ndof,nels),tensor(nst,nip,nels),fun(nod),       &
          etensor(nst,nip,nels),dee(nst,nst),pore(nip,nels),stress(nst),      &
          jac(ndim,ndim),der(ndim,nod),deriv(ndim,nod),g_num(nod,nels),       &
          bee(nst,ndof),km(ndof,ndof),eld(ndof),eps(nst),sigma(nst),          &
          bload(ndof),eload(ndof),erate(nst),evp(nst),devp(nst),g(ndof),      &
         m1(nst,nst),m2(nst,nst),m3(nst,nst),flow(nst,nst))
```

List 4.4: The FE codes of the declaration in elastic-plastic FE program (Smith and Griffiths, 2005)

After declaration in the linear elastic finite element program, the program enters the element stiffness integration and assembly stage. Data concerning the mesh and its properties are presented together with the nodal freedom data. The total number of nodes and equations can be generated by subroutine *geometry_8qyv*. The subroutine *geometry_8qyv* produces rectangular 8-node elements with the numbering in the y direction. The elements are looped to generate the "global" array, which contains the element node numbers, the element nodal coordinates and the element steering vectors.

The subroutine *deemat* and subroutine *sample* are used in this stage. The subroutine *sample* is called to return the local coordinates and weighting coefficients for the integration. Subroutine *geometry_8qyv* and subroutine *num_to_g* are used to find global coordinates and global node numbers.

In element stiffness matrix integration and assembly, subroutine *shape_der* is used to derive the shape functions with respect to the coordinates and subroutine *shape_fun* returns the shape function at the integrating point. Then, subroutine *bmataxi* formed the strain-displacement matrix and subroutine *formkv* is used in assembling the element stiffness matrix into the global stiffness system. The FE codes of element stiffness integration and assembly are presented in List 4.5.

```
!-------------------------     Element stiffness integration and assembly     ------------------------------
!-----------------------------------Codes in elastic-plastic program----------------------------------------
nf=1; read (10,*) nr ; if(nr>0) read(10,*)(k,nf(:,k),i=1,nr)
call  formnf(nf); neq=maxval(nf); read(10,*) width , depth; nband = 0
elements_1:   do iel = 1, nels
              call  geometry_8qyv(iel,nye,width,depth,coord,num)
              call num_to_g(num,nf,g) ;    g_num(:,iel)=num
              g_coord(: , num )=transpose(coord); g_g( : , iel ) = g
             if (nband<bandwidth(g)) nband = bandwidth(g)
end do elements_1
allocate(kv(neq*(nband+1)),loads(0:neq),bdylds(0:neq),oldis(0:neq),totd(0:neq))
          kv=0.0; oldis=0.0; totd=0.0 ; tensor = 0.0; etensor = 0.0
call deemat(dee,e,v); call sample(element,points,weights)
do i=1,nst; do j=1,nst;if(i/=3.and.j/=3) dee(i,j)=dee(i,j)+bulk; end do; end do
pi = acos( -1. ); snph = sin(phi*pi/180.)
dt = 4.*(1.+ v)*(1.-2.*v)/(e*(1.-2.*v+snph*snph))
elements_2: do iel=1, nels
              num = g_num(: ,iel ) ; coord = transpose (g_coord(: ,num ))
              g = g_g( : ,iel );  km=0.0
        gauss_pts_1:  do i =1 , nip    ; call shape_fun(fun,points,i)
              call shape_der (der,points,i);  jac = matmul(der,coord)
```

```
        det = determinant(jac)  ;   call invert(jac)
        deriv=matmul(jac,der);call bmataxi(bee,radius,coord,deriv,fun)
      km=km+matmul(matmul(transpose(bee),dee),bee)*det*weights(i)*radius
      tensor(1:2,i,iel)=cons; tensor(4,i,iel)=cons
    end do gauss_pts_1
  call formkv (kv,km,g,neq)
end do elements_2
```

List 4.5: The FE codes of element stiffness integration and assembly in elastic-plastic FE program (Smith and Griffiths, 2005)

After all element stiffness matrices are assembled, the equilibrium equation is solved. Here, the difference between the linear elastic FE program and non-linear elastic-plastic FE program is that the loads information is variable. The loads and the senses of freedom are read by the main program. Then, the plastic convergence tolerance, iteration ceiling, the number of constant loads increment and the magnitude of loads increment are associated by the main program with a *read* statement.

The loads increment, iteration loops and integration loops are entered after the global stiffness matrix has been assembled. Subroutine *shape_der* generates the shape function with respect to the coordinates, subroutine *bmataxi* forms the strain-displacement matrix and subroutine *formkv* assembles the stiffness matrix into the global stiffness. Subroutine *banred* and subroutine *bacsub* perform the solution of the equilibrium equation. The FE codes for adding the loads increment loop and solving the equilibrium equation are presented in List 4.6.

```
!----------------  loads increment loop and solution of equilibrium equation  ----------
!-----------------------------------Codes in elastic-plastic program----------------------------------------
read(10,*) loaded_nodes ; allocate(no(loaded_nodes),storkv(loaded_nodes))
read(10,*)no , presc  , incs , tol , limit
      do i=1,loaded_nodes
        kv(nf(2,no(i)))=kv(nf(2,no(i)))+1.e20
       storkv(i)=kv(nf(2,no(i)))
      end do;  call banred(kv,neq)
call deemat(dee,e,v); load_increments: do iy=1,incs; ptot = presc * iy
```

```
iterations: do
    iters=iters+1;  loads = .0
    do i=1,loaded_nodes;loads(nf(2,no(i)))=storkv(i)*presc; end do
    loads = loads + bdylds  ;  call bacsub(kv,loads)
```

List 4.6: The FE codes for adding loads increment loop and solving equilibrium equation in elastic-plastic FE program (Smith and Griffiths, 2005)

In the iteration loop, the total loads vector is updated as a result of the loads increment. The input variables such as the convergence tolerance and the maximum number of iterations are used to control the loads increment loop. The subroutine *checon* can be used to check convergence. The body loads are updated at each iteration loop. At convergence, the stresses are updated for the next iteration loop. The running information such as stress terms, strain terms, nodal displacements and plastic strains are stored in the dynamic arrays. Subroutine *mocouf*, subroutine *mocouq* and subroutine *form* can be used to check whether yield is violated. The FE codes for checking convergence and yield are presented in List 4.7.

```
!----------------------    Checking convergence and whether yield is violated    ----------------------
!-----------------------------------Codes in elastic-plastic program----------------------------------------
call checon(loads,oldis,tol,converged)
if(iters==1)converged=.false.
elements_3: do iel = 1 , nels; bload=.0
  num = g_num( : , iel ) ; coord = transpose( g_coord( : , num ))
  g = g_g( : , iel ) ; eld = loads ( g )
  gauss_points_2 : do i = 1 , nip
      call shape_fun(fun,points,i); call shape_der ( der,points,i)
      jac=matmul(der,coord); det = determinant(jac)
      call invert(jac); deriv = matmul(jac,der); call bmataxi (bee,radius,coord,deriv,fun)
       eps=matmul(bee,eld); det = det * radius;  eps=eps-evpt(:,i,iel)
      sigma=matmul(dee,eps)  ;   stress=sigma+tensor(: , i , iel)
    call invar(stress,sigm,dsbar,lode_theta); call mocouf (phi, c , sigm, dsbar , lode_theta , f )
     if (f>=.0) then; call mocouq(psi,dsbar,lode_theta,dq1,dq2,dq3)
    call formm(stress,m1,m2,m3); flow=f*(m1*dq1+m2*dq2+m3*dq3)
```

```
      erate=matmul(flow,stress)
      evp=erate*dt; evpt(:,i,iel)=evpt(:,i,iel)+evp; devp=matmul(dee,evp)
    eload=matmul(devp,bee)    ; bload=bload+eload*det*weights(i); end if
  if (converged.or.iters==limit)  then
    tensor(:,i,iel)=stress; etensor(:,i,iel)=etensor(:,i,iel)+eps+evpt(:,i,iel)
    pore(i,iel)=(etensor(1,i,iel)+etensor(2,i,iel)+etensor(4,i,iel))*bulk
  end if
 end do gauss_points_2
 bdylds( g ) = bdylds( g ) + bload ; bdylds(0) = .0
end do elements_3; if(converged.or.iters==limit)exit; end do iterations
totd = totd + loads; if(iters==limit)stop; end do load_increments
```

List 4.7: The FE codes for checking convergence and yield in elastic-plastic FE program (Smith and Griffiths, 2005)

The calculated results are stored in dynamic arrays. The load increment method in this program is very similar to the time increment method in creep damage analysis. Therefore, this non-linear elastic-plastic version program is investigated to obtain the techniques for dealing with the non-linear problem in programming HITSI for creep damage mechanics. The validation of the FE codes for the elastic-plastic program is performed in Chapter 5.

4.5 Development of the Plane Stress Version Creep Damage FE Program

4.5.1 The Structure of the Creep Damage FE Program for Plane Stress Problem

The FE codes for plane stress version creep damage FE program have been developed based on the investigation of the elastic and elastic-plastic FE programs. The creep deformation can be regarded as a time-related plastic deformation and the process of the creep damage is an absolutely transient problem. In creep damage FEM, the time domain should be discretization. Some of the techniques used in developing this FE program are based on the investigation of the linear elastic and non-linear elastic-plastic version programs. Here, four aspects need to be addressed:

- The general FE algorithm for the creep damage problem

- The creep damage constitutive equation
- The numerical time integration method
- The updating of stress and creep damage field variables

The structure chart of the FE program in Figure 4.6 represents the creep analysis of the plane stress problem. This corresponds to the development stage 4.

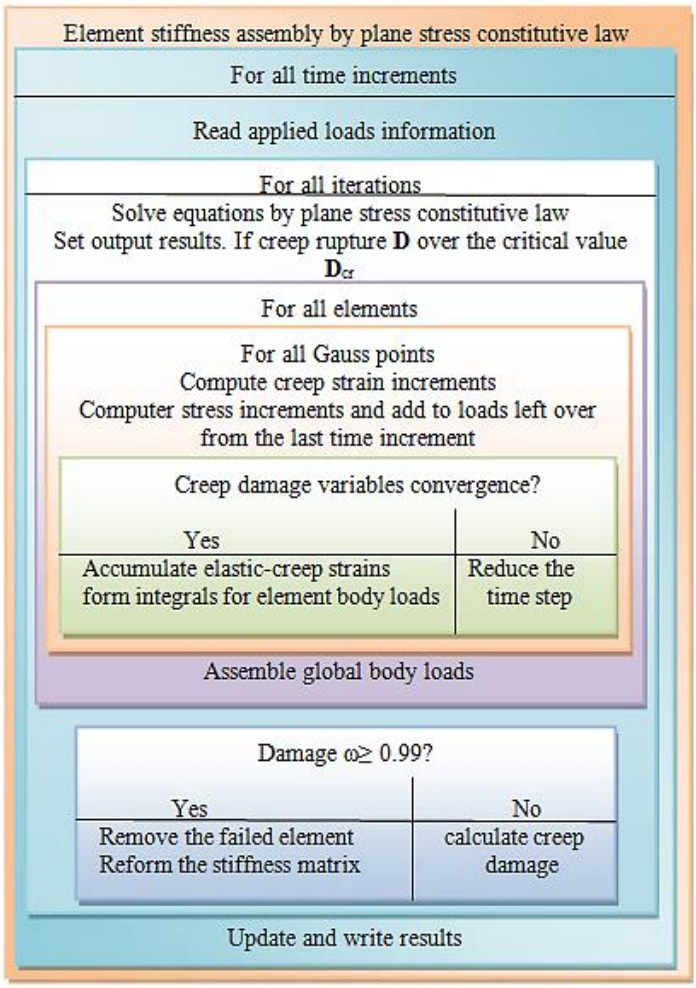


Figure 4.6: Structure chart of plane stress version FE program for creep damage problem

The initial stress method involves an explicit relationship between the increments of stress and increments of strain is used in developing this FE program. The initial elastic stresses are substituted into the creep damage constitutive equation and the creep damage fields such as creep strain rate and creep damage rate are integrated with respect to time. The FE algorithm for updating stress in Section 3.3.4 is used here for updating the total loads vector. The total loads vector consists of external applied loads and self-

equilibrating “body loads” at each time iteration. For each iterative step, compatibility and force equilibrium are explicitly satisfied.

In creep constitutive relationships, the complex creep damage phenomena can be depicted by a set of creep damage constitutive equations. The creep damage constitutive equation’s subroutines used in this program have been introduced in Section 3.3.2 and included in Feng Tan’s subroutine library. The library is based on the OOP approach and it contains constitutive equation subroutines for the Kachanov-Rabotnov-Hayhurst, the Kachanov-Rabotnov and the Kachanov-Rabotnov-Hayhurst-Xu methods. The user can select a different creep damage constitutive equation subroutine with a *call* statement according to the actual requirement.

In the numerical integration algorithm, the accuracy of the FE solution critically depends on the selection of the time step size associated with an appropriate integration method. The numerical integration algorithms used in this program have been reviewed in Chapter 2 and introduced in Section 3.3.3. The integration subroutines such as those for the Euler, the classical 4^{th} order Runge-Kutta, the Runge-Kutta-Merson and the Runge-Kutta-Fehlberg methods are programmed with an OOP approach and are included in Feng Tan’s subroutine library. The user can select a different integration algorithm with a *call* statement.

The creep damage increases monotonically with the time from the initial value zero to the critical value. An element that cannot support any further loads is said to be a failed element and the program will remove such elements. Here, the main program checks the creep damage value and forces the value of the element stiffness to zero when the creep damage value exceeds the critical value. Otherwise, the program calculates the creep damage until the rupture time occurs. Further specifications in the development of the FE program for creep damage analysis of the plane stress problem are illustrated in Section 4.5.2.

4.5.2 Specifications in Developing Plane Stress Version Creep Damage FE Program

This FE program is based on the development of the non-linear elastic-plastic FE program for creep analysis of the plane stress problem. In this program, the variables and arrays are declared first; then, the program enters the “input and initialisation” stage.

The declaration of new variables and arrays is summarized in Table 4.5 and Table 4.6, respectively. The FE codes for the declaration are presented in List 4.8.

Table 4.5: The declaration of new variables in creep damage FE program for plane stress problem

New variable name	Declaration
oppo	number of parameters in the creep damage constitutive equation
iy, iy, ii, ij	simple counters
iters	counts creep iterations
key1	output index for general geometry information
key2	output index for node number
key3	output index for element number
key4	output index for node displacements
key5	output index for body loads
key6	output index for the coordinates of integrating points
key7	output index for the stress
key8	output index for the strain
key9	output index for creep strain
key10	output index for creep damage
key11	output index for data transfer program
ESS	the equivalent stress
MPSS	the maximum principal stress
T	time increment
T0	the initial time point

Table 4.6: The declaration of new arrays in creep damage FE program for plane stress problem

New array name	Declaration
ABV	contains creep damage, strain, strain hardening, coarsening and material constant
crate	contains creep damage rate, strain rate, strain hardening rate and material constant rate
prop	element properties
evp	creep strain rate increment
devp	creep force
evpt	holds running total of creep strains
tabv	holds running total of creep damage, strain, strain hardening, coarsening and material constant
material	parameters in creep damage constitutive equation
tsigma	holds running total of stress terms
tevp	holds running total of creep strain increment
tdevp	holds running total of creep force
gc	integrating point coordinates
tgc	holds running total of integrating point coordinates
teps	holds running total of strain terms
bload	self-equilibrating element body loads due creep deformation
eload	integrating point creep force contribution to *bload*
bdylds	self-equilibrating global body loads due to creep deformation

```
!----------------------------------------- Declaration ----------------------------------------------
!----------------------Codes in plane stress version creep damage FE program----------------------
99998 format(1X,I4)
integer:: nels,neq,nband,nn,nr,nip,nodof,nod,nst,ndof,oppo, i,k,iel,ndim,          &
        loaded_nodes ,nprops,np_types,iy,j,ix,iters,ii,ij,key1=1,key2=2,           &
```

```
     key3=3,key4=4,key5=5,key6=6,key7=7,key8=8,key9=9,                    &
    key10=10,key11=9999
logical:: converged; character(len=15) :: element
doubleprecision::  ESS, MPSS,T,t0, e, v,det
doubleprecision, dimension (5):: ABV,crate
doubleprecision,  allocatable :: g_coord(:,:),points(:,:),weights(:),kv(:),          &
          km(:,:),dee(:,:),fun(:),der(:,:),jac(:,:),deriv(:,:),bee(:,:)  ,          &
         coord(:,:),loads(:),eld(:),sigma(:), prop(:,:), eps(:), evp(:),            &
       devp(:), bload(:),eload(:),evpt(:,:,:),bdylds(:),tabv(:,:,:),                &
       material(:),storkv(:,:),tsigma(:,:,:),tevp(:,:,:),tdevp(:,:,:),             &
       gc(:),tgc(:,:,:),teps(:,:,:)
integer,  allocatable :: g_num(:,:) ,nf(:,:),g(:),num(:),g_g(:,:),etype(:),no(:)
open (10,file='p1.dat',status='old',    action='read')
open (11,file='p1.res',status='replace',action='write')
read (10,*) element,nels,nn,nip,nodof,nod,nst,ndim,oppo; ndof=nod*nodof
allocate ( g_coord(ndim,nn),g_num(nod,nels),nf(nodof,nn), g_g(ndof,nels),        &
    g(ndof),points(nip,ndim),weights(nip),coord(nod,ndim),fun(nod),              &
    jac(ndim,ndim),der(ndim,nod),deriv(ndim,nod),bee(nst,ndof),                  &
    num(nod),km(ndof,ndof), eld(ndof), sigma(nst),etype(nels),                   &
    eps(nst), evp(nst), devp(nst), bload(ndof),eload(ndof),                      &
    evpt(nst,nip,nels), tabv(5,nip,nels), material(oppo),                        &
    tsigma(nst,nip,nels),tevp(nst,nip,nels), dee(nst,nst),                       &
    tdevp(nst,nip,nels),gc(ndim),tgc(ndim,nip,nels),                             &
    teps(nst,nip,nels))
```

List 4.8: The FE codes of the declaration in plane stress version creep damage FE program

After the declaration in the plane stress version creep damage FE program, the program enters the element stiffness integration and assembly stage. Data concerning the mesh and its properties are presented together with the nodal freedom data. The total number of nodes and elements are provided by the pre-process FE software FEMGV. The elements are looped to generate “global” arrays containing the element node numbers,

the element nodal coordinates and the element steering vectors. Here, the subroutine *num_to_g* is used to find global coordinates and global node numbers. Then, the subroutine *formnf* is called to return the nodal freedom array from the boundary conditions. Finally, subroutine *sample* is called to return the local coordinates and weighting coefficients for the numerical integration of the element type.

In element stiffness integration and assembly, subroutine *shape_der* is used to derive the shape functions with respect to the coordinates and subroutine *shape_fun* returns the shape function *fun* at the integrating point. Then, subroutine *beemat* returns the strain-displacement matrix for the shape function derivatives. Lastly, subroutine *formkv* is used to assemble the element stiffness matrix into the global stiffness. The FE codes of element stiffness integration and assembly is presented in List 4.9.

```
!------------------------    Element stiffness integration and assembly    ----------------------------
!----------------------Codes in plane stress version creep damage FE program----------------------
do i=1, nn; read (10,*) k, g_coord(:,i); end do
do i=1, nels; read (10,*)k, g_num(:,i); end do
nf=1; read(10,*) nr ; if(nr>0) read(10,*)(k, nf(:,k), i=1,nr)
call formnf (nf); neq=maxval(nf); nband = 0
elements_1:   do iel = 1, nels
              num=g_num(:,iel); call num_to_g(num,nf,g) ; g_g(:,iel)=g
              e=prop(1,etype(iel)); v=prop(2,etype(iel))
             if(nband<bandwidth(g))nband=bandwidth(g)
 end do elements_1
     dee=.0; dee(1,1)=e/(1.-v*v);dee(2,2)=dee(1,1);dee(3,3)=.5*e/(1.+v)
    dee(1,2)=v*dee(1,1);dee(2,1)=dee(1,2)
call sample(element,points,weights)
allocate( kv(neq*(nband+1)),loads(0:neq),bdylds(0:neq)); kv=0.0
elements_2: do iel = 1, nels; num = g_num(:, iel);  g = g_g( : , iel )
              coord = transpose(g_coord(:, num)) ; km=0.0
      gauss_pts_1: do i = 1, nip; call shape_fun(fun,points,i)
             call shape_der(der,points,i) ; jac = matmul(der,coord)
            det = determinant(jac); call invert(jac);  gc=matmul(fun,coord)
```

```
        tgc(:,i,iel)=gc; deriv = matmul(jac,der) ; call beemat (bee,deriv)
      km = km + matmul(matmul(transpose(bee),dee),bee) *det* weights(i)
    end do gauss_pts_1; call formkv (kv,km,g,neq)
  end do elements_2
```

List 4.9: The FE codes of element stiffness integration and assembly in plane stress version creep damage FE program

After the assembly of all element stiffness matrices, the equilibrium equation is solved. Here, the difference from the non-linear elastic-plastic program is that the loads variable in the elastic-plastic program is replaced by the time variable. The stress, strain, nodal displacement, body loads and creep damage field variables are updated with the time increment. The loads and the sense of freedoms are first read by the main program. Then, subroutine *bacsub* is called to solve the equilibrium equation and the initial stress can be recovered at this stage. The iterations of elements and integrating points are looped again to recover the initial stress at each integrating point. Subroutine *shape_der* is used to derive the shape functions with respect to the coordinates and subroutine *shape_fun* returns the shape function *fun* at the integrating point. The strain-displacement matrix for the shape function derivatives is returned by subroutine *beemat*; the displacement, stress and strain at each integrating point can be recovered through the above operation. The FE codes for solving the equilibrium equation and recovering the initial stress at each integrating point are presented in List 4.10.

```
!----------------     loads increment loop and solution of equilibrium equation     ----------
!----------------------Codes in plane stress version creep damage FE program----------------------
evpt(1,nip,nels)=0.0;  evpt(2,nip,nels)=0.0
evpt(3,nip,nels)=0.0;  evpt(4,nip,nels)=0.0
read (10,*) loaded_nodes;allocate(no(loaded_nodes),storkv(loaded_nodes,ndim))
read (10,*)(no(i),storkv(i,:),i=1,loaded_nodes)
      call banred(kv,neq); bdylds=.0; T=1; t0=0
        do i=1,nels; do j=1,nip; do k=1,5
          tabv(k,j,i)=0
        end do;  end do; end do
    tsigma=0; tevp=0; tdevp=0; do ii=1,2; ij=ii*iy; do iy=1,2; t0=t0+t
iters=0;bdylds=0;evpt=0; do i=1, loaded_nodes; loads(nf(:,no(i)))=storkv(i,:);end do
```

```
loads = loads + bdylds; call bacsub(kv,loads)
elements_3: do iel = 1 , nels; bload=.0
  num = g_num( : , iel ) ; coord = transpose( g_coord( : , num ))
  g = g_g( : , iel ) ; eld = loads ( g )
  integrating_pts_2 : do i = 1 , nip
       call shape_fun(fun,points,i); call shape_der ( der,points,i)
       jac=matmul(der,coord); call invert(jac)
      deriv = matmul(jac,der); call beemat(bee,deriv)
      eps=matmul(bee,eld); teps(:,i,iel)=eps;  det = determinant(jac)
      eps=eps-evpt(:,i,iel);  sigma=matmul(dee,eps)
      tsigma(:,i,iel)=sigma; abv=tabv(:,i,iel)
```

List 4.10: The FE codes for solving the equilibrium equation and recovering the initial stress at each integrating point in plane stress version creep damage FE program

Once the initial stress is calculated by the program, the time increment loop is executed. The equivalent stress and the maximum principal stress is obtained by substituting the initial stress into subroutine *rdmpes*. Then, the equivalent stress and the maximum principal stress are substituted into the creep damage constitutive equation for obtaining the creep damage variables. The Kachanov-Rabotnov creep damage constitutive equations with the Runge-Kutta integration method are used in calculating creep damage variables and the subroutine *RK4_KR* is used to perform the above tasks.

The creep strain is used in the calculation of body loads at each iteration loop. An element cannot support any further loads if the damage value increases from the initial value zero to the critical value and such an element is said to be a failed element and the program removes it. Here, the main program checks the creep damage value; the program forces the value of the element stiffness to zero when the creep damage value exceeds the critical value. Lastly, the element body loads are assembled into the global body load vector and the global body load is substituted into the equilibrium equation for the stress updating. The FE codes for calculating creep damage variables and stress updating are presented in List 4.11.

```
!------------------------     creep damage variables and stress updating    ----------------------------
!----------------------Codes in plane stress version creep damage FE program----------------------
```

```
call rdmpes (sigma,mpss,ess);  do ix=1, oppo; material(ix)=prop(ix+2,etype(iel))
end do; call RK4_KR (abv,crate,t,t0,sigma,ess,mpss,material)
    tabv(:,i,iel)=abv; evp(1)=crate(1)*t;evp(2)=crate(2)*t
    evp(3)=crate(3)*2*t; evp(4)=crate(4)*t; tevp(:,i,iel)=evp
    evpt(:,i,iel)=evpt(:,i,iel)+evp; devp=matmul(dee,evp)
   tdevp(:,i,iel)=devp; eload=matmul(devp,bee)
   bload=bload+eload*det*weights(i)
 end do integrating_pts_2
bdylds( g ) = bdylds( g )+ bload ; bdylds(0) = 0
end do elements_3; end do; end do
```

List 4.11: The FE codes for calculating creep damage variables and stress updating in plane stress version creep damage FE program

The program calculates the creep damage until the rupture time occurs. The results such as the coordinates of integrating points, node displacement, stress, strain, creep strain and creep damage are output for the post-processing. The FE codes for the output of all calculated results are presented in List 4.12.

```
!----------------------------------------     output the results    ----------------------------------------------
!----------------------Codes in plane stress version creep damage FE program----------------------
write(11,99998) key1; write(11,*)ndim,nn,nod,nels,element,nst,nip,t0,e,v
write(11,99998) key2;  do k=1,nn; write(11,*) k ,g_coord(:,k);end do
write(11,99998) key3;  do k = 1 , nels; write(11,*) k ,g_num(:,k), key1; end do
write(11,99998) key4; do k=1,nn; write(11,*) k,loads(nf(:,k)); end do
write(11,99998) key5; do k=1,nn; write(11,*) k,bdylds(nf(:,k)); end do
write(11,99998) key7;  do i=1,nels; write(11,*) i
      do j=1,nip; write(11,*) j, tsigma(:,j,i); end do;  end do
write(11,99998) key8; do i=1,nels; write(11,*) i
      do j=1,nip; write(11,*)  j, teps(:,j,i); end do; end do
write(11,99998) key9; do i=1,nels;  write(11,*) i
      do j=1,nip; write(11,*)  j, evpt(:,j,i);  end do; end do
write(11,99998) key10;  do i=1,nels; write(11,*) i
     do j=1,nip; write(11,*)  j, tabv(5,j,i); end do;  end do
```

write(11,99998) key11; end program planestress

List 4.12: The FE codes for the output of all calculated results in plane stress version creep damage FE program

The running results are stored in dynamic arrays and they can be output by a *write* statement in main program. The plain strain, axisymmetric and three-dimension versions of the FE programs are based on the plane stress version creep damage FE program. The main difference between them is the different constitutive relationship and this has been introduced in Chapter 3. The validation of this FE program is performed in Chapter 5.

4.6 Development of the Plane Strain Version Creep Damage FE Program

4.6.1 The Structure of the Creep Damage FE Program for Plane Strain Problem

The FE codes for the plane strain version creep damage FE program have been developed based on the plane stress version creep damage FE program. The FE algorithm of plane strain is very similar to that of plane stress for creep damage analysis, the main difference being the constitutive matrix. In the plane strain problem, a typical slice of, say, an underground tunnel that lies along the z axis might deform in essentially plane strain conditions. The plane stress and plane strain constitutive matrices are presented in Section 3.4.1. In this program, the element stiffness integration, element stiffness assembly and the solution of the general equilibrium equation are focused on the plane strain constitutive relationship. The structure chart of the FE program in Figure 4.7 is presented for the creep damage analysis of the plane strain problem. This corresponds to the development stage 5.

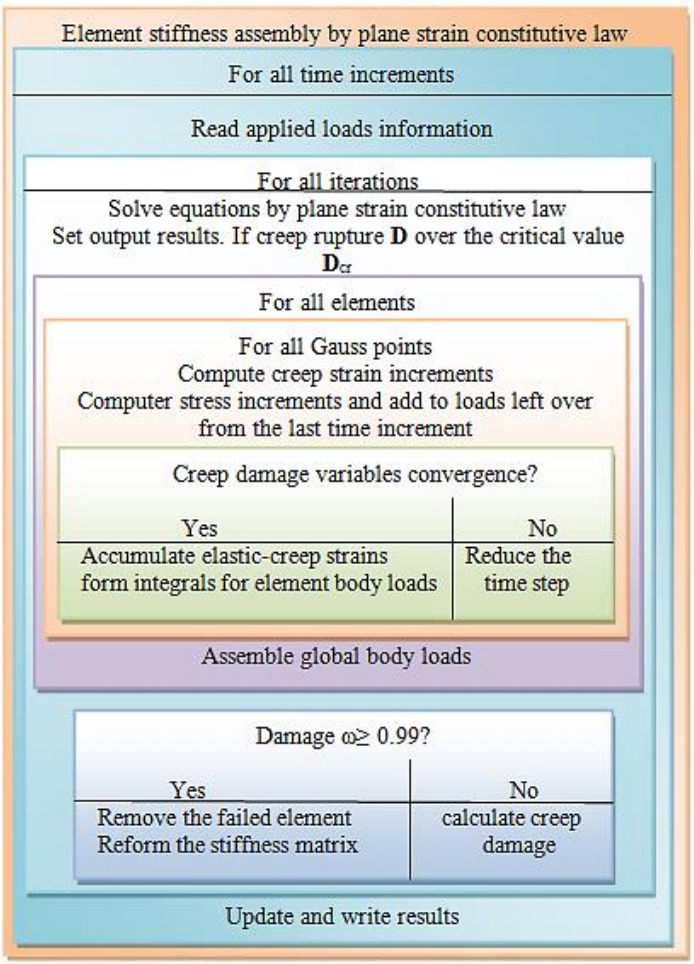


Figure 4.7: Structure chart of plane strain version FE program for creep damage problem

The creep damage constitutive equation's subroutines with the time integration method used in this program are included in Feng Tan's subroutine library and they have been introduced in Section 3.3.2 and Section 3.3.3. The library contains subroutines for the Kachanov-Rabotnov-Hayhurst, the Kachanov-Rabotnov and the Kachanov-Rabotnov-Hayhurst-Xu equations. The integration subroutines such as the Euler, the classical 4th order Runge-Kutta, the Runge-Kutta-Merson and the Runge-Kutta-Fehlberg methods are used in this program. The user can select a different creep damage constitutive equation subroutine and different time integration method with a *call* statement according to the actual requirement.

The FE algorithm for updating the stress and creep damage field variables, introduced in Section 3.3.4, is used in developing this program. Further specifications in the development of the FE program for the creep damage analysis of the plane strain problem are illustrated in Section 4.6.2.

4.6.2 Specifications in Developing Creep Damage FE Program for Plane Strain Problem

This FE program has been developed for creep damage analysis of the plane strain problem. The different two-dimensional element types for this program have been described in Section 3.4.2. In this program, the variables and arrays are declared first; then, the program enters the "input and initialisation" stage. The declaration of variables has been summarized in Table 4.5 and a new dynamic array used in this program is shown in Table 4.7. The FE codes of the declaration are presented in List 4.13.

Table 4.7: The declaration of new array in creep damage FE program for plane strain problem

New array name	Declaration
kdiag	diagonal term location vector

```
!------------------------------------------ Declaration ----------------------------------------------------
!----------------------Codes in plane strain version creep damage FE program-----------------------
99998 format(1X,I4)
integer:: nels,neq,nn,nr,nip,nodof,nod,nst,ndof,oppo,i,k,iel,ndim,                    &
          loaded_nodes ,nprops,np_types,iy,j,ix,iters,ii,ij,key1=1,                   &
         key2=2, key3=3,key4=4,key5=5,key6=6,key7=7,key8=8,                          &
         key9=9,key10=10,key11=9999
doubleprecision:: ESS, MPSS,T,t0, e, v,det
doubleprecision, dimension (5):: ABV,crate
character(len=15) :: element
doubleprecision, allocatable :: g_coord(:,:),points(:,:),weights(:),kv(:),           &
                                km(:,:),dee(:,:),fun(:),der(:,:),jac(:,:),deriv(:,:),bee(:,:) ,  &
                               coord(:,:),loads(:),eld(:),sigma(:), prop(:,:), eps(:), evp(:),  &
                               devp(:), bload(:),eload(:),evpt(:,:,:),bdylds(:),tabv(:,:,:),    &
                               material(:),storkv(:,:),tsigma(:,:,:),tevp(:,:,:),tdevp(:,:,:),  &
                               gc(:),tgc(:,:,:),teps(:,:,:)
integer, allocatable :: g_num(:,:) ,nf(:,:),g(:),num(:),g_g(:,:),etype(:),no(:),     &
                        kdiag(:)
```

```
open (10,file='p2.dat',status='old',   action='read')
open (11,file='p2.res',status='replace',  action='write')
read (10,*) element,nels,nn,nip,nodof,nod,nst,ndim,oppo
ndof=nod*nodof
allocate ( g_coord(ndim,nn),g_num(nod,nels),nf(nodof,nn), g_g(ndof,nels),      &
     g(ndof),points(nip,ndim),weights(nip),coord(nod,ndim),fun(nod),           &
     jac(ndim,ndim),der(ndim,nod),deriv(ndim,nod),bee(nst,ndof),               &
     num(nod),km(ndof,ndof), eld(ndof), sigma(nst),etype(nels),                &
     eps(nst), evp(nst), devp(nst), bload(ndof),eload(ndof),                   &
     evpt(nst,nip,nels), tabv(5,nip,nels), material(oppo),                     &
     tsigma(nst,nip,nels),tevp(nst,nip,nels), dee(nst,nst),                    &
     tdevp(nst,nip,nels),gc(ndim),tgc(ndim,nip,nels),                          &
     teps(nst,nip,nels))
```

List 4.13: The FE codes of the declaration in plane strain version creep damage FE program

After the declaration, the program enters the element stiffness integration and assembly stage. Data information concerning the mesh and its properties is provided by the pre-processing FE software FEMGV. The elements are looped to generate "global" arrays containing the element node numbers, the element nodal coordinates and the element steering vectors. Here, the subroutine *num_to_g* is used to find global coordinates and global node numbers. Then, the subroutine *formnf* is called to return the nodal freedom array from boundary conditions and subroutine *fkdiag* is used to hold the diagonal term location. Finally, subroutine *sample* is called to return the local coordinates and weighting coefficients for the numerical integration of an element type.

In the element stiffness integration and assembly, subroutine *beemat* is used to return the strain-displacement matrix for the shape function derivatives and subroutine *shape_der* derives the shape functions with respect to the coordinates. Then, subroutine *shape_fun* returns the shape function *fun* at the integrating point and subroutine *deemat* returns the elastic stress-strain. Lastly, subroutine *fsparv* is used in assembling the element stiffness matrix into the global stiffness matrix. The FE codes of element stiffness integration and assembly are presented in List 4.14.

```
!------------------------  Element stiffness integration and assembly  ----------------------------
!----------------------Codes in plane strain version creep damage FE program----------------------
do i=1, nn; read (10,*) k, g_coord(:,i); end do
do i=1, nels; read (10,*)k, g_num(:,i); end do
nf=1; read(10,*) nr ; if(nr>0) read(10,*)(k, nf(:,k), i=1,nr)
call formnf (nf); neq=maxval(nf); allocate(kdiag(neq)) ;  kdiag = 0
elements_1:   do iel = 1, nels
                num=g_num(:,iel); call num_to_g(num,nf,g) ; g_g(:,iel)=g
               call fkdiag(kdiag,g)
end do elements_1
      kdiag(1)=1; do i=2,neq; kdiag(i)=kdiag(i)+kdiag(i-1); end do
allocate( kv(kdiag(neq)),loads(0:neq),bdylds(0:neq)); kv=0.0
call sample(element,points,weights)
elements_2: do iel = 1, nels; num = g_num(:, iel);  g = g_g( : , iel )
                coord = transpose(g_coord(:, num)) ; km=0.0
       gauss_pts_1: do i = 1, nip; e=prop(1,etype(iel)); v=prop(2,etype(iel));
                call deemat(dee,e,v); call shape_fun(fun,points,i)
                call shape_der(der,points,i) ; jac = matmul(der,coord)
               det = determinant(jac); call invert(jac);  gc=matmul(fun,coord)
               tgc(:,i,iel)=gc; deriv = matmul(jac,der) ; call beemat (bee,deriv)
          km = km + matmul(matmul(transpose(bee),dee),bee) *det* weights(i)
      end do gauss_pts_1; call fsparv (kv,km,g,kdiag)
 end do elements_2
```

List 4.14: The FE codes of element stiffness integration and assembly in plane strain version creep damage FE program

After the assembly of all element stiffness matrices, the equilibrium equation is solved. Here, unlike the plane stress version creep damage FE program, the solution loads overwrite the RHS by forward and back substitution on the Cholesky factorized global stiffness matrix stored as a skyline. The stress, strain, nodal displacement, body loads and creep damage field variables are updated with the time increment of. Then, subroutine *sparin* and subroutine *spabac* are called to solve the equilibrium equation

and the initial stress will be given at this stage. The iteration of elements and integrating points is looped again to recover the initial stress at each integrating point. Subroutine *shape_der* derives the shape functions with respect to the coordinates and subroutine *shape_fun* returns the shape function at the integrating point. The strain-displacement matrix for the shape function derivatives is returned by subroutine *beemat;* the displacement, stress and strain at each integrating point can be recovered through the above operation. The FE codes for solving the equilibrium equation and recovering the initial stress at each integrating point are presented in List 4.15.

```
!----------------  loads increment loop and solution of equilibrium equation  ----------------
!----------------------Codes in plane strain version creep damage FE program----------------------
evpt(1,nip,nels)=0.0;  evpt(2,nip,nels)=0.0
evpt(3,nip,nels)=0.0;  evpt(4,nip,nels)=0.0
read (10,*) loaded_nodes;allocate(no(loaded_nodes),storkv(loaded_nodes,ndim))
read (10,*)(no(i),storkv(i,:),i=1,loaded_nodes)
        call sparin (kv,kdiag); bdylds=.0; T=1; t0=0
            do i=1,nels; do j=1,nip; do k=1,5
               tabv(k,j,i)=0
            end do;  end do; end do
       tsigma=0; tevp=0; tdevp=0; do ii=1,2; ij=ii*iy; do iy=1,2; t0=t0+t
iters=0;bdylds=0;evpt=0; do i=1, loaded_nodes; loads(nf(:,no(i)))=storkv(i,:);end do
loads = loads + bdylds; call spabac(kv,loads,kdiag)
elements_3: do iel = 1 , nels; bload=.0
  num = g_num( : , iel ) ; coord = transpose( g_coord( : , num ))
  g = g_g( : , iel ) ; eld = loads ( g )
  integrating_pts_2 : do i = 1 , nip
       call shape_fun(fun,points,i); call shape_der ( der,points,i)
       jac=matmul(der,coord); det = determinant(jac)
      call invert(jac);deriv = matmul(jac,der)
      call beemat(bee,deriv);eps=matmul(bee,eld)
      eps=eps-evpt(:,i,iel);  sigma=matmul(dee,eps)
      tsigma(:,i,iel)=sigma;
```

```
abv=tabv(:,i,iel)
```

List 4.15: The FE codes for solving the equilibrium equation and recovering the initial stress at each integrating point in plane strain version creep damage FE program

Once the initial stress is obtained by the program, the time increment loop will be executed. The equivalent stress and the maximum principal stress are calculated by supplying the initial stress to subroutine *rdmpes*. The equivalent stress and the maximum principal stress will be substituted into the creep damage constitutive equation for the calculation of creep damage variables. The subroutine *EULER_KR* is used to calculate the creep damage variables and the creep strain is used in the calculation of element body loads at each element. Lastly, the element body loads are assembled into the global body loads and the global body loads will be substituted into the equilibrium equation for updating the stress. The FE codes for calculating creep damage variables and stress updating are presented in List 4.16.

```
!------------------------  creep damage variables and stress updating  ---------------------------
!----------------------Codes in plane strain version creep damage FE program-----------------------
call rdmpes (sigma,mpss,ess);  do ix=1, oppo; material(ix)=prop(ix+2,etype(iel))
end do; call EULER_KR (abv,crate,t,t0,sigma,ess,mpss,material)
    tabv(:,i,iel)=abv; evp(1)=crate(1)*t;evp(2)=crate(2)*t
    evp(3)=crate(3)*2*t; evp(4)=crate(4)*t; tevp(:,i,iel)=evp
    evpt(:,i,iel)=evpt(:,i,iel)+evp; devp=matmul(dee,evp)
   tdevp(:,i,iel)=devp; eload=matmul(devp,bee)
   bload=bload+eload*det*weights(i)
 end do integrating_pts_2
bdylds( g ) = bdylds( g )+ bload ; bdylds(0) = 0
end do elements_3; end do; end do
```

List 4.16: The FE codes for calculating creep damage variables and stress updating in plane strain version creep damage FE program

The creep damage increases monotonically with the time until the rupture time occurs. The results such as the coordinates of integrating points, node displacement, stress, strain, creep strain and creep damage are output by the main program for post-

processing. The FE codes for the output of all calculated results are presented in List 4.17.

```
!---------------------------------------  output the results  ----------------------------------------------
!----------------------Codes in plane stress version creep damage FE program----------------------
write(11,99998) key1; write(11,*)ndim,nn,nod,nels,element,nst,nip,t0,e,v
write(11,99998) key2;  do k=1,nn; write(11,*) k ,g_coord(:,k);end do
write(11,99998) key3;  do k = 1 , nels; write(11,*) k ,g_num(:,k), key1; end do
write(11,99998) key4; do k=1,nn; write(11,*) k,loads(nf(:,k)); end do
write(11,99998) key5; do k=1,nn; write(11,*) k,bdylds(nf(:,k)); end do
write(11,99998) key7;  do i=1,nels; write(11,*) i
      do j=1,nip; write(11,*) j, tsigma(:,j,i); end do;  end do
write(11,99998) key8; do i=1,nels; write(11,*) i
      do j=1,nip; write(11,*)  j, teps(:,j,i); end do; end do
write(11,99998) key9; do i=1,nels;  write(11,*) i
      do j=1,nip; write(11,*)  j, evpt(:,j,i);  end do; end do
write(11,99998) key10;  do i=1,nels; write(11,*) i
     do j=1,nip; write(11,*)  j, tabv(5,j,i); end do;  end do
write(11,99998) key11; end program planestrain
```

List 4.17: The FE codes for the output of all calculated results in plane strain version creep damage FE program

The running results are stored in dynamic arrays and they can be output at the end of program by a *write* statement. This program is based on the plane stress version creep damage FE program and the validation is performed in Chapter 5.

4.7 Development of the Axisymmetric Version Creep Damage FE Program

4.7.1 The Structure of the Creep Damage FE Program for Axisymmetric Problem

The FE codes for the axisymmetric version creep damage FE program have been developed based on the plane strain version creep damage FE program. The FE algorithm for both axisymmetric and plane strain version program for creep damage analysis is very similar, the main variation being the different constitutive relationship.

In the axisymmetric problem, a constant value of displacement in the circumferential direction should be considered. The axisymmetric constitutive matrix in the FE method is presented in Section 3.4.1. In this program, the element stiffness integration, element stiffness assembly and solution of the general equilibrium equation are focused on the axisymmetric constitutive relationship. The structure chart of the FE program in Figure 4.8 is presented for the creep analysis of the axisymmetric problem. This corresponds to the development stage 6.

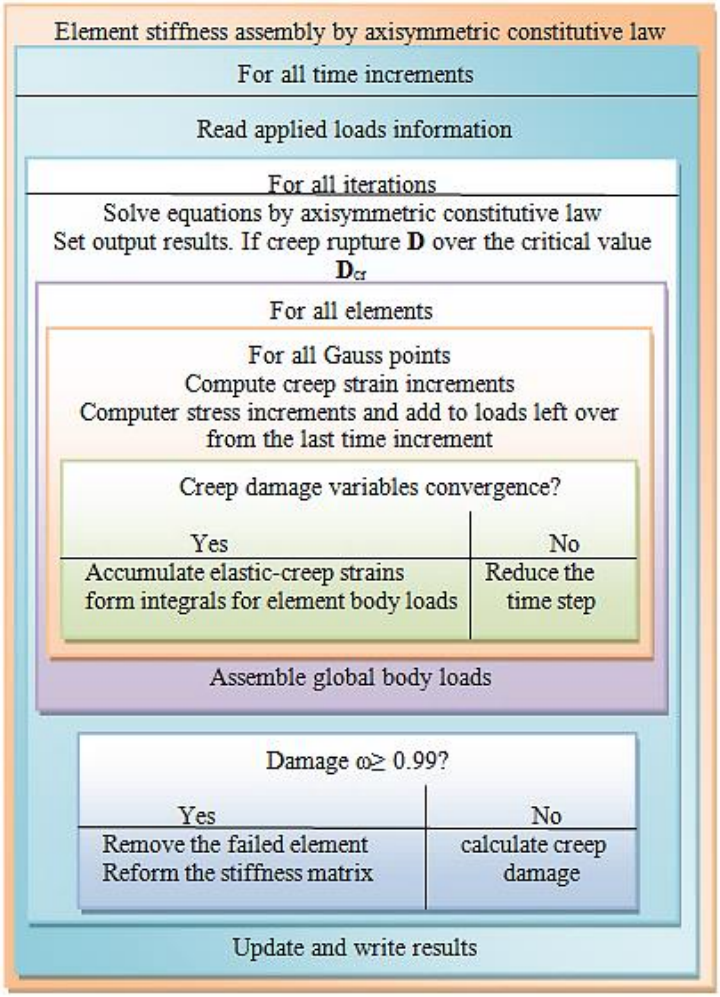


Figure 4.8: Structure chart of axisymmetric version FE program for creep damage problem

The creep damage constitutive equation's subroutines with the time integration method used in this program are included in Feng Tan's subroutine library and they have been introduced in Section 3.3.2 and Section 3.3.3. The library contains subroutines for the Kachanov-Rabotnov-Hayhurst, the Kachanov-Rabotnov and the Kachanov-Rabotnov-Hayhurst-Xu equations. The integration subroutines such as the Euler, the classical 4th order Runge-Kutta, the Runge-Kutta-Merson and the Runge-Kutta-Fehlberg methods

are used in this program. The user can select a different creep damage constitutive equation subroutine and different time integration method with a *call* statement according to the actual requirement.

The FE algorithm for updating the stress and creep damage field variables, introduced in Section 3.3.4, is used in developing this program. Further specifications in the development of the FE program for the creep damage analysis of the plane strain problem are illustrated in Section 4.7.2.

4.7.2 Specifications in Developing Creep Damage FE Program for Axisymmetric Problem

This FE program has been developed for the creep damage analysis of the axisymmetric problem. The different two-dimensional element types for this program have been described in Section 3.4.2. In this program, the variables and arrays are declared first; then, the program enters the "input and initialisation" stage. Since a constant value of displacement in the circumferential direction should be considered, one extra variable and one extra dynamic array are used in the development of this program. The declaration of the new variable and array is summarized in Table 4.8 and Table 4.9, respectively. The FE codes of the declaration are presented in List 4.18.

Table 4.8: The declaration of new variable in creep damage FE program for axisymmetric problem

New variable name	Declaration
radius	r-coordinates of Gauss point

Table 4.9: The declaration of new array in creep damage FE program for axisymmetric problem

New array name	Declaration
S	component of stress

```
!---------------------------------------- Declaration ----------------------------------------------
!----------------------Codes in axisymmetric version creep damage FE program----------------------
99998 format(1X,I4)
integer:: nels,neq,nn,nr,nip,nodof,nod,nst,ndof,oppo,i,k,iel,ndim,                    &
```

```
 loaded_nodes ,nprops,np_types,iy,j,ix,iters,ii,ij,key1=1,                &
key2=2, key3=3,key4=4,key5=5,key6=6,key7=7,key8=8,                       &
key9=9,key10=10,key11=9999
doubleprecision::  ESS, MPSS,T,t0, e, v,det, radius
doubleprecision, dimension (5):: ABV,crate
doubleprecision, dimension (4)::S
character(len=15) :: element
doubleprecision,  allocatable :: g_coord(:,:),points(:,:),weights(:),kv(:),  &
                  km(:,:),dee(:,:),fun(:),der(:,:),jac(:,:),deriv(:,:),bee(:,:) ,  &
                  coord(:,:),loads(:),eld(:),sigma(:), prop(:,:), eps(:), evp(:),  &
                  devp(:), bload(:),eload(:),evpt(:,:,:),bdylds(:),tabv(:,:,:),  &
                  material(:),storkv(:,:),tsigma(:,:,:),tevp(:,:,:),tdevp(:,:,:),  &
                  gc(:),tgc(:,:,:),teps(:,:,:)
integer,  allocatable :: g_num(:,:) ,nf(:,:),g(:),num(:),g_g(:,:),etype(:),no(:),
open (10,file='p3.dat',status='old',    action='read')
open (11,file='p3.res',status='replace',   action='write')
read (10,*) element,nels,nn,nip,nodof,nod,nst,ndim,oppo
ndof=nod*nodof
allocate ( g_coord(ndim,nn),g_num(nod,nels),nf(nodof,nn), g_g(ndof,nels),  &
     g(ndof),points(nip,ndim),weights(nip),coord(nod,ndim),fun(nod),  &
     jac(ndim,ndim),der(ndim,nod),deriv(ndim,nod),bee(nst,ndof),  &
     num(nod),km(ndof,ndof), eld(ndof), sigma(nst),etype(nels),  &
     eps(nst), evp(nst), devp(nst), bload(ndof),eload(ndof),  &
     evpt(nst,nip,nels), tabv(5,nip,nels), material(oppo),  &
     tsigma(nst,nip,nels),tevp(nst,nip,nels), dee(nst,nst),  &
     tdevp(nst,nip,nels),gc(ndim),tgc(ndim,nip,nels),  &
     teps(nst,nip,nels))
```

List 4.18: The FE codes of the declaration in axisymmetric version creep damage FE program

After the declaration, the program enters the element stiffness integration and assembly stage. Data information concerning the mesh and its properties are provided by the pre-

processing FE software FEMGV. The elements are looped to generate "global" arrays containing the element node numbers, the element nodal coordinates and the element steering vectors. Here, the subroutine *formnf* is called to return the nodal freedom array from boundary conditions. Then, the subroutine *num_to_g* is used to find global coordinates and global node numbers and subroutine *sample* is called to return the local coordinates and weighting coefficients for the numerical integration of an element type.

In the element stiffness integration and assembly, subroutine *deemat* returns the elastic stress-strain matrix. Then, subroutine *shape_der* is used to derive the shape functions with respect to the coordinates and subroutine *shape_fun* returns the shape function at the integrating point. The subroutine *bmataxi* is called to form the strain-displacement matrix. Lastly, subroutine *formkv* is used to assemble the element stiffness matrix into the global stiffness system. The FE codes of element stiffness integration and assembly are presented in List 4.19.

```
!------------------------  Element stiffness integration and assembly  ----------------------------
!---------------------Codes in axisymmetric version creep damage FE program----------------------
do i=1, nn; read (10,*) k, g_coord(:,i); end do
do i=1, nels; read (10,*)k, g_num(:,i); end do
nf=1; read(10,*) nr ; if(nr>0) read(10,*)(k, nf(:,k), i=1,nr)
call formnf (nf); neq=maxval(nf); nband=0
elements_1:  do iel = 1, nels
              num=g_num(:,iel); call num_to_g(num,nf,g) ; g_g(:,iel)=g
             if(nband<bandwidth(g))nband=bandwidth(g)
end do elements_1
call sample(element,points,weights)
allocate( kv(neq*(nband+1)),loads(0:neq),bdylds(0:neq)); kv=0.0
elements_2: do iel = 1, nels; num = g_num(:, iel);  g = g_g( : , iel )
              coord = transpose(g_coord(:, num)) ; km=0.0
              e=prop(1,etype(iel)); v=prop(2,etype(iel))
             call deemat(dee,e,v); do ix=1, oppo
             material(ix)=prop(ix+2,etype(iel)); end do
      gauss_pts_1: do i = 1, nip; call shape_fun(fun,points,i)
```

```
        call shape_der(der,points,i); jac=matmul(der,coord)
        det = determinant(jac); call invert(jac);  gc=matmul(fun,coord)
        tgc(:,i,iel)=gc; deriv = matmul(jac,der)
       call bmataxi(bee,radius,coord,deriv,fun); det =det*radius
    km = km + matmul(matmul(transpose(bee),dee),bee) *det* weights(i)
  end do gauss_pts_1; call formkv (kv,km,g,neq)
end do elements_2
```

List 4.19: The FE codes of element stiffness integration and assembly in axisymmetric version creep damage FE program

After the assembly of all element stiffness matrices, the equilibrium equation is solved. Here, the difference with the plane strain version creep damage FE program is the constitutive relationship, and the solution method in the axisymmetric program is based on Jacobi rotations (Smith et al., 2013). The stress, strain, nodal displacement, body loads and creep damage field variables are updated with the time increment. Then, subroutine *banred* and subroutine *bacsub* are called to solve the equilibrium equation and the initial stress will be given at this time. The iteration of elements and integrating points is looped again to recover the initial stress at each integrating point. Subroutine *shape_der* is used to derive the shape function with respect to the coordinates and subroutine *shape_fun* returns the shape function at the integrating point. The strain-displacement matrix for the shape function derivatives is returned by subroutine *bmataxi*; the displacement, stress and strain at each integrating point can be recovered through the above operation. The FE codes for solving the equilibrium equation and recovering the initial stress at each integrating point are presented in List 4.20.

```
!----------------  loads increment loop and solution of equilibrium equation  ----------
!---------------------Codes in axisymmetric version creep damage FE program----------------------
read (10,*) loaded_nodes;allocate(no(loaded_nodes),storkv(loaded_nodes,ndim))
read (10,*)(no(i),storkv(i,:),i=1,loaded_nodes)
      call banred(kv,neq); bdylds=.0; T=1; t0=0
        do i=1,nels; do j=1,nip; do k=1,5
          tabv(k,j,i)=0
       end do;  end do; end do
    tsigma=0; tevp=0; tdevp=0; do ii=1,2; ij=ii*iy; do iy=1,2; t0=t0+t
```

```
iters=0;bdylds=0;evpt=0; do i=1, loaded_nodes; loads(nf(:,no(i)))=storkv(i,:);end do
loads = loads + bdylds; call bacsub(kv,loads)
elements_3: do iel = 1 , nels; bload=.0
  num = g_num( : , iel ) ; coord = transpose( g_coord( : , num ))
  g = g_g( : , iel ) ; eld = loads ( g )
  integrating_pts_2 : do i = 1 , nip
       call shape_fun(fun,points,i); call shape_der ( der,points,i)
       jac=matmul(der,coord); call invert(jac)
      deriv=matmul(jac,der); call bmataxi(bee,radius,coord,deriv,fun)
      eps=matmul(bee,eld); teps(:,i,iel)=eps; det=det*radius
     eps=eps-evpt(:,i,iel)); sigma=matmul(dee,eps)
     tsigma(:,i,iel)=sigma; abv=tabv(:,i,iel)
```

List 4.20: The FE codes for solving the equilibrium equation and recovering the initial stress at each integrating point in axisymmetric version creep damage FE program

Once the initial stress is calculated by the program, the time increment loop will be executed. In order to obtain the equivalent stress and the maximum principal stress subroutine *stress_deviator_2D*, subroutine *equivalent_stress_2D* and subroutine *max_principal_stress_2D*, developed by the author's colleague Feng Tan, are used. The component of stress can be obtained by substituting the initial stress into subroutine *stress_deviator_2D*. The equivalent stress and the maximum principal stress are calculated by subroutine *equivalent_stress_2D* and subroutine *max_principal_stress_2D*, respectively. The equivalent stress and the maximum principal stress will be substituted into the creep damage constitutive equation for the calculation of creep damage variables. The subroutine *Euler_KR* is used to calculate the creep damage variables and the creep strain is used in the calculation of body loads at each element. Lastly, the element body loads are assembled to get the global body loads vector and the global body loads will be substituted into the equilibrium equation for updating stress. The FE codes for calculating creep damage variables and stress updating are presented in List 4.21.

```
!----------------------- creep damage variables and stress updating ----------------------
!--------------------Codes in axisymmetric version creep damage FE program----------------------
do ix=1, oppo; material(ix)=prop(ix+2,etype(iel))
```

```
call STRESS_DEVIATOR_2D (sigma,S); call equivalent_stress_2D (S,ESS)
call  max_PRINCIPAL_STRESS_2D (sigma,MPSS)
call EULER_KR (abv,crate,t,t0,sigma,ess,mpss,material)
    if(tabv(5,i,iel)>=0.99)then; tabv(5,i,iel)=0.9999
    tevp(1,i,iel)=0.0; tevp(2,i,iel)=0.0; tevp(3,i,iel)=0.0
    tevp(4,i,iel)=0.0; km=0.0; else; tabv(:,i,iel)=abv
     tevp(:,i,iel)=evp; evpt(:,i,iel)=evpt(:,i,iel)+evp
     end if
     devp=matmul(dee,evp); tdevp(:,i,iel)=devp
   eload=matmul(devp,bee); bload=bload+eload*det*weights(i)
 end do integrating_pts_2
bdylds( g ) = bdylds( g )+ bload ; bdylds(0) = 0
end do elements_3; end do; end do
```

List 4.21: The FE codes for calculating creep damage variables and stress updating in axisymmetric version creep damage FE program

The creep damage increases monotonically with the time until the rupture time occurs. The results such as the coordinates of integrating points, node displacement, stress, strain, creep strain and creep damage are output by the main program for the post-processing. The FE codes for the output of all calculated results are presented in List 4.22.

```
!---------------------------------------     output the results    ----------------------------------------------
!---------------------Codes in axisymmetric version creep damage FE program----------------------
write(11,99998) key1; write(11,*)ndim,nn,nod,nels,element,nst,nip,t0,e,v
write(11,99998) key2;  do k=1,nn; write(11,*) k ,g_coord(:,k);end do
write(11,99998) key3;  do k = 1 , nels; write(11,*) k ,g_num(:,k), key1; end do
write(11,99998) key4; do k=1,nn; write(11,*) k,loads(nf(:,k)); end do
write(11,99998) key5; do k=1,nn; write(11,*) k,bdylds(nf(:,k)); end do
write(11,99998) key7;  do i=1,nels; write(11,*) i
      do j=1,nip; write(11,*) j, tsigma(:,j,i); end do;  end do
write(11,99998) key8; do i=1,nels; write(11,*) i
      do j=1,nip; write(11,*)  j, teps(:,j,i); end do; end do
```

```
write(11,99998) key9; do i=1,nels;  write(11,*) i
    do j=1,nip; write(11,*)  j, evpt(:,j,i);  end do; end do
write(11,99998) key10;  do i=1,nels; write(11,*) i
    do j=1,nip; write(11,*)  j, tabv(5,j,i); end do;  end do
write(11,99998) key11; end program axisymmetric
```

List 4.22: The FE codes for the output of all calculated results in axisymmetric version creep damage FE program

The running results are stored in dynamic arrays and they can be output at the end of program by a *write* statement. This program based on the plane strain version creep damage FE program and the validation is performed in Chapter 5.

4.8 Development of the Three-dimensional Version Creep Damage FE Program

4.8.1 The Structure of the Creep Damage FE Program for Three-dimensional Problem

The three-dimensional FE program for creep damage analysis has been preliminarily developed based on the two-dimensional version FE program. In three dimensions, the number of degrees of freedoms of a three-dimensional element is much larger than of a two-dimensional element; thus, it will result in a very large number of simultaneous equations for the solution of practical three-dimensional problems. The conventional storage and solution strategies (Smith et al., 2013) can be used in developing this FE program; however, the skyline stiffness vector requires many more locations than that of a two-dimensional problem and the bandwidth of the equations system may become very large leading to huge computer storage requirements (Hall, 1990). In order to improve the computing efficiency for three-dimensional problems, a one dimension variable-bandwidth storage method (Smith et al., 2013) to store the data of the global matrix, so that the storage is minimised, is used in programming this three-dimensional FE program for creep damage analysis.

The general FE algorithm for the three-dimensional and two-dimensional programs for creep damage analysis is very similar. In actual programming, the constitutive matrix for the two cases is different. The three-dimensional constitutive matrix is introduced in Section 3.4.1. Thus, different strategies for element stiffness integration, element

stiffness assembly and the solution of general equilibrium equation are used in developing the three-dimensional version FE program. The structure chart of the FE program in Figure 4.9 is for the creep damage analysis of the three-dimensional problem. This corresponds to the development stage 7.

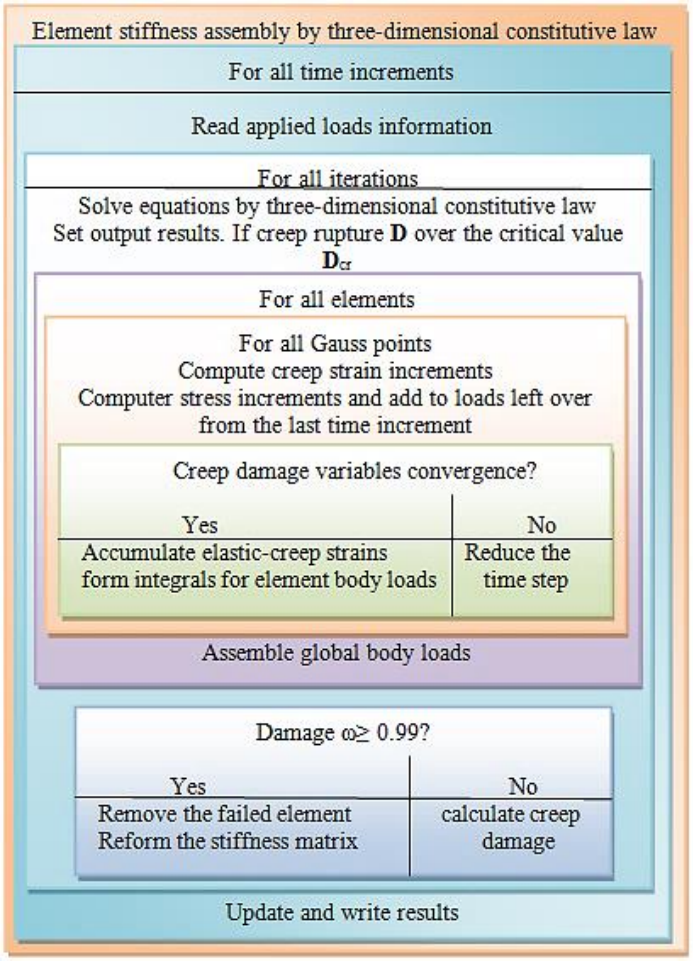


Figure 4.9: Structure chart of three-dimensional version FE program for creep damage problem

The Kachanov-Rabotnov creep damage constitutive equation and Euler integration method are used in this FE program. The FE algorithm introduced in Section 3.3.4 for updating stress and creep damage field variables is utilized. Further specifications in the development of FE program for the creep damage analysis of the three-dimensional problem are illustrated in Section 4.8.2.

4.8.2 Specifications in Developing Creep Damage FE Program for Three-dimensional Problem

This FE program has been developed for the creep damage analysis of three-dimensional problem. Several element types in Section 3.4.2 can be utilized in

developing this FE program. In this program, the variables and arrays are declared first; then, the program enters the "input and initialisation" stage. The declaration of the new variable and the new dynamic arrays has been summarized in Table 4.10 and Table 4.11, respectively. The FE codes of the declaration are presented in List 4.23.

Table 4.10: The declaration of new variable in creep damage FE program for three-dimensional problem

New variable name	Declaration
fixed_nodes	number of fixed nodes

Table 4.11: The declaration of new arrays in creep damage FE program for three-dimensional problem

New array name	Declaration
sense	hold fixed-node information
value	applied nodal load weightings

```
!------------------------------------------ Declaration ----------------------------------------------------
!--------------------Codes in three-dimensional version creep damage FE program--------------------
99998 format(1X,I4)
integer:: nels,neq,nn,nr,nip,nodof,nod,nst,ndof,oppo,i,k,iel,ndim,                    &
          loaded_nodes ,nprops,np_types,iy,j,ix,iters,ii,ij,key1=1,                   &
         key2=2, key3=3,key4=4,key5=5,key6=6,key7=7,key8=8,                          &
         key9=9,key10=10,key11=9999, fixed_nodes
doubleprecision:: ESS, MPSS,T,t0, e, v,det
doubleprecision, dimension (5):: ABV,crate
character(len=15) :: element
doubleprecision, allocatable :: g_coord(:,:),points(:,:),weights(:),kv(:),           &
                  km(:,:),dee(:,:),fun(:),der(:,:),jac(:,:),deriv(:,:),bee(:,:) ,    &
                 coord(:,:),loads(:),eld(:),sigma(:), prop(:,:), eps(:), evp(:),     &
                 devp(:), bload(:),eload(:),evpt(:,:,:),bdylds(:),tabv(:,:,:),       &
                 material(:),storkv(:,:),tsigma(:,:,:),tevp(:,:,:),tdevp(:,:,:),     &
                 load_store(:),value(:) ,gc(:),tgc(:,:,:),teps(:,:,:)
```

```
integer,  allocatable :: g_num(:,:) ,nf(:,:),g(:),num(:),g_g(:,:),etype(:),no(:),          &
        kdiag(:),sense(:), node(:)
open (10,file='p4.dat',status='old',    action='read')
open (11,file='p4.res',status='replace',   action='write')
read (10,*) element,nels,nn,nip,nodof,nod,nst,ndim,oppo
ndof=nod*nodof
allocate ( g_coord(ndim,nn),g_num(nod,nels),nf(nodof,nn), g_g(ndof,nels),       &
        g(ndof),points(nip,ndim),weights(nip),coord(nod,ndim),fun(nod),         &
        jac(ndim,ndim),der(ndim,nod),deriv(ndim,nod),bee(nst,ndof),             &
        num(nod),km(ndof,ndof), eld(ndof), sigma(nst),etype(nels),              &
        eps(nst), evp(nst), devp(nst), bload(ndof),eload(ndof),                 &
        evpt(nst,nip,nels), tabv(7,nip,nels), material(oppo),                   &
        tsigma(nst,nip,nels),tevp(nst,nip,nels), dee(nst,nst),                  &
        tdevp(nst,nip,nels),gc(ndim),tgc(ndim,nip,nels),                        &
        teps(nst,nip,nels))
```

List 4.23: The FE codes of the declaration in three-dimensional version creep damage FE program

After the declaration, the program enters the element stiffness integration and assembly stage. The elements are looped to generate the "global" arrays for containing the element node numbers, the element nodal coordinates and the element steering vectors. Here, the subroutine *num_to_g* is used to find global coordinates and global node numbers. Then, subroutine *sample* is called to return the local coordinates and weighting coefficients for the numerical integration of a finite element type. In the element stiffness integration and assembly, subroutine *shape_der* is used to derive the shape function with respect to the coordinates. The subroutine *beemat* returns the strain-displacement matrix for the shape function derivatives. Lastly, subroutine *fsparv* is used to assemble the element stiffness matrix into the global stiffness. The FE codes of element stiffness integration and assembly are presented in List 4.24.

```
!------------------------  Element stiffness integration and assembly  ----------------------------
!--------------------Codes in three-dimensional version creep damage FE program-------------------
do i=1, nn; read (10,*) k, g_coord(:,i); end do
```

```
do i=1, nels; read (10,*)k, g_num(:,i); end do
nf=1; read(10,*) nr ; if(nr>0) read(10,*)(k, nf(:,k), i=1,nr)
call formnf (nf); neq=maxval(nf); allocate(kdiag(neq) ;   kdiag = 0
elements_1:   do iel = 1, nels
              num=g_num(:,iel); call num_to_g(num,nf,g)
              g_g(:,iel)=g ; call fkdiag(kdiag,g)
end do elements_1
kdiag(1)=1; do i=2,neq; kdiag(i)=kdiag(i)+kdiag(i-1); end do
allocate( kv(kdiag(neq)),loads(0:neq),bdylds(0:neq), load_store(0:neq)); kv=0.0
call sample(element,points,weights)
elements_2: do iel = 1, nels; num = g_num(:, iel);  g = g_g( : , iel )
              coord = transpose(g_coord(:, num)) ; km=0.0
      gauss_pts_1: do i = 1, nip; e=prop(1,etype(iel))
              v=prop(2,etype(iel)); call deemat(dee,e,v)
              call shape_der(der,points,i); jac=matmul(der,coord)
              det = determinant(jac); call invert(jac);  gc=matmul(fun,coord)
              tgc(:,i,iel)=gc; deriv = matmul(jac,der); call beemat (bee,deriv)
         km = km + matmul(matmul(transpose(bee),dee),bee) *det* weights(i)
    end do gauss_pts_1; call fsparv (kv,km,g,kdiag)
 end do elements_2
```

List 4.24: The FE codes of element stiffness integration and assembly in three-dimensional version creep damage FE program

After the assembly of all element stiffness matrices, the equilibrium equation is solved. Here, the one dimension variable-bandwidth method (Smith et al., 2013) is used to store the data of the global matrix. The stress, strain, nodal displacement, body loads and creep damage field variables are updated with the time increment. Then, subroutine *sparin* and subroutine *spabac* are called to solve the equilibrium equation and the initial stress will be given at this stage. The iteration of elements and integrating points is looped again for recovering the initial stress at each integrating point. Subroutine *shape_der* is used to derive the shape function with respect to the coordinates. The strain-displacement matrix for the shape function derivatives is returned by subroutine

beemat, the displacement, stress and strain at each integrating point can be recovered through the above operation. The FE codes for solving the equilibrium equation and recovering the initial stress at each integrating point are presented in List 4.25.

```
!----------------  loads increment loop and solution of equilibrium equation  ----------------
!-------------------Codes in three-dimensional version creep damage FE program----------------
evpt(1,nip,nels)=0.0;  evpt(2,nip,nels)=0.0
evpt(3,nip,nels)=0.0;  evpt(4,nip,nels)=0.0
read(10,*) loaded_nodes; if(loaded_nodes/=0) then
   read(10,*)(k,loads(nf(:,k)),i=1,loaded_nodes); load_store = loads
    end if
read(10,*) fixed_nodes; if(fixed_nodes /=0) then
   allocate(node(fixed_nodes),sense(fixed_nodes),value(fixed_nodes),      &
         no(fixed_nodes),storkv(fixed_nodes))
   read(10,*) (node(i), sense(i), value(i),i=1,fixed_nodes)
   do i=1,fixed_nodes; no(i)=nf(sense(i),node(i)); end do
   kv(kdiag(no)) = kv(kdiag(no)) + 1.e20  ; storkv = kv(kdiag(no))
  end if
        call sparin (kv,kdiag); bdylds=.0; T=1; t0=0
            do i=1,nels; do j=1,nip; do k=1,7
               tabv(k,j,i)=0
            end do;  end do; end do
       tsigma=0; tevp=0; tdevp=0; do ii=1,2; ij=ii*iy; do iy=1,2; t0=t0+t
iters=0;bdylds=0;evpt=0; loads =.0; if(loaded_nodes/=0) loads = load_store
if(fixed_nodes/=0) loads(no) = storkv * value
loads = loads + bdylds; call spabac(kv,loads,kdiag)
elements_3: do iel = 1 , nels; bload=.0
  num = g_num( : , iel ) ; coord = transpose( g_coord( : , num ))
  g = g_g( : , iel ) ; eld = loads ( g )
  integrating_pts_2 : do i = 1 , nip
       call shape_fun(fun,points,i); call shape_der ( der,points,i)
       jac=matmul(der,coord); det = determinant(jac)
```

```
call invert(jac);deriv = matmul(jac,der)
call beemat(bee,deriv);eps=matmul(bee,eld)
eps=eps-evpt(:,i,iel);  sigma=matmul(dee,eps)
tsigma(:,i,iel)=sigma; abv=tabv(:,i,iel)
```

List 4.25: The FE codes for solving the equilibrium equation and recovering the initial stress at each integrating point in three-dimensional version creep damage FE program

Once the initial stress is obtained by the program, the time increment loop will be executed. The components of stress will be substituted into the creep damage constitutive equation for obtaining the creep damage variables. Here, the Kachanov-Rabotnov creep damage constitutive equation and Euler integration method are used. Then, the creep strain is used in the calculation of body loads at each element and the element body loads are assembled to get the global body loads vector for updating the equilibrium equation. The FE codes for calculating creep damage variables and stress updating are presented in List 4.26.

```
!------------------------     creep damage variables and stress updating    ----------------------------
!--------------------Codes in three-dimensional version creep damage FE program-----------------
call rdmpes (sigma,mpss,ess);  do ix=1, oppo; material(ix)=prop(ix+2,etype(iel))
end do; call EULER_KR (abv,crate,t,t0,sigma,ess,mpss,material)
    tabv(:,i,iel)=abv; tevp(:,i,iel)=evp; evpt(:,i,iel)=evpt(:,i,iel)+evp
    devp=matmul(dee,evp); tdevp(:,i,iel)=devp
    eload=matmul(devp,bee)
    bload=bload+eload*det*weights(i)
  if(tabv(7,i,iel)>=0.99)then
        tabv(7,i,iel)=0.99;  tevp(:,i,iel)=0.0
           km=0.0 ; else; tabv(:,i,iel)=abv
            tevp(:,i,iel)=evp
          evpt(:,i,iel)=evpt(:,i,iel)+evp
        end if
end do integrating_pts_2
bdylds( g ) = bdylds( g )+ bload
  bdylds(0) = 0
```

end do elements_3; end do; end do

List 4.26: The FE codes for calculating creep damage variables and stress updating in three-dimensional version creep damage FE program

The creep damage increases monotonically with the time until the rupture time occurs. The running results are stored in dynamic arrays and they can be output at the end of program by a *write* statement. The output method is the same as the two-dimensional version creep damage FE program. This program is based on the two-dimensional version creep damage FE program and the validation is performed in Chapter 5.

4.9 Development of the Multi-materials Version Creep Damage FE Codes

The components of weldment are complex, thus the multi-materials version FE codes have been developed to cope with this situation. Some new dynamic arrays are used in the development of such FE codes. The element materials information and boundary conditions are stored in a "*dat*" file and they can be read by the main program. The declaration of new variables and new array is shown in Table 4.12 and Table 4.13, respectively. The FE codes of the declaration are presented in List 4.27.

Table 4.12: The declaration of new variables in multi-materials version creep damage FE program

New variable name	Declaration
nprops	number of material property
np_types	number of different property type

Table 4.13: The declaration of new arrays in multi-materials version creep damage FE program

New array name	Declaration
etype	element property type vector
prop	element properties

```
!------------------------------------------ Declaration ----------------------------------------------------
!---------------------Codes in multi-materials version creep damage FE program----------------------
99998 format(1X,I4)
integer:: nels,neq,nn,nr,nip,nodof,nod,nst,ndof,oppo,i,k,iel,ndim,                    &
          loaded_nodes ,nprops,np_types,iy,j,ix,iters,ii,ij,key1=1,                   &
          key2=2, key3=3,key4=4,key5=5,key6=6,key7=7,key8=8,                          &
          key9=9,key10=10,key11=9999
doubleprecision:: ESS, MPSS,T,t0, e, v,det
doubleprecision, dimension (5):: ABV,crate
character(len=15) :: element
doubleprecision, allocatable :: g_coord(:,:),points(:,:),weights(:),kv(:),           &
                   km(:,:),dee(:,:),fun(:),der(:,:),jac(:,:),deriv(:,:),bee(:,:) ,   &
                   coord(:,:),loads(:),eld(:),sigma(:), prop(:,:), eps(:), evp(:),   &
                   devp(:), bload(:),eload(:),evpt(:,:,:),bdylds(:),tabv(:,:,:),     &
                   material(:),storkv(:,:),tsigma(:,:,:),tevp(:,:,:),tdevp(:,:,:),   &
                   gc(:),tgc(:,:,:),teps(:,:,:)
integer, allocatable :: g_num(:,:) ,nf(:,:),g(:),num(:),g_g(:,:),etype(:),no(:)
open (10,file='p5.dat',status='old',   action='read')
open (11,file='p5.res',status='replace',  action='write')
read (10,*) element,nels,nn,nip,nodof,nod,nst,ndim,oppo
ndof=nod*nodof
allocate ( g_coord(ndim,nn),g_num(nod,nels),nf(nodof,nn), g_g(ndof,nels),          &
          g(ndof),points(nip,ndim),weights(nip),coord(nod,ndim),fun(nod),          &
          jac(ndim,ndim),der(ndim,nod),deriv(ndim,nod),bee(nst,ndof),              &
          num(nod),km(ndof,ndof), eld(ndof), sigma(nst),etype(nels),               &
          eps(nst), evp(nst), devp(nst), bload(ndof),eload(ndof),                  &
          evpt(nst,nip,nels), tabv(5,nip,nels), material(oppo),                    &
          tsigma(nst,nip,nels),tevp(nst,nip,nels), dee(nst,nst),                   &
          tdevp(nst,nip,nels),gc(ndim),tgc(ndim,nip,nels),                         &
          teps(nst,nip,nels))
```

```
read(10,*) nprops , np_types; allocate(prop(nprops,np_types))
read(10,*) prop; etype = 1 ; if(np_types>1)
 read(10,*) etype
```

List 4.27: The FE codes of the declaration in multi-materials version creep damage FE program

In this program, *nprops* and *np_types* are two integer variables representing the number of the material property and number of the different property type, respectively. The definition of these integer variables is stored in an input file. *etype* is a dynamic integer array and it represents the element property type vector. *prop* is a dynamic real array and it represents the element properties. The definition of these arrays is stored in an input file; the user can define different element material properties and element types in the input file. Whether a single material model or multi-material model, the program can check the material property type automatically and all the material parameters are provided by an "*input-dat*" file.

Once the element material's property and element type are defined, the program will assemble them into the global stiffness matrix. The FE codes for the multi-material zones problem have been implanted into the 2D (plane stress, plane strain and axisymmetric) and 3D version creep damage FE programs to cope with the creep damage analysis of weldment components.

4.10 Summary

This chapter presents the development of the in-house FE software HITSI for creep damage analysis. Subsequently, the general flow diagram for the development of HITSI and the strategy used with eight development stages are proposed.

HITSI has been developed and the current version includes four main version FE codes (plane stress, plane strain, axisymmetric and three-dimensional) due to the different characteristics of the constitutive matrix. The OOP approach has been considered in developing this FE software; for example the numerical integration method and the creep damage constitutive equation were built in the FE library under this approach; however, the standard FE library (Smith and Griffiths, 2005) was programmed in the Fortran 90 programming language under a structured programming approach. Smith's FE library can be modified and programmed using the OOP approach, and this work

will be reported in Chapter 7. Furthermore, treatment of multi-material zones, failed element removal and stress and creep damage field variables updating has been achieved in the development of HITSI. User guidance of this FE software has been developed and it has been attached in Appendix D.

Originally, the project was implicitly only aiming at the development of a 2D version of software. With successful progress on the 2D version and recognising the practical importance of a more general 3D version, the 3D version software has also been developed. This required some additional work but did not significantly deviate from the overall project.

The author acknowledges that some important achievements and findings in this chapter have been published in Liu et al. (2013b) and Liu et al. (2013c) at various stages in this research.

Chapter 5 Validation of the Finite Element Codes for in-house Software HITSI

5.1 Introduction

This chapter reports the validation of the FE codes for in-house FE software HITSI for creep damage analysis. In order to make the procedure work in a step by step fashion, as well as to be logical and efficient, the strategy in this validation can be described in seven stages corresponding to the development strategy in Chapter 4. The validation stages can be summarized as follows:

1) For the linear elastic FE program, techniques such as input and initialisation, loop elements to find bandwidth and number of equations, element stiffness integration and assembly, equation solution and stress recovery at the central gauss-point have been validated.

2) For the non-linear (single material and time independent) elastic-plastic FE program, techniques such as adding load or displacement increment loop, executing the plastic iteration loop, checking plastic convergence, updating the gauss point stresses and computing the total body loads vector have been validated based on the validation in stage 1.

3) For the in-house FE codes for the plane stress creep damage problem, a two-dimensional uni-axial tension model is used in the validation of the FE codes for adding time increment loop, creep damage constitutive equations, the time integration algorithm, updating the gauss point stress and damage field variables.

4) For the in-house FE codes for the plane strain creep damage problem, a two-dimensional uni-axial tension model is used in the validation of the FE codes for the expanded techniques such as element stiffness integration, element stiffness assembly and the solution of the general equilibrium equation in the plane strain problem based the validation in stage 3.

5) For the in-house FE codes for the axisymmetric creep damage problem, a simple thick wall pipe case is used to validate the FE codes for the expanded techniques such as element stiffness integration, element stiffness assembly and the solution

of the general equilibrium equation in the axisymmetric problem based on the validation in stage 4.

6) For the in-house FE codes for the three-dimensional creep damage problem, a simple three-dimensional uni-axial tension model is used in testing the FE codes for the expanded techniques such as element stiffness integration, element stiffness assembly and the solution of the general equilibrium equation in the three-dimensional problem based on the validation in stage 5.

7) For the in-house FE codes for the multi-materials creep damage problem, a two-dimensional uni-axial tension model is used to validate the multi-material zones version FE codes.

In this chapter, the validation of each FE program is conducted through the comparisons between the FE simulated results from the uni-axial case and the correlative theoretical results. To simulate accurately the rupture time of creep the parameters such as Young's modulus E and Poisson's ratio υ must be well characterised since they strongly influence the stress-strain matrix relationship in FEM, and therefore the initial stress values to creep damage constitutive equation. Kachanov-Rabotnov creep constitutive equation exhibits a stress range dependent description of the creep and damage behaviour, which has to be taken into account for the use of Kachanov-Rabotnov creep constitutive equation in this chapter. The parameter choice in the analytical model should be ensuring the effects of stress and strain states are taken into account in a phenomenological sense. Consequently, in order to make the comparisons between the FE simulated results and theoretical results more intuitionistic the choice of parameter should meet the stress range condition in Kachanov-Rabotnov creep constitutive equation.

This chapter primarily consists of nine sections: 1) Introduction; 2) Validation of the elastic FE program; 3) Validation of the elastic-plastic FE program; 4) Validation of the plane stress version creep damage FE program; 5) Validation of the plane strain version creep damage FE program; 6) Validation of the axisymmetric version creep damage FE program; 7) Validation of the three-dimensional version creep damage FE program; 8) Validation of the FE codes for multi-materials version FE codes; 9) Summary.

5.2 Validation of the Elastic FE Program

5.2.1 Introduction

The validation of elastic FE program is conducted via a two-dimensional tension model which is adopted from Smith's version linear elastic FE program: geotech / software / prog_fe / P50.F90 in (Smith and Griffiths, 2005). Here, the uniform 3-node triangular element numbered in the x-direction is selected for calculating the plane stress of an elastic solid. In this validation, some basic techniques which have been used in developing HITSI can be validated and can be summarized as:

- The technique for reading the mesh, loads and boundary conditions information.
- The technique for assembling element the stiffness matrix into the global system.
- The technique for integrating points to find nodal coordinates and the steering vector.
- The technique for factorising the global stiffness matrix and solving the equation.
- The technique for recovering stresses at the central gauss-point.

In this simulation, the FE model is shown in Figure 5.1.

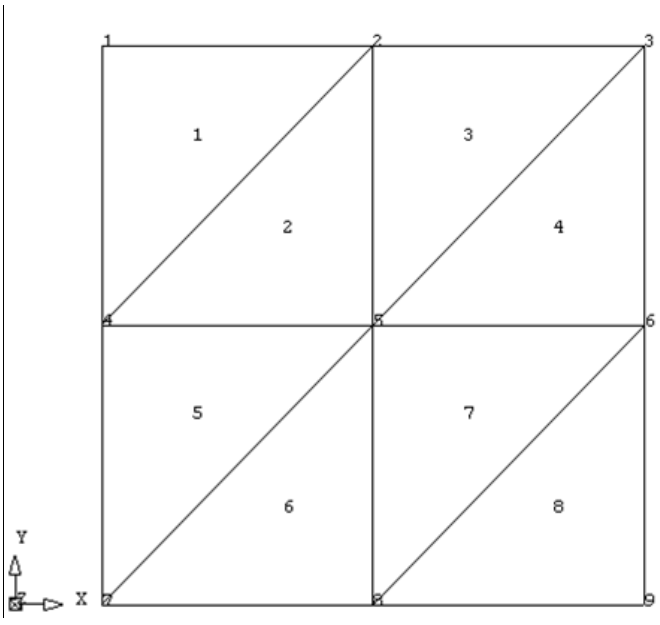


Figure 5.1: The two-dimensional tension model for validating elastic FE codes

The mesh, loads information and boundary conditions for this FE model are given in Table 5.1

Table 5.1: The mesh, loads information and boundary conditions for elastic FE model

Number of elements	Number of elements in x-coordinates direction	Number of nodes	Number of integrating points per element
8	2	9	1
x-coordinates of mesh layout	y-coordinates of mesh layout	E	V
0.5	0.5	2.e6	0.3
Number of restrained nodes			
5			
K (simple counter), nodal freedom matrix (:, K), I=1, number of restrained nodes			
1 (0, 1); 4(0, 1); 7(0, 0); 8(1, 0); 9(1, 0) r			
Number of loaded nodes			
3			
K (simple counter), loads (nodal freedom matrix (:, K)), I=1, number of loaded nodes			
1 (0,-30); 2(0, -60); 3 (0, -30)			

5.2.2 Result and Discussion

The global node number connection information for this FE model are output in the "*res*" file and presented in Table 5.2.

Table 5.2: The global node number connection information for elastic FE model

Element number	**Global node number**
Element 1	1 2 4
Element 2	5 4 2
Element 3	2 3 5
Element 4	6 5 7
Element 5	4 5 7
Element 6	8 7 5
Element 7	5 6 8
Element 8	9 8 6
There are 12 equations and the half-bandwidth is 6	

The program enters the "input and initialisation" stage after the declaration of arrays. Data concerning the mesh and the element properties are presented together with the nodal freedom data. The total number of nodes and equations are provided by subroutine *geometry_3tx*. Then, the elements are looped to generate "global" arrays for

finding the element node numbers, the element nodal coordinates and the element steering vectors.

Once the global stiffness matrix has been assembled, the node connection information, number of equations and the bandwidth of stiffness matrix can be calculated. At this stage, the technique for reading the mesh, loads and boundary conditions information and the technique for assembling element stiffness matrix into global system have been validated. Then, the global coordinate and nodal displacements for this elastic solid FE model have been output and are shown in Table 5.3.

Table 5.3: The global coordinate and nodal displacements for the FE model

Node number	Global coordinates	Nodal displacements
1	(0.0000E+00 0.0000E+00)	(0.0000E+00 -0.6000E-04)
2	(0.5000E+00 0.0000E+00)	(0.9000E-05 -0.6000E-04)
3	(0.1000E+01 0.0000E+00)	(0.1800E-04 -0.6000E-04)
4	(0.0000E+00 -0.50000E+00)	(0.0000E+00 -0.3000E-04)
5	(0.5000E+00 -0.5000E+00)	(0.9000E-05 -0.3000E-04)
6	(0.1000E+01 -0.5000E+00)	(0.1800E-04 -0.3000E-04)
7	(0.0000E+00 -0.1000E+01)	(0.0000E+00 0.0000E+00)
8	(0.5000E+00 -0.1000E+01)	(0.9000E-05 0.0000E+00)
9	(0.1000E+01 -0.1000E+01)	(0.1800E-04 0.0000E+00)

The local coordinates of each integrating point are extracted from the point array, and the derivatives of the shape functions with respect to those coordinates are provided by the library subroutine *shape_der* (Smith and Griffiths, 2005).. The loads and fixed nodes are read by the main program. The global node number connection information and the global coordinate with the nodal displacement for this FE model have been correctly output in Table 5.2 and Table 5.3, respectively. Thus, the technique for integrating points to find nodal coordinates and the steering vector has been validated.

The stresses can be calculated by computing the strain-displacement matrix and the stress-strain matrix at the stage of recovering stresses at integration points. The central point stresses for the elastic solid model are shown in Table 5.4.

Table 5.4: The central gauss point stresses for the FE model

Element number	The central point stress in x-coordinates	The central point stress in y-coordinates	The central point shear stress τ_{xy}
1	-0.2547E-04	-0.1200E+03	-0.5597E-05
2	-0.1261E-04	-0.1200E+03	-0.9795E-05
3	-0.4984E-05	-0.1200E+03	0.1399E-05
4	-0.1871E-06	-0.1200E+03	0.0000E+00
5	-0.5418E-05	-0.1200E+03	-0.2798E-05
6	-0.1871E-06	-0.1200E+03	-0.2798E-05
7	-0.1871E-06	-0.1200E+03	0.4198E-05
8	0.4610E-05	-0.1200E+03	-0.2798E-05

The theoretical stress in y direction is 120 Pa. The stress in x direction and shear stress should be zero. According to Table 5.4, the simulated stress in y direction has been shown to be in good agreement with the theoretical values. The simulated stress in the x direction and the simulated shear stress are negligible. Thus, the technique for factorising the global stiffness matrix, solving the equilibrium equation and the technique for recovering stresses at central gauss-point have been validated.

Through the investigation of the FE program for elastic solid analysis, the techniques such as input and initialisation, loop elements to find bandwidth and number of equation, element stiffness integration and assembly, equation solution and recovering stresses at central gauss-point have been validated and such techniques will be used in the future development of the FE program for the creep damage analysis.

5.3 Validation of the Elastic-plastic FE Program

5.3.1 Introduction

The validation of the elastic-plastic FE program is conducted via an axisymmetric 'un-drained' strain of an elastic-plastic solid case which was introduced in: geotech / software / prog_fe / P66.F90 in (Smith and Griffiths, 2005) and the 8-node quadrilateral element is selected in this validation. The biggest difference between the linear elastic version program and this elastic-plastic program is that the non-linear processes pose much greater analytical problems than do the linear processes. The techniques used in this FE program have been validated and are summarized in following:

- The technique for adding load or displacement increment loop

- The technique for executing the plastic iteration loop
- The technique for checking plastic convergence
- The technique for checking whether yield is violated and updating the gauss point stresses
- The technique for computing the total body loads vector

In this simulation, the FE model is shown in Figure 5.2.

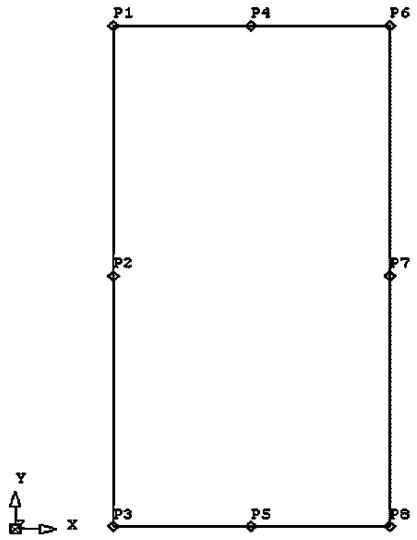


Figure 5.2: The FE model for validating elastic-plastic FE program

The FE mesh, loads information and boundary conditions for this elastic solid FE model are given in Table 5.5.

Table 5.5: The FE mesh, load information and boundary conditions for elastic-plastic solid model

Friction angle (degrees): 30	Cohesion: 0	Dilation angle (degrees): 0 (a); 30 (b)	E: 2.5E4	V: 0.25	Fluid bulk modulus: 1.E6	Consolidating stress: -20
Number of elements: 1	Number of element in x-coordinates: 1	Number of element in y-coordinates: 1		Number of nodes in mesh: 8		Number of integrating points: 4
Number of restrained nodes: 5						
Restrained nodes information: 1 (0, 1); 2(0, 1); 3(0, 0); 5(1, 0); 8(1, 0)						
Width: 1.0			Depth: -2.0			
Number of loaded nodes: 3; I=1, number of loaded nodes: 1 4 6						
Pressure: -5.e-4	Number of load increments: 6	Plastic convergence tolerance: 0.0001		Plastic iteration ceiling: 50		

This case has been investigated by Smith and Griffiths (2005); the maximum iterations to converge is 4 at the 6th load increment and the pore pressure is 0.07934 MPa; the minimum iterations to converge is 2 at the 1st load increment and the pre pressure is 0.02451 MPa. This case is re-investigated here for the validation of the techniques for dealing with the non-linear problem.

5.3.2 Result and Discussion

The global node number connection information for the elastic-plastic FE model is shown in Table 5.6 and the displacement, the stress and the number of iterations to converge are shown in Table 5.7.

Table 5.6: The global node number connection information for elastic-plastic FE model

Node number	**Global coordinates**
1	(0.0000E+00 0.0000E+00)
2	(0.0000E+00 -0.1000E+01)
3	(0.0000E+00 -0.2000E+01)
4	(0.5000E+00 0.0000E+00)
5	(0.5000E+00 -0.2000E+01)
6	(0.1000E+01 0.0000E+00)
7	(0.1000E+01 -0.1000E+01)
8	(0.1000E+01 -0.2000E+01)
Global node numbers	
Element 1	3 2 1 4 6 7 8 5

Table 5.7: The displacement, stress and the number of iterations to converge for elastic-plastic FE model

Load increment	**Displacement**	**Effective stress in x-coordinates**	**Effective stress in x-coordinates**	**Effective shear stress τ_{xy}**	**Deviator stress**	**pore pressure**	**Iterations to converge**
1	-0.5000E-03	-0.1755E+02	-0.2502E+02	-0.1755E+02	0.7475E+01	-0.2451E+01	2
2	-0.1000E-02	-0.1510E+02	-0.3005E+02	-0.1510E+02	0.1495E+02	-0.4902E+01	2
3	-0.1500E-02	-0.1265E+02	-0.3507E+02	-0.1265E+02	0.2243E+02	-0.7353E+01	2
4	-02000E-02	-0.1207E+02	-0.3626E+02	-0.1207E+02	0.2419E+02	-0.7931E+01	4
5	-0.2500E-02	-0.1207E+02	-0.3626E+02	-0.1207E+02	0.2420E+02	-0.7934E+02	4
6	-0.3000E-02	-0.1207E+02	-0.3626E+02	-0.1207E+02	0.2420E+02	-0.7934E+02	4

The local coordinates of each integrating point are extracted from the dynamic array *points*. Subroutine *shape_der* is used to derive the shape function with respect to the coordinates and subroutine *shape_fun* returns the shape function *fun* at the integrating point; this process is similar to the linear elastic FE program. The results in Table 5.7 have been shown to be in good agreement with the results from Smith and Griffiths (2005).

According to Table 5.7, the technique for adding loads increment loop has been validated and the total number of the load increment is 6, which is the same as the maximum load increment set in the input file. Then, the technique for executing the plastic iteration loop and the technique for checking plastic convergence have been shown to be in good agreement with the results from Smith and Griffiths (2005). Lastly, the techniques for checking whether yield is violated and updating the gauss point stress as well as computing the total body loads vector have been validated through comparing the computed results with Smith and Griffiths (2005). The above techniques have been tested and they can be used in the development of the non-linear FE program.

Through the investigation of the FE program for non-linear elastic-plastic solid analysis, the techniques associated with the non-linear problem such as add load or displacement increment loop, execute the plastic iteration loop, check plastic convergence, check whether yield is violated and update the gauss point stresses have been validated and such techniques will be used in the development of the non-linear FE program for creep damage analysis.

5.4 Validation of the in-house FE Codes for Plane Stress Creep Damage Problem

5.4.1 The FE Model and Boundary Conditions

The validation of the in-house FE codes for the plane stress problem is performed in this section and is conducted via the two-dimensional tension model in Figure 5.3. The length of a side is set to 1 metre. The Young's modulus E and Poisson's ratio υ are set to 170 GPa and 0.3, respectively. A uniformly distributed linear load of 40 MPa is applied on the top line of this uni-axial tension model. The Kachanov-Rabotnov creep damage constitutive equation is used. Comparisons are made between the simulated results predicted by the plane stress version creep damage FE program and the theoretical values.

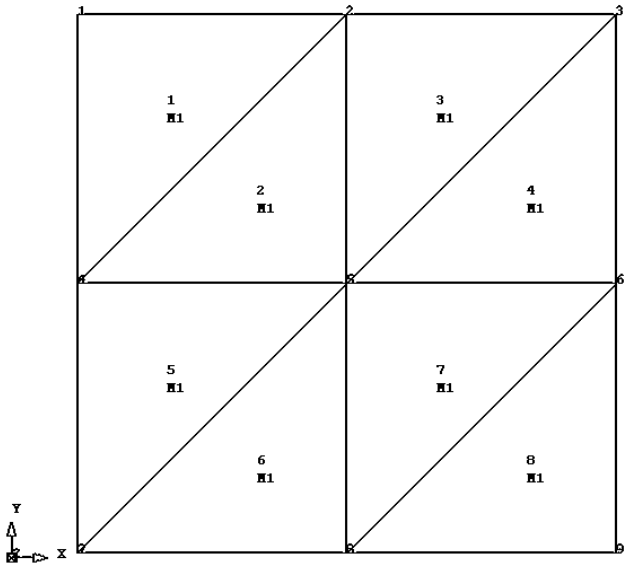


Figure 5.3: 2D plane stress tension FE model

This is a two-dimensional plane stress uni-axial tension case and the boundary conditions should preserve the uni-axial tension's characteristics. The boundary conditions and loads information are listed in Table 5.8.

Table 5.8: The boundary conditions for 2D plane stress tension mode

Node number	Constraint in x direction	Constraint in y direction	Load in x direction	Load in y direction
Node No.1	shut	open	0	10
Node No.2	open	open	0	20
Node No.3	open	open	0	10
Node No.4	shut	open	0	0
Node No.5	open	open	0	0
Node No.6	open	open	0	0
Node No.7	shut	shut	0	0
Node No.8	open	shut	0	0
Node No.9	open	shut	0	0

5.4.2 Results and Discussion

The simulated results will be compared with the theoretical values to validate the FE codes. The stress in the x direction should be zero. The stress values should remain the same throughout the creep test up to failure. The theoretical stress in the y direction can be calculated by:

$$\sigma_y = \frac{P}{A} = \frac{40}{1.0} = 40 \text{ MPa} \tag{5.1}$$

Thus, the theoretical stress in the y direction is 40 MPa and by substituting the theoretical stress value into the Kachanov-Rabotnov creep damage constitutive equation with the Euler integration method, the theoretical rupture time and creep damage can be obtained by the proven subroutine developed by the author's colleague Feng Tan. The theoretical rupture time and creep damage are shown in Table 5.9.

Table 5.9: The theoretical rupture time and creep damage

Rupture time	Creep damage
23773	0.99

The stress distributions in the y direction and the x direction obtained from FE software, with the stress updating invoked due to creep deformation, are shown in Figure 5.4 and Figure 5.5 separately. The initial elastic stress for each element, without stress updating, and the stress involving creep deformation with stress updating are shown in Table 5.10 and Table 5.11, respectively. Both confirmed the uniform distribution of stresses, and the values of stress in the y direction obtained from FE software are correct, and the stress in the x direction is negligible.

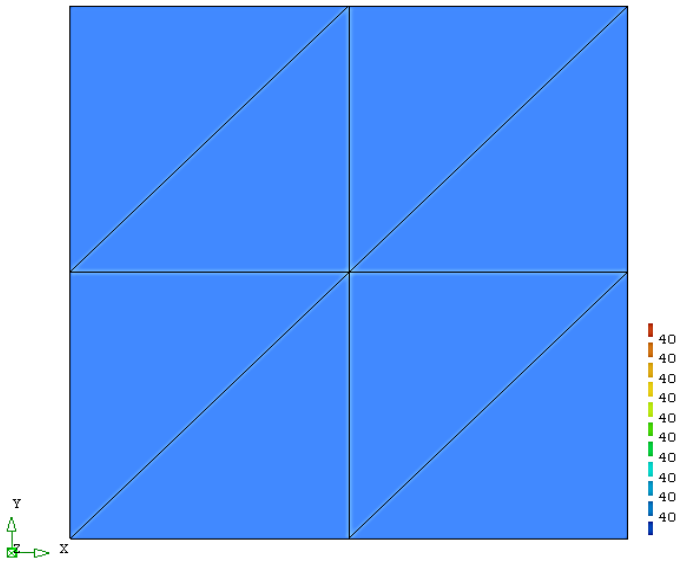


Figure 5.4: The simulated stress distribution in the y direction with stress updating at rupture time

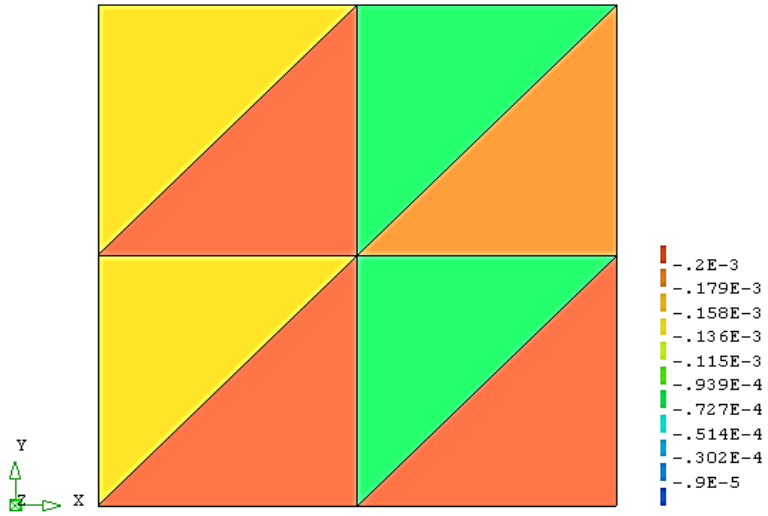


Figure 5.5: The simulated stress distribution in the x direction with stress updating at rupture time

Table 5.10: The initial elastic stress obtained from plane stress version FE program without stress updating for each element

Element number	Stress in x-direction	Stress in y-direction
Element No.1	-7.1054E-06	0.4000E+02
Element No.2	0.0000E-000	0.4000E+02
Element No.3	1.7764E-06	0.4000E+02
Element No.4	- 8.8818E-06	0.4000E+02
Element No.5	-1.7764E-06	0.4000E+02
Element No.6	1.7764E-06	0.4000E+02
Element No.7	-1.7764E-06	0.4000E+02
Element No.8	-1.7764E-06	0.4000E+02

Table 5.11: The stress obtained from plane stress version FE program with stress updating for each element

Element number	Stress in x-direction	Stress in y-direction
Element No.1	-1.3871E-04	0.4000E+02
Element No.2	-2.8081E-04	0.4000E+02
Element No.3	-8.6551E-05	0.4000E+02
Element No.4	-1.9584E-04	0.4000E+02
Element No.5	-1.4892E-04	0.4000E+02
Element No.6	-2.7864E-04	0.4000E+02
Element No.7	-7.7958E-05	0.4000E+02
Element No.8	-2.0437E-04	0.4000E+02

Using the Kachanov-Rabotnov creep damage constitutive equation and a one hour time step with the Euler integration method, the rupture time and creep damage values from the FE software at rupture time can be obtained and they are shown in Table 5.12.

Table 5.12: Rupture time and creep damage obtained from plane stress version FE program at failure time

Element number	Rupture time	Creep damage
Element No.1	23774	0.99E+00
Element No.2	23774	0.99E+00
Element No.3	23774	0.99E+00
Element No.4	23774	0.99E+00
Element No.5	23774	0.99E+00
Element No.6	23774	0.99E+00
Element No.7	23774	0.99E+00
Element No.8	23774	0.99E+00

Table 5.13: The relative error between theoretical rupture time and simulated rupture time from plane stress version FE program

$$\text{Rupture time relative error} = \left|\frac{23773 - 23774}{23773}\right| = 0.000042$$

A comparison of the results shown in Table 5.9 and Table 5.12 and an examination of the percentage errors shown in Table 5.13 clearly show the results obtained from the plane stress version FE program agree with the expected theoretical values and the relative error is negligible.

5.5 Validation of the in-house FE Codes for Plane Strain Creep Damage Problem

5.5.1 The FE Model and Boundary Conditions

The validation of the in-house codes for the plane strain problem is performed in this section and is conducted via the two-dimensional tension model in Figure 5.6. The width of this model is set to 4 metres. The Young's modulus E and Poisson's ratio υ are set to 1,000 GPa and 0.29, respectively. A uniformly linear distributed load of 60 MPa is applied on the top line of this uni-axial tension model. The Kachanov-Rabotnov creep damage constitutive equation is used. Comparisons are made between the simulated

results predicted by the plane strain version creep damage FE program and the theoretical values.

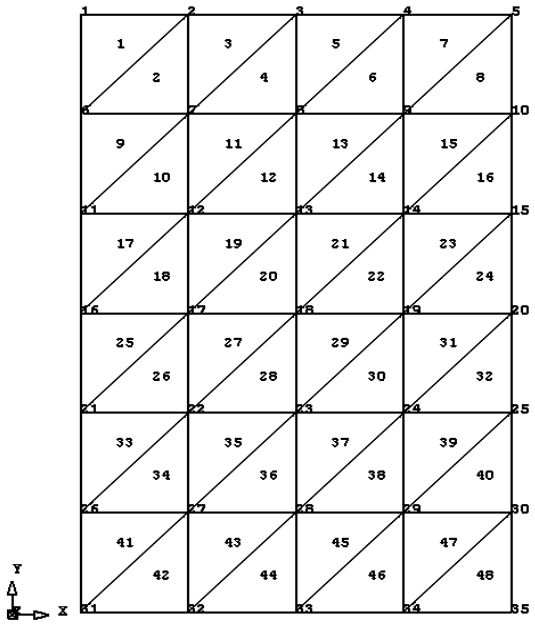


Figure 5.6: Plane strain tension FE model

This is a two-dimensional plane strain tension case and the boundary conditions should preserve the uni-axial tension's characteristics. The boundary conditions and loads information have been listed in Table 5.14.

Table 5.14: The boundary conditions for 2D plane strain tension FE mode

Node number	Constraint in x direction	Constraint in y direction	Load in x direction	Load in y direction
Node No.1	shut	open	0	30
Node No.2	open	open	0	60
Node No.3	open	open	0	60
Node No.4	open	open	0	60
Node No.5	open	open	0	30
Node No.6	shut	open	0	0
Node No.11	shut	open	0	0
Node No.16	shut	open	0	0
Node No.21	shut	open	0	0
Node No.26	shut	open	0	0
Node No.31	shut	shut	0	0
Node No.32	open	shut	0	0
Node No.33	open	shut	0	0
Node No.34	open	shut	0	0
Node No.35	open	shut	0	0

5.5.2 Results and Discussion

The simulated results will be compared with the theoretical values to validate the in-house FE codes. The theoretical stress in the y direction can be shown by:

$$\sigma_y = \frac{P}{A} = \frac{240}{4.0} = 60 \text{ MPa} \tag{5.2}$$

The theoretical stress in the z direction can be shown by:

$$\sigma_z = E * \epsilon_z = E * \upsilon * \epsilon_y = E * \upsilon * \frac{\sigma_y}{E} = 0.29 * 60 = 17.4 \text{ MPa} \tag{5.3}$$

By substituting the theoretical stress value into the Kachanov-Rabotnov creep damage constitutive equation with the Euler integration method, the theoretical rupture time and creep damage may be obtained by the proven subroutine developed by the author's colleague Feng Tan. The theoretical rupture time and creep damage are shown in Table 5.15.

Table 5.15: The theoretical rupture time and creep damage for plane strain case

Rupture time	Creep damage
7004	0.99

The stress in the y and z directions obtained from the plane strain version creep damage FE program, with stress updating invoked due to creep deformation, are shown in Figure 5.7 and Figure 5.8. The displacements in the y and x directions are shown in Figure 5.9 and Figure 5.10, respectively.

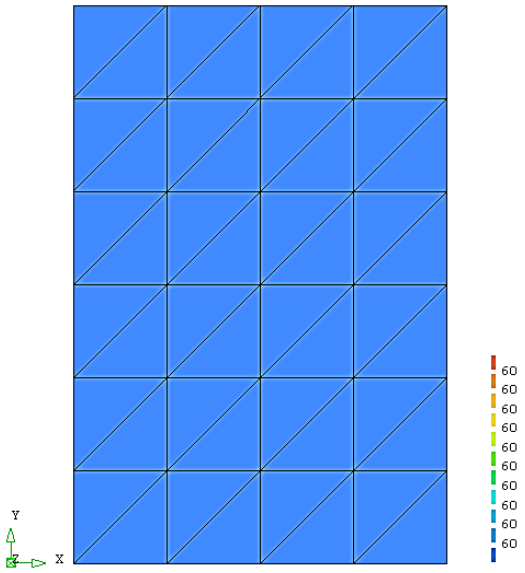


Figure 5.7: Stress distribution in y direction

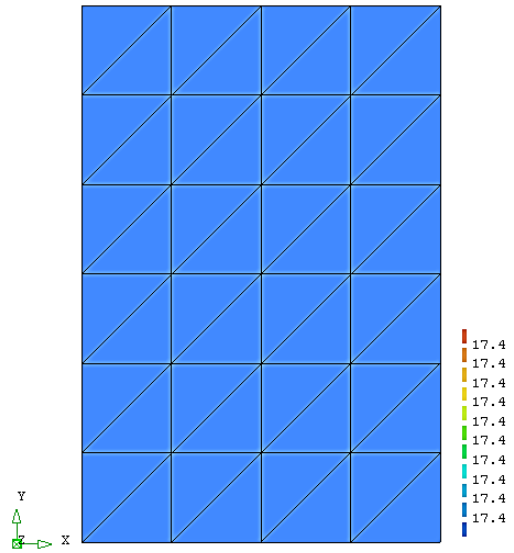


Figure 5.8: Stress distribution in z direction

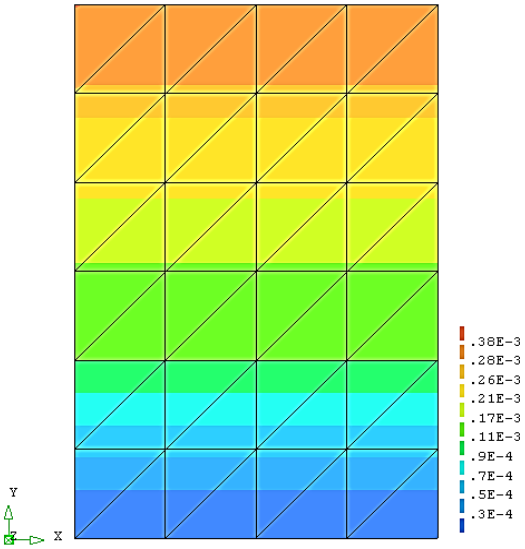


Figure 5.9: Displacement distribution in y axis

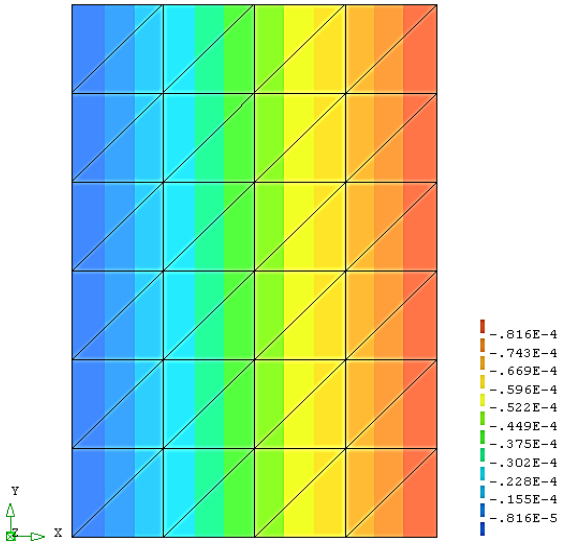


Figure 5.10: Displacement distribution in x axis

The damage distribution obtained from plane strain version creep damage FE program at failure is shown in Figure. 5.11.

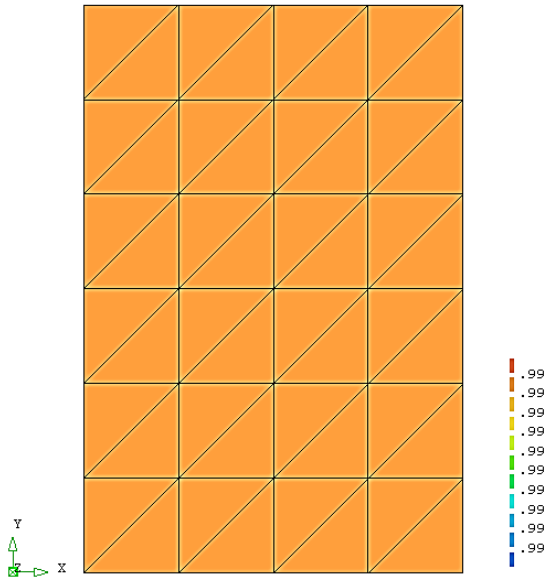


Figure 5.11: Damage distribution on 7039h

Figures 5.7 and 5.8 show that the results obtained from the plane strain version creep damage FE program agree with the expected theoretical values. The displacement is distributed reasonably in Figures 5.9 and 5.10. Table 5.15 and Figure 5.11 show that the rupture time and damage obtained from the FE software are in good agreement with the theoretical values obtained from the subroutine directly.

5.6 Validation of the in-house FE Codes for Axisymmetric Creep Damage Problem

5.6.1 The FE Model and Boundary Conditions

The validation of the in-house FE codes for the axisymmetric problem is performed in this section and is conducted via a two-dimensional uni-axial tension model in Figure 5.12. The Young's modulus E and Poisson's ratio υ are set to 100 GPa and 0.3, respectively. The thickness of the pipe is set to 60 mm. A uniformly distributed tensile force of 50 MPa is applied on the bottom line of this uni-axial tension model. The Kachanov-Rabotnov creep damage constitutive equation subroutine, developed by the author's colleague Feng Tan, has been used. Comparisons are made between the simulated results predicted by the axisymmetric version creep damage FE program and the theoretical values.

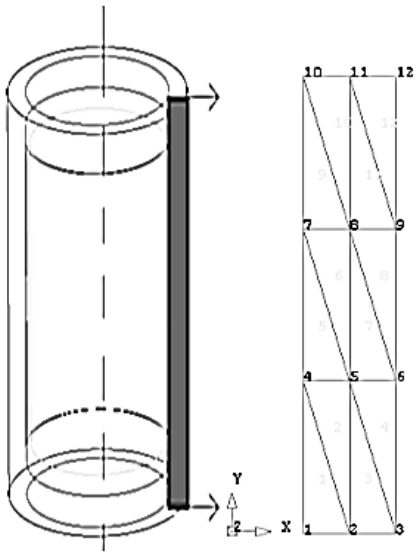


Figure 5.12: Axisymmetric FE model

This is a two-dimensional axisymmetric tension case and the boundary conditions should preserve the uni-axial tension's characteristics. The nodal loads information is calculated by the nodal force calculator developed by the author's colleague Feng Tan. The boundary conditions and loads information are listed in Table 5.16.

Table 5.16: The boundary conditions for axisymmetric tension FE mode

Node number	Constraint in x direction	Constraint in y direction	Load in radial direction	Load in axial direction
Node No.1	open	open	0	9.3750000E+04
Node No.2	open	open	0	2.1750000E+05
Node No.3	open	open	0	1.2375000E+05
Node No.4	open	open	0	0
Node No.5	open	open	0	0
Node No.6	open	open	0	0
Node No.7	open	open	0	0
Node No.8	open	open	0	0
Node No.9	open	open	0	0
Node No.10	open	shut	0	0
Node No.11	open	shut	0	0
Node No.12	open	shut	0	0

5.6.2 Results and Discussion

By substituting the theoretical stress value into the Kachanov-Rabotnov creep damage constitutive equation, the theoretical rupture time and creep damage can be obtained and the theoretical results are shown in Table 5.17.

Table 5.17: The theoretical rupture time and creep damage for axisymmetric case

Rupture time	Creep damage
10692	0.99

The simulated stress from the axisymmetric version creep damage FE program is shown in Figure 5.13 and the displacement in axial and radial directions obtained from FE software, with the stress updating invoked due to creep deformation, are shown in Figure 5.14 and Figure 5.15, respectively.

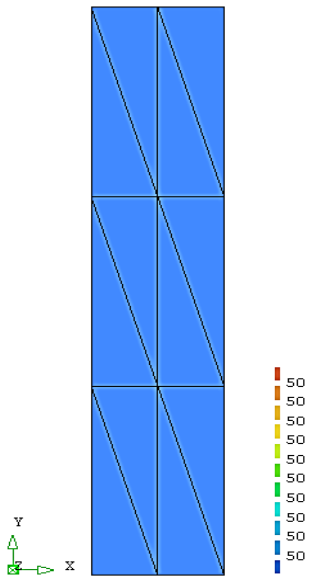


Figure 5.13: Stress distribution in axial direction

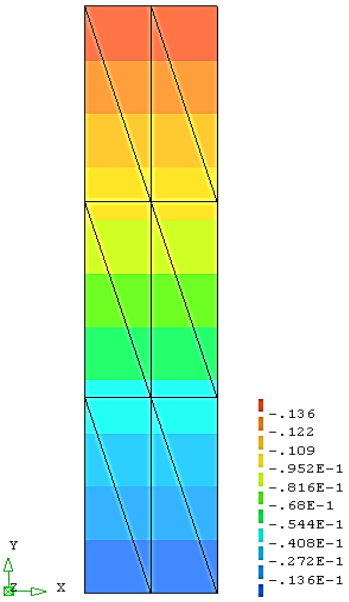


Figure 5.14: Displacement distribution in axial direction

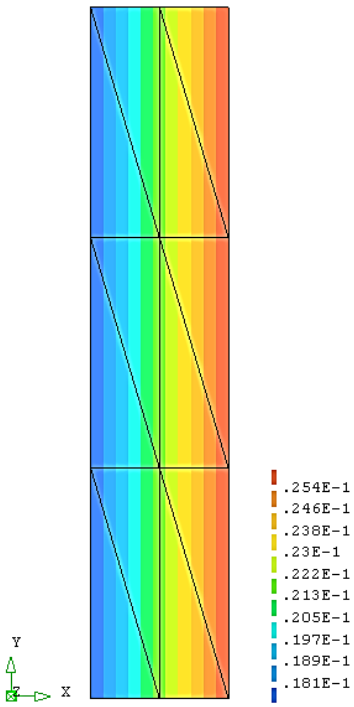


Figure 5.15: Displacement distribution in radial direction

The damage distribution obtained from the axisymmetric version creep damage FE program at failure is shown in Figure 5.16.

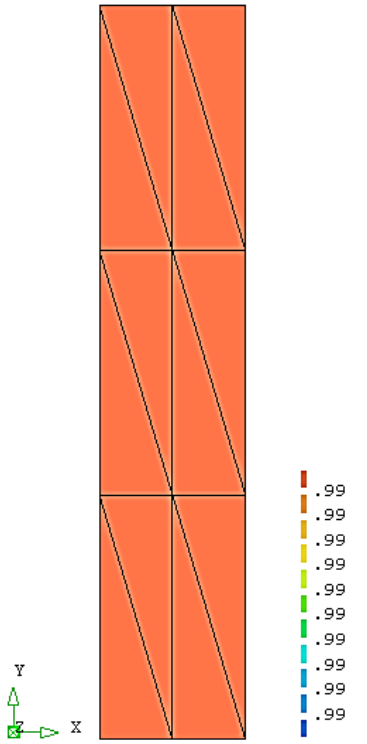


Figure 5.16: Damage distribution on 10693h

The stress has been uniformly distributed in Figure 5.13 and agrees with the theoretical values. Rupture time and damage obtained from the axisymmetric version creep damage FE program have been shown to have a good agreement with the theoretical values.

5.7 Validation of the in-house FE Codes for Three-dimensional Creep Damage Problem

5.7.1 The FE Model and Boundary Conditions

The validation of the in-house FE codes for the three-dimensional problem is conducted via a three-dimensional uni-axial tension model in Figure 5.17. The length of a side is set to 1 metre and a uniformly distributed displacement of 0.0005 m was applied on the top surface of this uni-axial tension model. The Young's modulus E and Poisson's ratio υ are set to 170 GPa and 0.3, respectively. The Kachanov-Rabotnov creep damage constitutive equation subroutine with Euler integration method has been used. Comparisons are made between the simulated results predicted by the three-dimensional version creep damage FE program and the theoretical values.

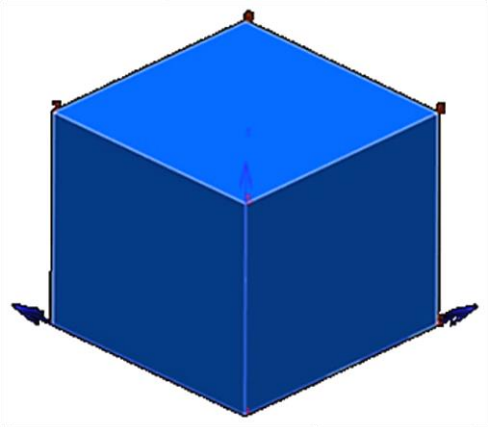


Figure 5.17: The three-dimensional uni-axial tension model

This is a three-dimensional uni-axial tension case and the boundary conditions should keep the uni-axial tension’s characteristics. The boundary conditions and loads information are listed in Table 5.18.

Table 5.18: The boundary conditions for three-dimensional uni-axial tension model

Node number	Constraint in x direction	Constraint in y direction	Constraint in z direction	Displacement in z direction
Node No.1	shut	shut	open	0.0005
Node No.2	open	shut	open	0.0005
Node No.3	open	shut	open	0.0005
Node No.4	shut	shut	open	0
Node No.5	open	shut	open	0
Node No.6	shut	shut	shut	0
Node No.7	open	shut	shut	0
Node No.8	open	shut	shut	0
Node No.9	shut	open	open	0.0005
Node No.10	open	open	open	0.0005
Node No.11	shut	open	shut	0
Node No.12	open	open	shut	0
Node No.13	shut	open	open	0.0005
Node No.14	open	open	open	0.0005
Node No.15	open	open	open	0.0005
Node No.16	shut	open	open	0
Node No.17	open	open	open	0
Node No.18	open	shut	open	0
Node No.19	open	open	shut	0
Node No.20	open	open	shut	0

5.7.2 Results and Discussion

The uniformly distributed displacement of 0.0005 m was applied on the top surface of this uni-axial tension model. Thus the theoretical stress can be calculated:

$$\sigma = E * \mathcal{E} = E * \frac{\Delta l}{l} = 170000 \text{ MPa} * \frac{0.0005}{1.0} = 85 \text{ MPa} \qquad (5.4)$$

The theoretical stress in the z direction is 85 MPa. The stress in the x and y directions should be zero and these stress values should remain the same throughout the creep test up to failure. The stress obtained from the three-dimensional version creep damage FE program, with the stress updating, is shown in Table 5.19 and a one hour time step is selected with the Euler integration method.

Table 5.19 shows that the results obtained from the three-dimensional version creep damage FE program agree with the expected theoretical values. The stress involving

creep deformation and stress updating confirmed the uniform distribution of the stresses, that the values of stress in the z direction obtained from FE software are correct, and that the stress in the x and y directions is negligible.

Table 5.19: The stress obtained from three-dimensional version creep damage FE program with stress updating

Integration point	σ_x	σ_y	σ_z
No. 1	8.5265E-014	-2.8422E-014	8.5E+01
No. 2	8.5265E-014	-2.8422E-014	8.5E+01
No. 3	1.2789E-013	6.3948E-014	8.5E+01
No. 4	7.8160E-014	0.0000E-014	8.5E+01
No. 5	2.1316E-014	-2.8422E-014	8.5E+01
No. 6	4.2633E-014	4.2633E-014	8.5E+01
No. 7	8.5265E-014	3.5527E-014	8.5E+01
No. 8	-7.1054E-015	-7.1054E-015	8.5E+01

Table 5.20: The theoretical values and FE results from three-dimensional version creep damage FE program

The results	Theoretical values	FE results
Rupture time	1602	1603
Damage	0.99	0.99

The lifetime and creep strain at failure, and other field variables can be obtained for the simple tensile case that has been illustrated in above. The theoretical values are obtained by direct integration of the uni-axial version of constitutive equation for a given stress. FE results are produced by the three-dimensional version creep damage FE program. Table 5.20 shows that the FE results are in good agreement with the theoretical values obtained from the subroutine directly.

5.8 Validation of the in-house FE Codes for Multi-materials Version Program

5.8.1 The FE Model and Boundary Conditions

The validation of the in-house FE codes for the multi-materials version is conducted via a two-dimensional tension model in Figure 5.18. In this program, the number of material properties *nprops* is set to 1 and 2 separately. The number of different property types *np_types* is set 2 (Young's modulus E and Poisson's ratio υ). The length of a side

is set to 1 metre. The Young's modulus E and Poisson's ratio υ are set to 1,000 MPa and 0.3 respectively. A uniformly distributed linear load of 40 KN/m was applied to the top line of this uni-axial tension model. Table 5.22 shows the material property of each element when *nprops* is set to 1 and when *nprops* is set to 2 respectively.

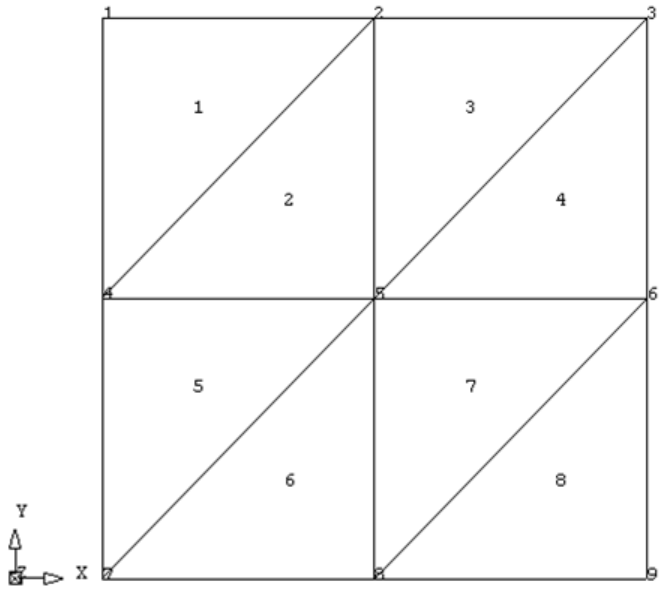


Figure 5.18: 2D tension model

This is a two-dimensional plane strain tension case and the boundary conditions should preserve the uni-axial tension's characteristics. The boundary conditions and loads information are listed in Table 5.21.

Table 5.21: The boundary conditions for 2D tension FE mode

Node number	**Constraint in x direction**	**Constraint in y direction**	**Load in x direction**	**Load in y direction**
Node No.1	shut	open	0 KN	10 KN
Node No.2	open	open	0 KN	20 KN
Node No.3	open	open	0 KN	10 KN
Node No.4	shut	open	0 KN	0 KN
Node No.5	open	open	0 KN	0 KN
Node No.6	open	open	0 KN	0 KN
Node No.7	shut	shut	0 KN	0 KN
Node No.8	open	shut	0 KN	0 KN
Node No.9	open	shut	0 KN	0 KN

In order to test the multi-materials version program, the material properties of each element have been divided into *nprops* = 1 and *nprops* =2, respectively. Comparisons are made between the simulated results predicted when *nprops* = 1 and *nprops* =2. The material properties of each element when *nprops* = 1 and *nprops* = 2 have been shown in Table 5.22.

Table 5.22: The material properties of each element when *nprops* = 1 and *nprops* =2

nprops	**Materials group 1 (E and υ)**	**Materials group 2 (E and υ)**
nprops=1	Element No.1, 2, 3, 4, 5, 6, 7 and 8	No element
nprops=2	Element No.1, 2, 3 and 4	Element No 5, 6, 7 and 8

5.8.2 Results and Discussion

Comparisons are made between the simulated stress distribution in the y direction at rupture time predicted by *nprops* = 1 and *nprops* =2. The stress distribution in the y direction at rupture time when the *nprops* = 1 is shown in Figure 5.19 and the stress distribution in the y direction at rupture time when the *nprops* = 2 is shown in Figure 5.20.

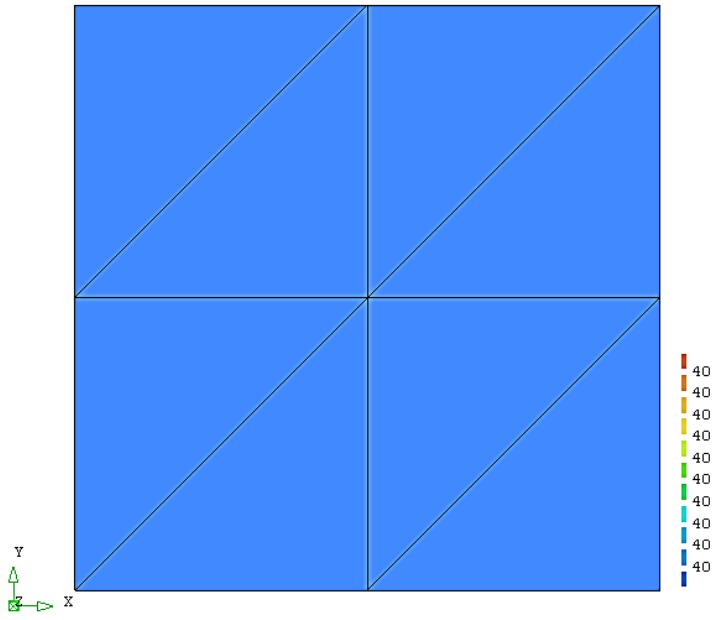


Figure 5.19: The stress distribution in y direction at rupture time when *nprops* = 1

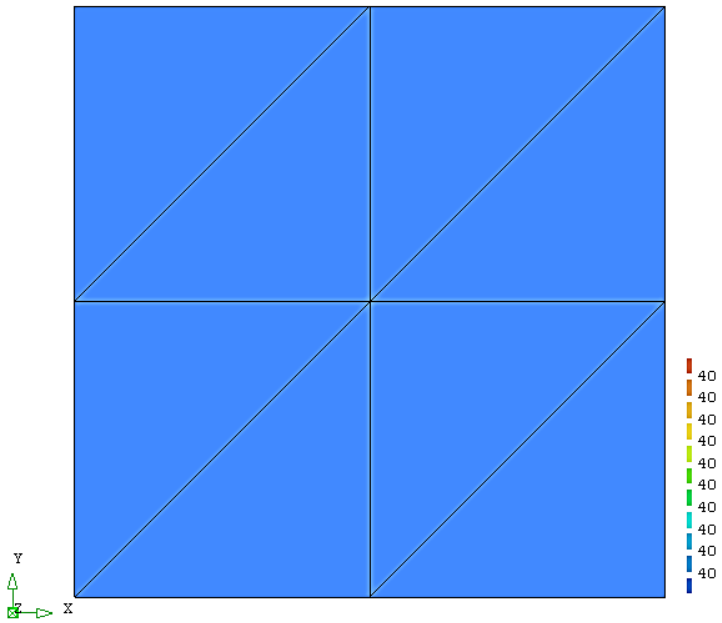


Figure 5.20: The stress distribution in y direction at rupture time when *nprops* = 2

When *nprops* = 2, there are two kinds of material properties. In the program, we can assume two kinds of material properties in model; however, the values of material properties in the "input-dat" file are the same from *nprops* = 1 to *nprops* = 2. Thus, the stresses distribution in the y direction between *nprops* = 1 and *nprops* = 2 should be same. Figure 5.19 and Figure 5.20 show a good agreement with this deduction.

A reliable prediction of the creep damage behaviour of materials in dependence on the stress regime and the temperature is a very complex challenge. Especially multi-material zones in responsible structures need to be characterized very exactly. In FEM for the analysis of creep damage in multi-material zones, different body loads in the material regions are produced with the growth of creep deformation and this phenomenon causes the stresses to redistribute. Subsequently, the non-linear behaviour is appeared due to stress redistribution and there is no direct method to solve the non-linear equation in mathematics. As a result the non-linear creep behaviour is difficult to depict through the investigation of analytical solutions. Here the values of above two material properties are defined as same in order to test the FE codes through the comparison of the stresses distribution by different settings in program. To validate the multi-materials version FE program for creep damage analysis, a real multi-materials Cr-Mo-V steam pipe weldment case in chapter 6 will be investigated.

5.9 Summary

This chapter presents the validation of the FE codes for the in-house FE software HITSI for creep damage analysis. A step by step validation in accord with the development strategy is proposed. The FE simulated results from HITSI (uni-axial case) are compared with the theoretical results to demonstrate the validity of the FE program. All results have been shown to be in good agreement with the expected or theoretical values.

The author acknowledges that some important achievements and findings in this chapter have been published in Liu et al. (2013d) and Liu et al. (2013e) at various stages in this research.

Chapter 6 Benchmark Test of HITSI via the Numerical Investigation of Creep Damage Behaviour of a Cr-Mo-V Steam Pipe Weldment Case

6.1 Introduction

The computational FEM based CDM approach for the in-house FE software HITSI has been developed and applied to the analysis of deformation and creep damage in welds. This chapter presents the benchmark test of HITSI via the numerical investigation of creep damage behaviour of a Cr-Mo-V steam pipe weldment case. It should be noted that some benchmark tests of FE in-house software such those of Hall and Hayhurst (1991), Wong (1999) and (Becker et al., 2002) have previously been presented; here, benchmark test of HITSI are performed based on the studies of Hall and Hayhurst (1991), Wong (1999) and (Becker et al., 2002). Furthermore, Ling et al. (2000) reported the fourth order Runge-Kutta integration scheme used in Hall and Hayhurst (1991) might be incorrect. Through the study and comparison of Ling et al. (2000) and Hall and Hayhurst (1991), the author concludes that the expression of Runge-Kutta integration equations between Ling et al. (2000) and Hall and Hayhurst (1991) is different and the use of Runge-Kutta integration method in Hall and Hayhurst (1991), which has published by the Royal Society, is correct and this argument is not affecting the benchmark test of HITSI..

This chapter primarily consists of two parts: firstly, the damage evolution of a 2.25Cr 1Mo: 0.5Cr 0.5Mo 0.25V thick steam pipe weldment from a constant pressure (455 bar) vessel test (Coleman et al., 1985), at a constant temperature of 565°C, is modelled by HITSI and the benchmark test against the known results is presented; secondly, the efficiency and accuracy of the integration schemes (Euler and Runge-Kutta) and the normalized Kachanov-Rabotnov creep damage constitutive equation (Hayhurst et al., 1984) are investigated and commented upon.

The specific knowledge relevant to this chapter is presented below in detail and includes:

1) The verification of HITSI via the numerical investigation of creep damage behaviour of a steam pipe weldment case; the computational results, such as damage distributions, stress and failure times, are compared with the known results from laboratory tests (Coleman et al., 1985) and another FE software program, Damage XX (Hall and Hayhurst, 1991), respectively. Finally, the in-house software HITSI is shown to predict reasonably well the failure history of the pressure vessel weldment.

2) The investigation of the efficiency and accuracy of the numerical integration schemes (Euler and Runge-Kutta) through the analysis of creep damage behaviour in this weldment case; the result reveals that the total computation time can be reduced by the Runge-Kutta method in a problem with a large set of system equations.

3) The investigation of the normalized Kachanov-Rabotnov creep damage constitutive equation (Hayhurst et al., 1984) through the analysis of creep damage behaviour in this weldment case; the result reveals that the computing efficiency can be increased through the use of a normalized Kachanov-Rabotnov creep damage constitutive equation.

6.2 Description of the Cr-Mo-V Steam Pipe Weldment Case

6.2.1 Description of the Experiment

The creep strain data and the whole rupture history of a 2.25Cr 1Mo: 0.5Cr 0.5Mo 0.25V thick steam pipe weldment from a constant pressure (455 bar) vessel test at a constant temperature of 565°C were compiled by Coleman et al. (1985) and some details have been described by Hall and Hayhurst (1991). The micrograph in Figure 6.1 shows a section through a 2.25Cr 1Mo: 0.5Cr 0.5Mo 0.25V multi-materials weldment, which is identical to the welds used in the thick steam pipe tests (Coleman et al., 1985). According to Coleman et al. (1985), the wall thickness of this steam pipe section is 60 mm, the external radius is 175 mm and the external to internal diameter ratio is approximately 1.52. The end caps of the vessel were forged and the seamless pipe sections of the parent metal were hot drawn.

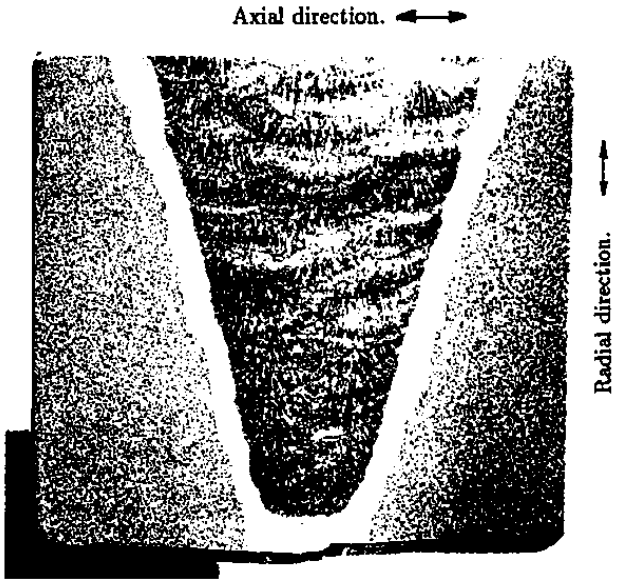


Figure 6.1: Micrograph showing a section through a 2.25Cr 1Mo: 0.5Cr 0.5Mo 0.25V multi-materials weld, identical to the welds used in the thick steam pipe tests of Coleman et al. (1985)

A summary of rupture evolution of the 2.25Cr1Mo:0.5Cr0.5Mo0.25V steam pipe weldment from the pressure vessel test by Coleman et al. (1985) is shown in Table 6.1. In Section 6.6, the computational results from HITSI will be compared with the experimental results to allow verification of HITSI.

Table 6.1: A summary of rupture evolution of the 2.25Cr1Mo:0.5Cr0.5Mo0.25V steam pipe weldment from a pressure vessel test by Coleman et al. (1985)

Time/h	**t/t_f**	**Observation**
20000	43%	Creep cracks appear on the pressure vessel, as transverse cracks in the coarse columnar regions of the weld metal and HAZ
35000	76%	More clearly defined transverse cracks in capping weld bead and the depth are less than 5mm
35000	>76%	Circumferential cracks appear in the weld metal coarse columnar regions close to the fusion boundaries
42000	91%	Obvious circumferential cracks can be observed; the transverse cracks increased significantly with a depth of 20mm and extend through the weld metal, across the HAZ into the parent metal
46000	100%	Numerous transverse and circumferential cracks lead to steam leakage in a bulged region of the pressure vessel; the pressure vessel has reached its rupture life

6.2.2 Description of the FE Model in FE Software Damage XX

The deformation and failure processes through macroscopic cracking in the pressure vessel test (Coleman et al., 1985) were modelled by Hall and Hayhurst (1991) through the use of FE software Damage XX. A three materials weld FE model is used and the discrete regions of the FE model are assigned the creep properties of the parent metal, HAZ and the weld metal. The Kachanov-Rabotnov creep damage constitutive equation was embedded into Damage XX for this investigation; the axisymmetric FE model used to represent the thick-steam pipe weldment is shown in Figure 6.2.

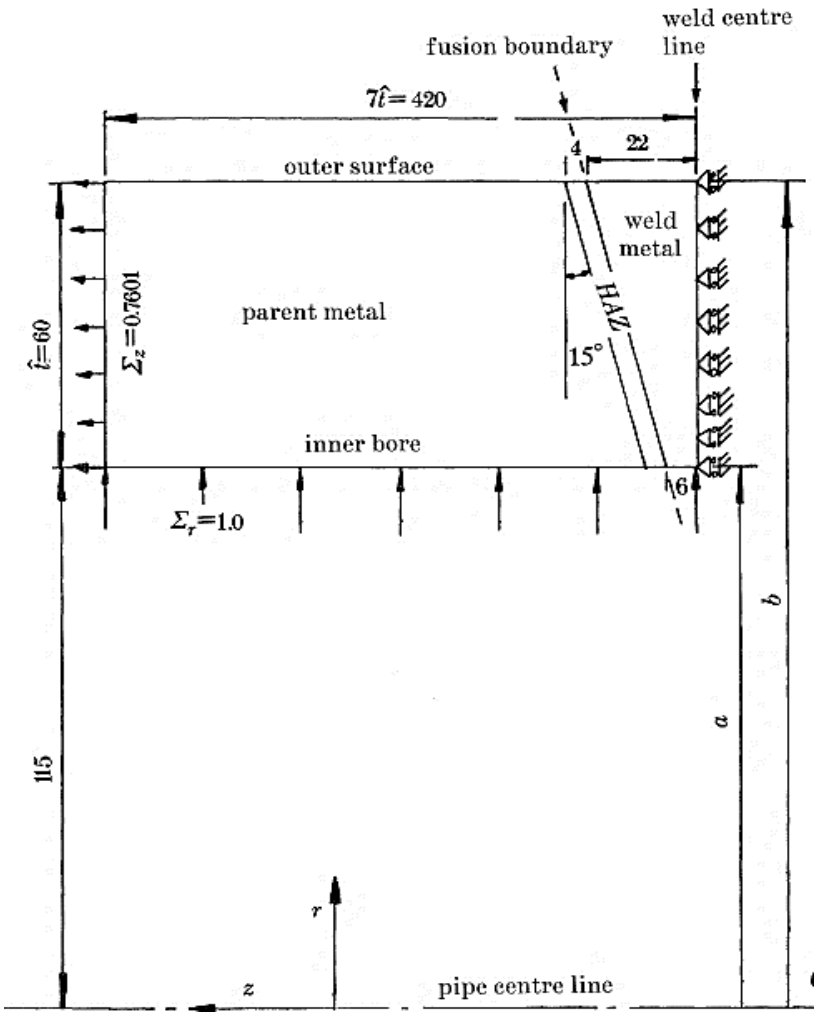


Figure 6.2: The diagram showing the axisymmetric FE model that be used to represent the thick-steam pipe weld laboratory test (Hall and Hayhurst, 1991)

A brief summary of rupture evolution of the 2.25Cr1Mo:0.5Cr0.5Mo0.25V steam pipe weldment from the FE model by Damage XX (Hall and Hayhurst, 1991) is shown in Table 6.2. In Section 6.6, the computational results from HITSI will be compared with the results from Damage XX to allow verification of HITSI.

Table 6.2: A brief summary of rupture evolution of the 2.25Cr1Mo:0.5Cr0.5Mo0.25V steam pipe weldment from the FE model by Damage XX (Hall and Hayhurst, 1991**)**

Time/h	**t/t_f**	**Observation**
50	0.12%	The maximum elastic stress elements are concentrating on the inner bore and the initial damage rate is highest at this position.
17682	45.2%	The damage distribution in the weld is more uniform and the maximum damaged region occurs along the fusion boundary between the weld metal and the HAZ
24841	63.5%	The most damaged zone on the fusion boundary has become wider
30199	77.2%	The maximum damaged zone on the fusion boundary close to surface is becoming more and more intense
31608	80.8%	The centroid of the damaged zone has moved slightly off the fusion boundary into the weld metal
34034	87.0%	The intense damage on the fusion boundary spreads both inward and outwards
38728	99%	The damaged zone on the fusion boundary now has higher damage levels
39119	99.9%	The coalescence of the most damaged zones into two main localized damaged regions

6.2.3 The Relative Error between Experimental Results and Simulated Results by Damage XX

The actual failure time of the pressure vessel test by Coleman et al. (1985) is 46000 hours and the simulated failure time by Hall and Hayhurst (1991) through the use of Damage XX is 39119 hours. Thus, the relative error between the simulated failure time by Damage XX and the failure time of the pressure vessel test can be summarized in Table 6.3.

Table 6.3: The relative error between the simulated failure time by Damage XX and the failure time of the pressure vessel test

Rupture time from Damage XX	**Rupture time from laboratory test**
39119 hours	46000 hours
Rupture time relative error $= \left\lvert \frac{39119 - 46000}{46000} \right\rvert = 0.15$	

The simulated results from the FE weldment model by Damage XX gave a lifetime prediction 15% less than the real failure time of the weldment laboratory test. Due to the complexity of creep damage behaviour in weldment, Hall and Hayhurst (1991) reported that good predictions were obtained through the use of Damage XX to simulate the pressure vessel laboratory test.

6.3 Details of the Nodal Force Calculator for the Internal Pressure Loading of the Tube

In order to set the internal pressure loads for the tube, a uniform load should be considered and a nodal force calculator developed by the author's colleague Feng Tan is utilized to calculate the equivalent nodal loads information for the FE model. The calculator includes two parts: the axial nodal force information and the radial nodal force information.

a) *Axial nodal force information*:

The calculator for the axial nodal force requires the inner and outer radius of each element and the expected uniform load. The nodal forces for each node from inner to outer can be calculated given the inner radius of each element, outer radius of each element and the expected uniform load.

The axial nodal force applied on the top boundary of the FE model can be calculated by:

$$F_i = \frac{1}{6}({r_{i+1}}^2 + r_i r_{i+1} - 2{r_i}^2)\sigma_z \tag{6.1}$$

$$F_{i+1} = \frac{1}{6}\left(2{r_{i+1}}^2 - r_i r_{i+1} - {r_i}^2\right)\sigma_z \tag{6.2}$$

Where the F is the nodal force; σ_z is the axial stress; r is the radius; i and i+1 are nodal numbers in the radial direction.

b) *Radial nodal force information*:

The calculator for the radial nodal force requires the total element number in the radial direction, the outer radius of the FE model, the expected uniform load and the distance between the adjacent two nodes on the inner surface of the model.

The nodal forces for each node from bottom node to top node on the inner surface can be given from the outer radius of the FE model, the expected uniform load and the

distance between the adjacent two nodes on the inner surface of the model. The radial nodal force applied on the vertical boundary of the FE model can be calculated by:

$$F_j = F_{j+1} = \frac{\sigma_r}{2} r_{jj+1} L_{jj+1} \tag{6.3}$$

Where the F is the nodal force; σ_r is the radial stress; r is the radius; L is the distance between node j and j+1; j and j+1 are nodal numbers.

6.4 Specifications of the Weldment FE Model in HITSI

6.4.1 The Mesh and Boundary Conditions

The diagram in Figure 6.2 shows the axisymmetric FE model used to represent the thick-steam pipe weldment case; this diagram is also used in HITSI for the generation of mesh and boundary conditions information.

The FE model with the mesh information is shown in Figure 6.3 and this FE model has 140 nodes and 233 elements.

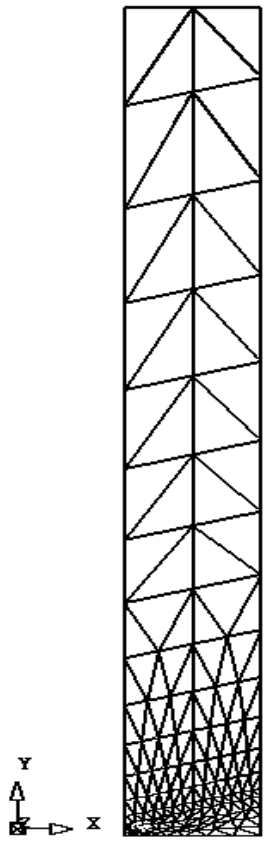


Figure 6.3: The FE mesh information

The boundary conditions are summarized in Table 6.4.

Table 6.4: The boundary conditions

Node number	Node No.1	Node No.2	Node No.3	Node No.4	Node No.5	Node No.6	Node No.7	Node No.8
Constraint in x direction	open	open	open	open	open	open	open	open
Constraint in y direction	shut	shut	shut	shut	shut	shut	shut	shut

6.4.2 The Material Properties

The material constants of the Kachanov-Rabotnov creep damage constitutive equation have been reported by Hall and Hayhurst (1991). The material constants are given units of stress in (MPa), strain in (%) and time in hours. These constants are shown in Table 6.5 and have been used in the verification of HITSI through the numerical investigation of the same steam pipe weldment case.

Table 6.5: The material constants used for the creep damage test of 2.25Cr1Mo:0.5Cr0.5Mo0.25V steam pipe weldment (Hall and Hayhurst, 1991)

Material	**Stress Range**	***K***	***n***	***m***	***M***	***Φ***	***χ***	***a***
Parent Metal	$\sigma \leq \sigma^{\wedge}$	2.8531d-14	4.8971	-0.2031	1.4522d-10	5.4141	3.0110	0.5955
	$\sigma > \sigma^{\wedge}$	1.3485d-25	10.3442	-0.2031	8.8846d-19	12.5486	6.9613	0.5955
HAZ (G.C.P.)	$\sigma \leq \sigma^{\wedge}$	1.0358d-7	1.3654	-0.1700	2.3062d-10	1.4231	2.7858	0.4298
	$\sigma > \sigma^{\wedge}$	8.7207d-25	8.9364	-0.1700	1.3459d-9	14.8589	9.0982	0.4298
Weld Metal (Fine)	$\sigma \leq \sigma^{\wedge}$	2.93965d-12	4.3680	-0.2031	1.15878d-9	4.9667	2.8554	0.4298
	$\sigma > \sigma^{\wedge}$	1.3485d-25	7.2496	-0.2031	1.7418d-15	8.9029	5.7669	0.4298

The different material zones in this FE model are presented in Figure 6.4.

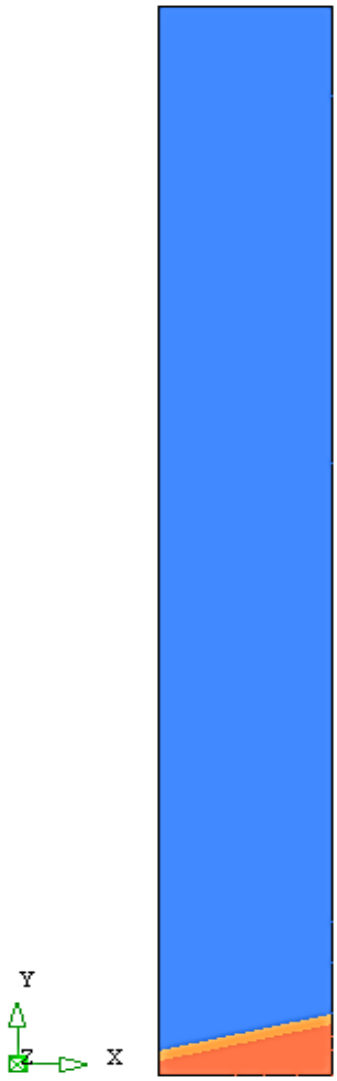


Figure 6.4: The different material zones in FE model

The three materials weldment FE model is used and the discrete regions of the FE mesh are assigned the creep properties of the parent metal in blue, the HAZ in yellow and the weld metal in red. According to the laboratory test (Coleman et al., 1985), the elastic modulus E of each material zone in this weldment case is assumed to be the same, with a value of E = 170 GPa.

6.4.3 The Internal Pressure Loading Information

In this FE model, the uniform loads in the axial and radial directions are 34.6 MPa and 45.5 MPa, respectively. The equivalent nodal loads information can be obtained by the nodal force calculator for this FE model and are shown in Table 6.6 and Table 6.7 for the axial and radial directions, respectively.

Table 6.6: The equivalent nodal loads information in axial direction

Node number	Node force
Node No.138	6.4875000E+04
Node No.139	1.5051000E+05
Node No.140	8.5635000E+04

Table 6.7: The equivalent nodal loads information in radial direction

Node number	Node force
Node No.1	3.9243750E+03
Node No.9	7.8487500E+03
Node No.18	7.8487500E+03
Node No.29	7.8487500E+03
Node No.42	1.4389375E+04
Node No.57	3.1395000E+04
Node No.72	5.2325000E+04
Node No.85	7.3255000E+04
Node No.96	9.4185000E+04
Node No.105	1.1511500E+05
Node No.112	1.3604500E+05
Node No.117	1.5697500E+05
Node No.120	1.7790500E+05
Node No.123	1.9883500E+05
Node No.126	2.1976500E+05
Node No.129	2.4069500E+05
Node No.132	2.6162500E+05
Node No.135	2.6685700E+05
Node No.138	1.3081200E+05

6.5 Verification the FE codes in FE Model

Verification of the FE codes (to ensure the validity of the mesh information, the boundary conditions, the loads information, the element stiffness integration and assembly, the solution of the equilibrium equation and results recovery at integrating points that are used in this complex multi-material zones weldment case) is essential

before analysis of the creep damage stage. The thick-steam pipe weldment FE model in Figure 6.3 is utilized in the verification.

In this case, the uniform pressure in the axial directions is 34.6 MPa and it should be distributed uniformly in the FE model. Thus, the initial stress values and distributions calculated by HITSI before the start of the time loop iteration should agree with the expected stress values. The initial stresses in the axial directions are shown in Figure 6.5. Furthermore, the displacement and initial strain distributions in the FE model are shown in Figure 6.6 and Figure 6.7, respectively.

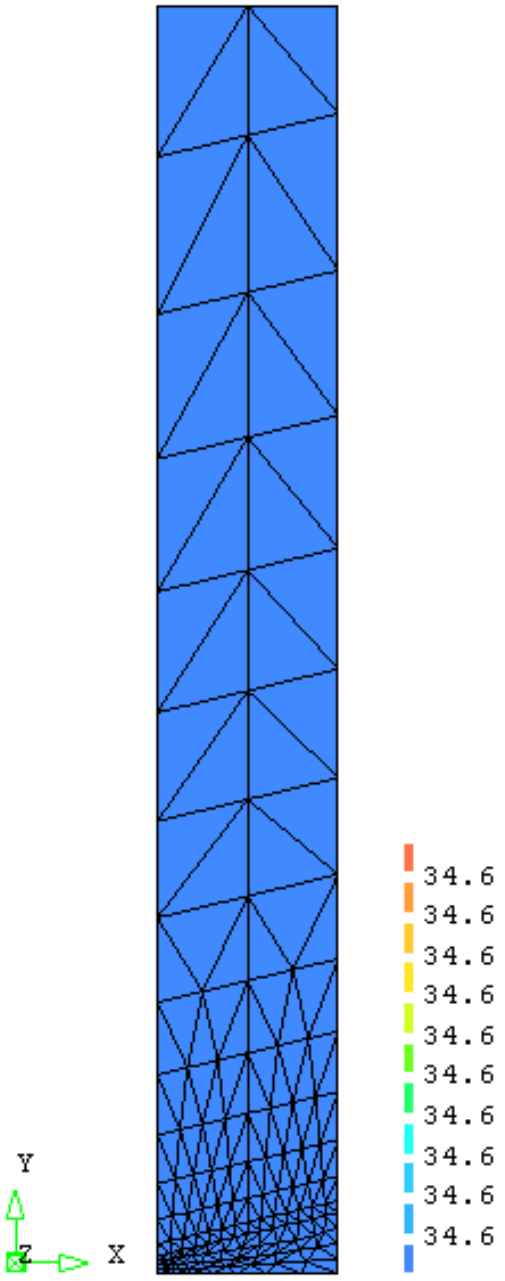


Figure 6.5: The initial stress distribution in axial direction

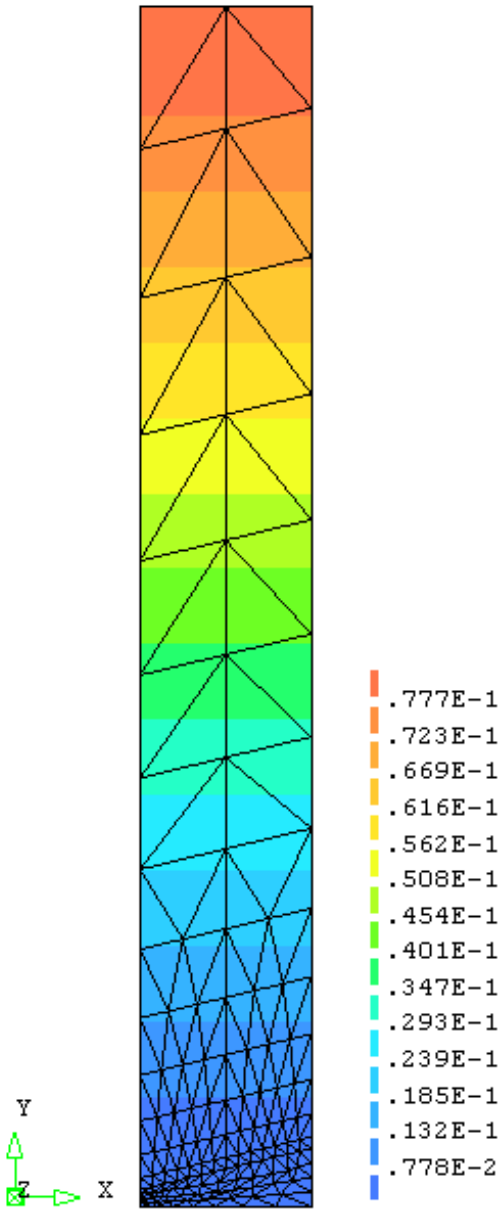


Figure 6.6: The displacement distribution in axial direction

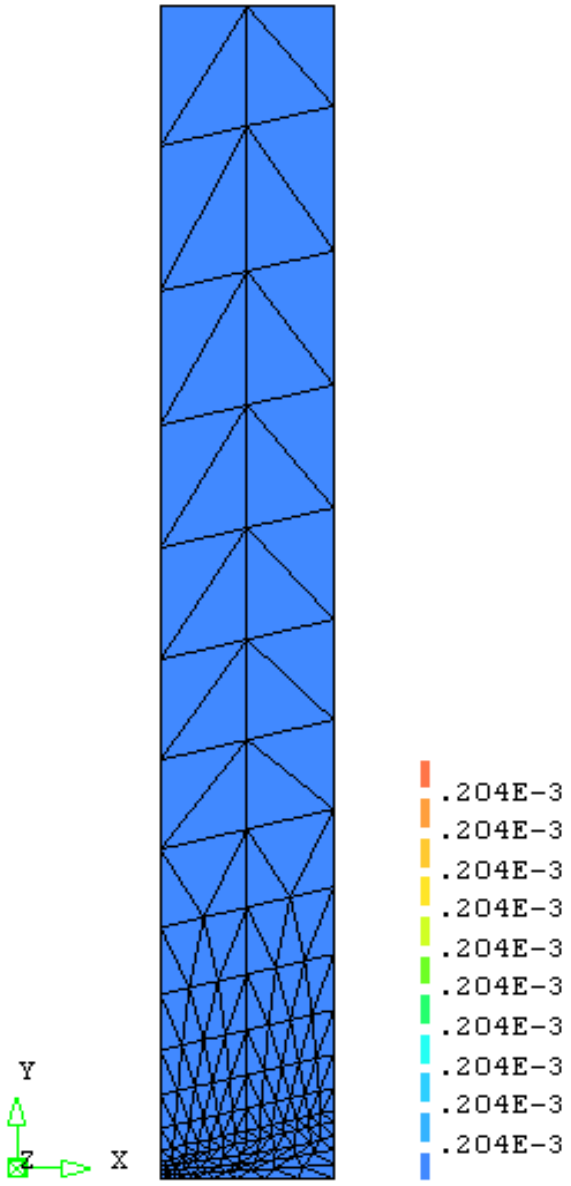


Figure 6.7: The elastic strain distribution in axial direction

Figure 6.5 shows that the initial stress distributions in the axial direction are uniformly distributed and the stress value is shown to be in good agreement with the expected stress values. Figure 6.6 and Figure 6.7 show the displacement and elastic strain are both distributed uniformly. Therefore, the validity of the mesh information, the boundary conditions, the loads information, the element stiffness integration and assembly, the solution of the equilibrium equation and the recovery of results at integrating points have been verified and the FE codes can be used in further investigations.

6.6 Evolution of Creep Damage Fields

6.6.1 Damage Distribution

The predicted damage distributions from HITSI are presented against the background of the life fractions of 0.12%, 20.4%, 45.2%, 63.5%, 77.2%, 80.8%, 87.0% and 99.9%. The first failed element occurred at element number 56 with the life fraction of 20.4%.

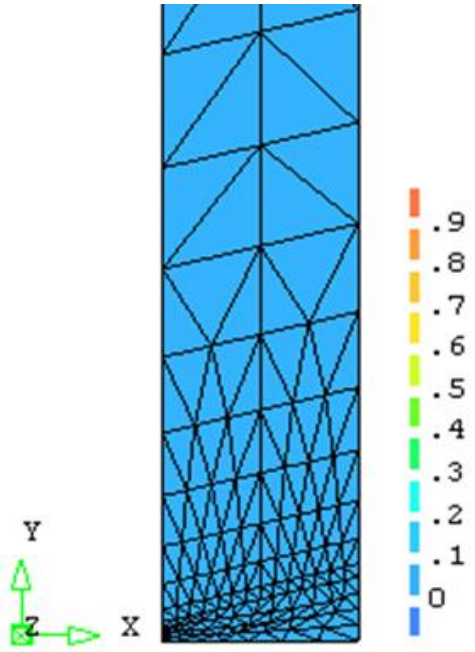


Figure 6.8: The damage distribution at life fractions of 0.12%

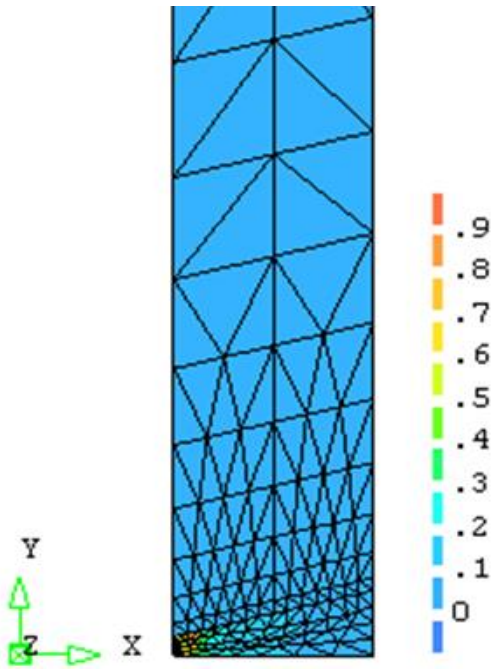


Figure 6.9: The damage distribution at life fractions of 20.4%

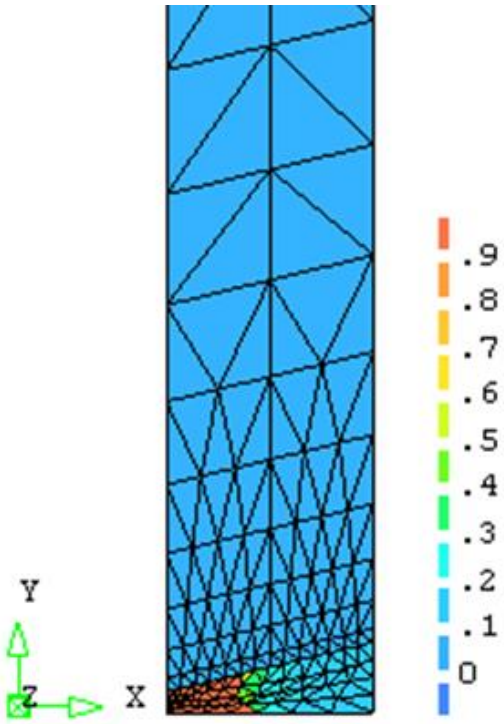


Figure 6.10: The damage distribution at life fractions of 45.2%

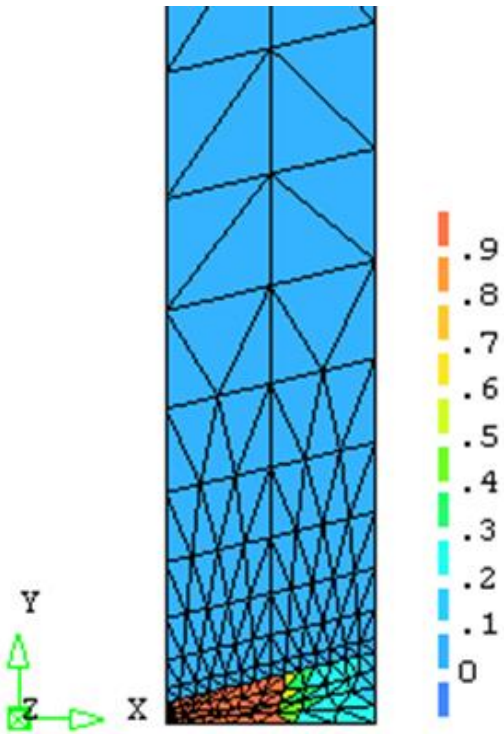


Figure 6.11: The damage distribution at life fractions of 63.5%

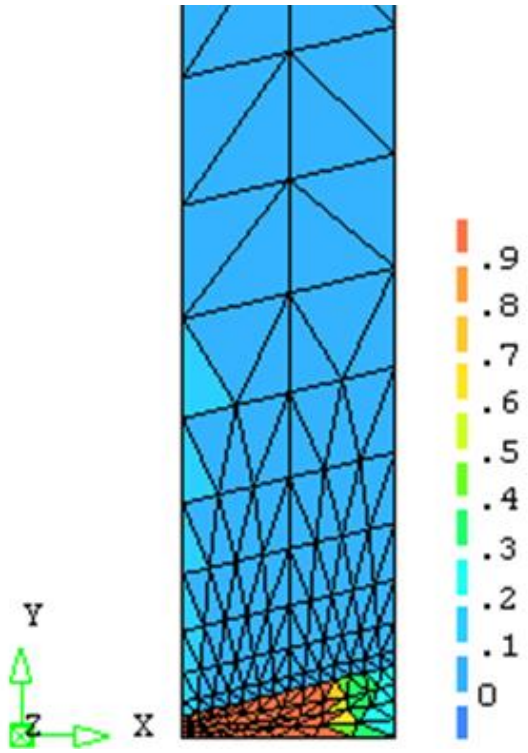


Figure 6.12: The damage distribution at life fractions of 77.2%

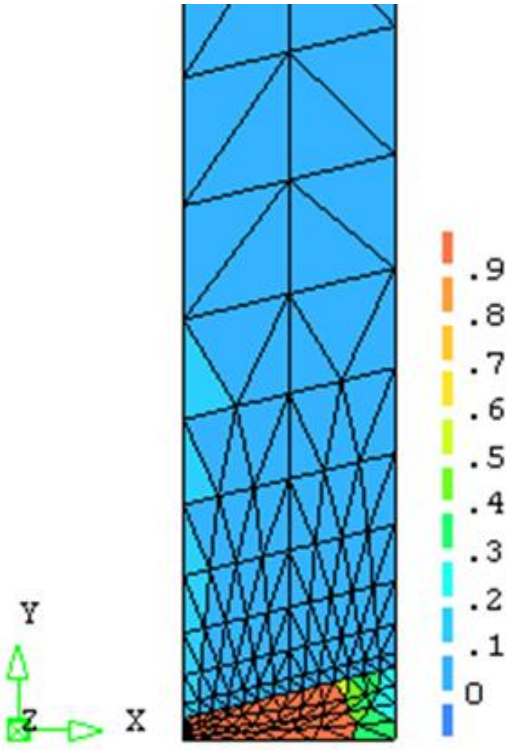


Figure 6.13: The damage distribution at life fractions of 80.8%

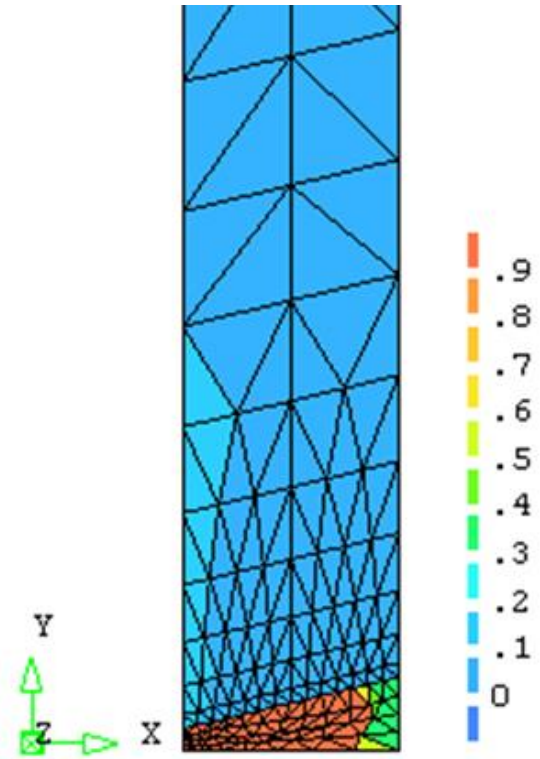


Figure 6.14: The damage distribution at life fractions of 87.0%

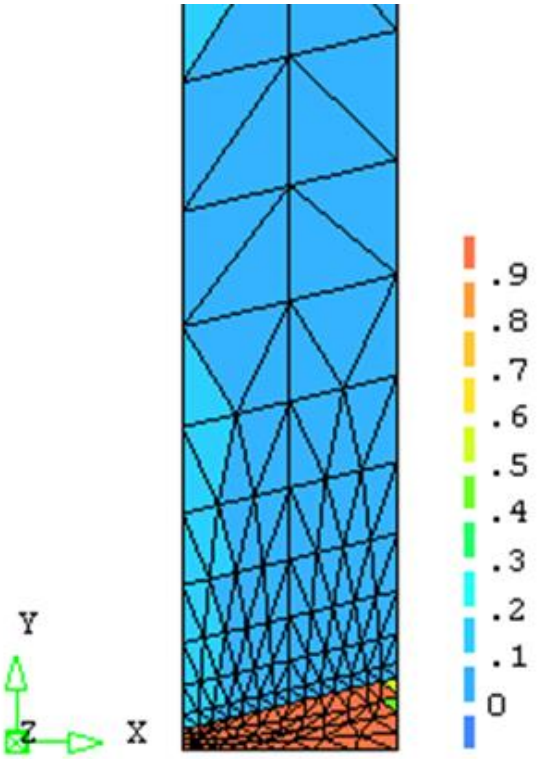


Figure 6.15: The damage distribution at life fractions of 99.9%

Table 6.8: A brief summary of damage evolution of the 2.25Cr1Mo:0.5Cr0.5Mo0.25V steam pipe weldment from the FE model in HITSI

Time/h	t/t_f	Characteristics
50	0.12%	The creep damage rate is increasing rapidly at this stage and the initial damage rate is highest between the HAZ and weld at the inner bore.
8320	20.4%	The first failed element occurs between the weld metal and HAZ
18387	45.2%	The damage rate is declining at this stage and the intense damage on the fusion boundary spread both inward and outwards
25832	63.5%	The damage distribution in the weld is more uniform and becoming wider towards to outer bore and the growth of creep damage rate is beginning to stabilize at this stage
31405	77.2%	The maximum damaged zone is becoming more and more intense in the weld metal and the HAZ at this stage
32869	80.8%	The centroid of the damaged zone has moved into the weld metal
35392	87.0%	the damaged zone on the fusion boundary and weld metal now have higher damage levels
40680	99.9%	The coalescence of the most damaged zones into two main localized damaged regions and the weldment is called failure at this stage

6.6.2 Creep Strain Rate in FE Model

The distributions of the predicted creep strain rate from HITSI are presented against the background of life fractions from when the first failed element occurred to the final failure time.

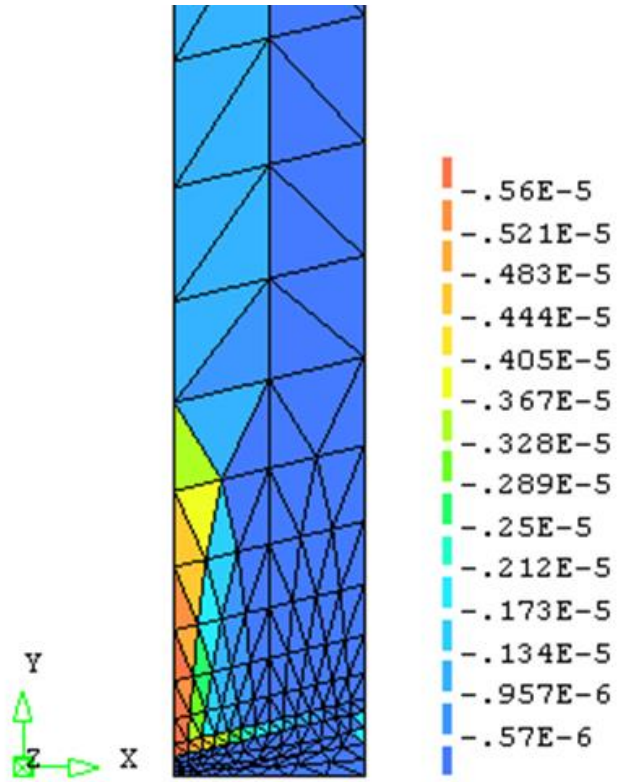


Figure 6.16: The creep strain rate in radial direction when the first failed element occurred

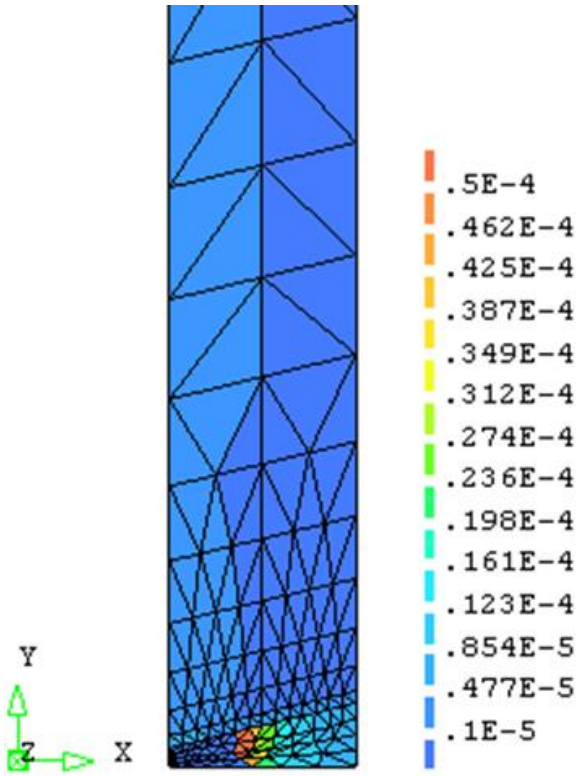


Figure 6.17: The creep strain rate in axial direction when the first failed element occurred

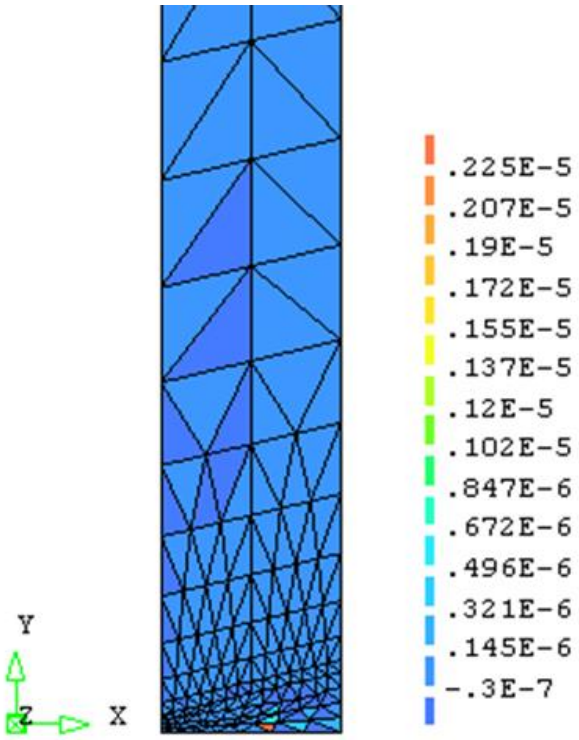


Figure 6.18: The creep strain rate in shear stress (r-z) direction when the first failed element occurred

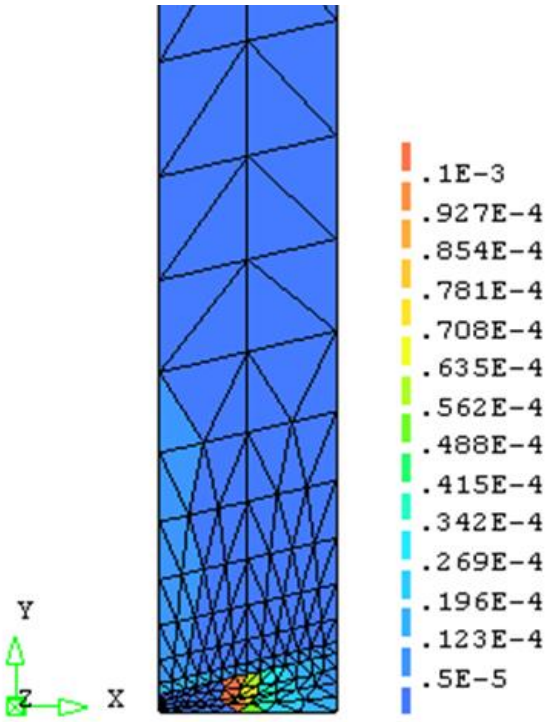


Figure 6.19: The creep strain rate in hoop stress direction when the first failed element occurred

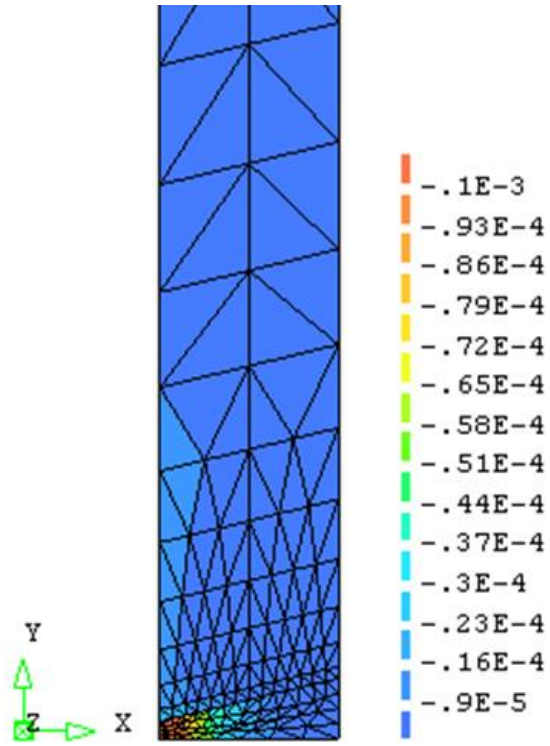


Figure 6.20: The creep strain rate in radial direction at failure time

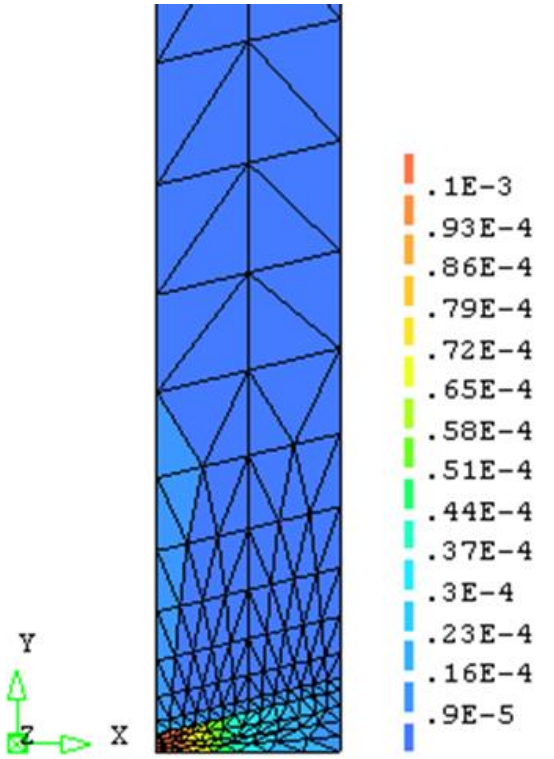


Figure 6.21: The creep strain rate in axial direction at failure time

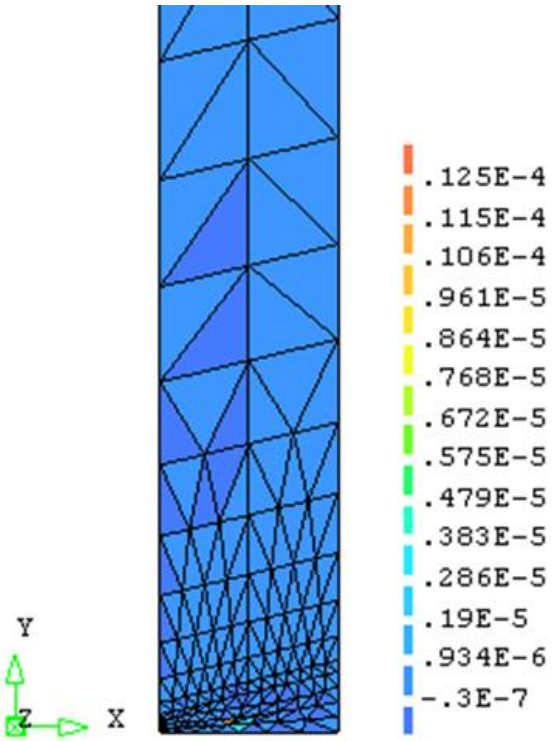


Figure 6.22: The creep strain rate in shear stress (r-z) direction at failure time

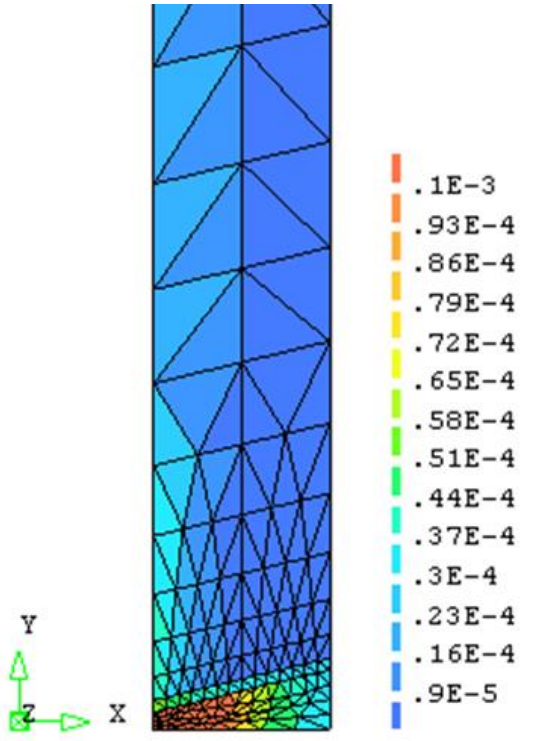


Figure 6.23: The creep strain rate in hoop stress direction at failure time

6.6.3 The Stress and Displacement Distribution at Failure Time

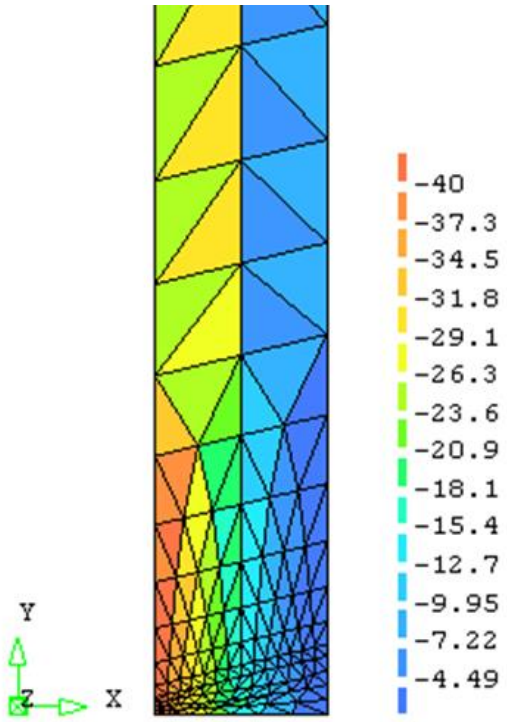


Figure 6.24: The radial stress distribution at failure time

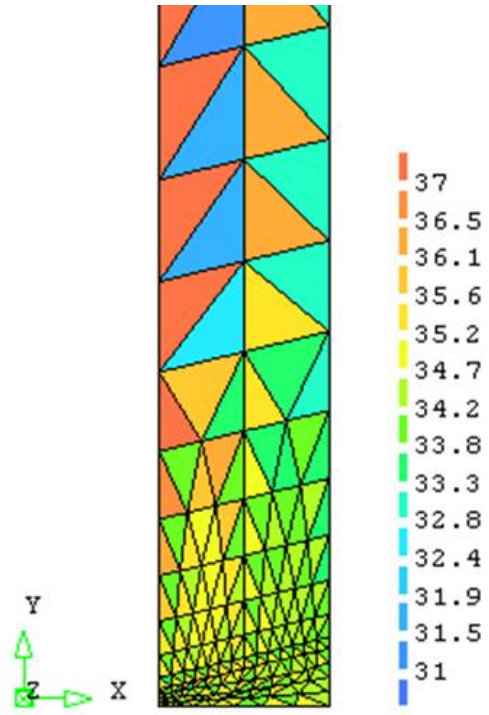


Figure 6.25: The axial stress distribution at failure time

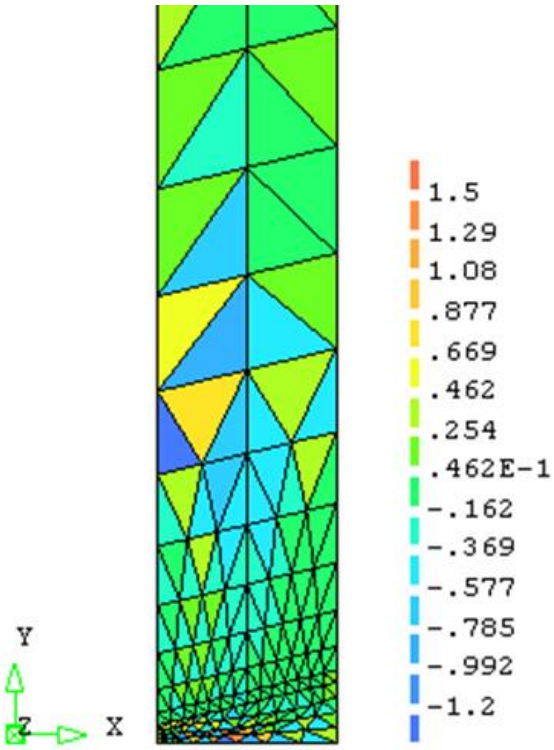


Figure 6.26: The shear stress (r-z) distribution at failure time

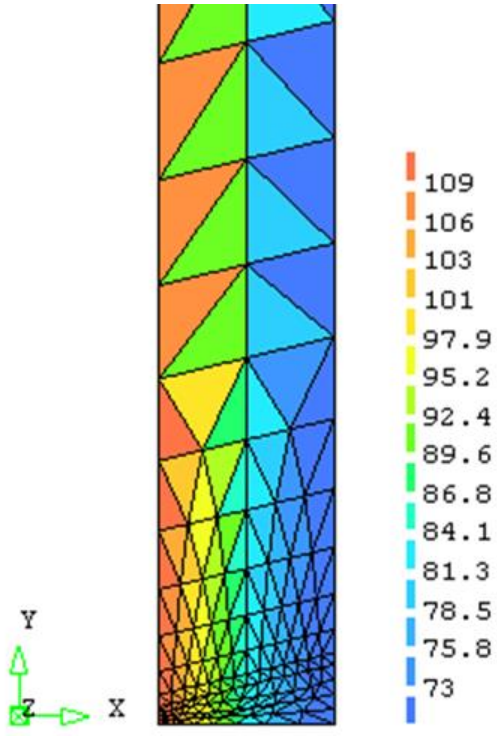


Figure 6.27: The hoop stress distribution at failure time

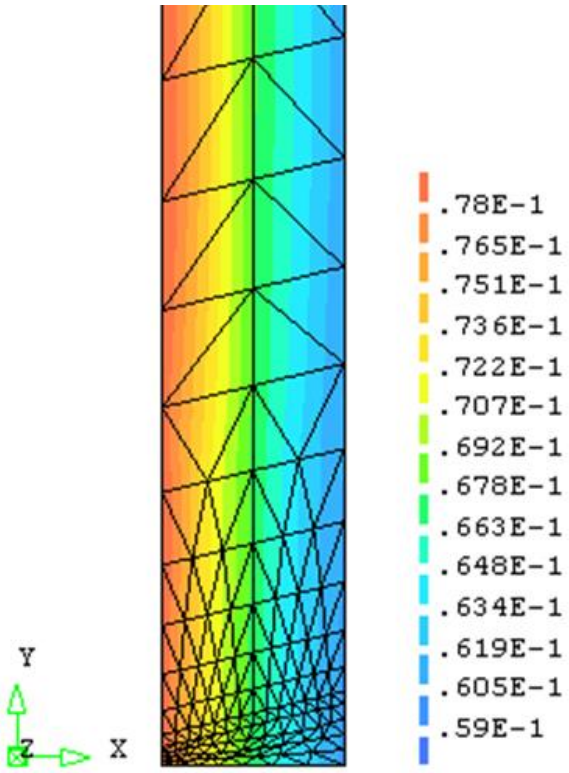


Figure 6.28: The radial displacement distribution at failure time

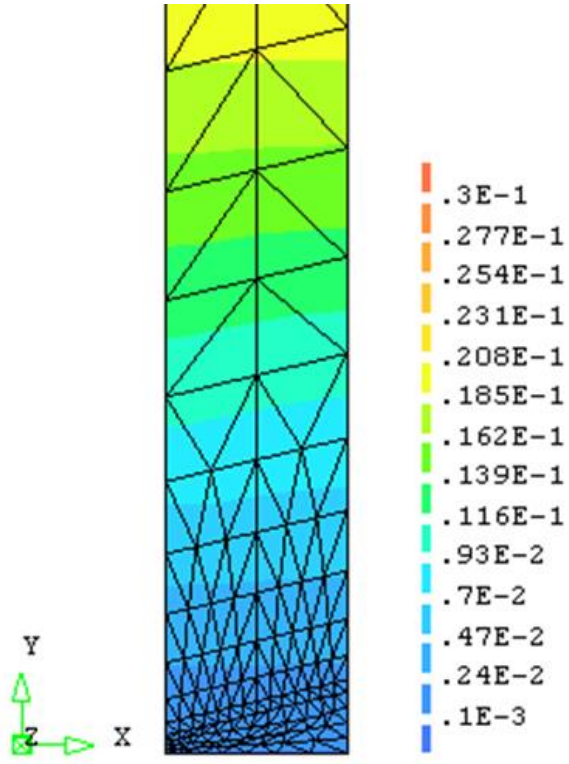


Figure 6.29: The axial displacement distribution at failure time

6.6.4 Discussion

The elements with the maximum elastic stress concentrate on the inner bore. Therefore the creep damage rate increases rapidly and the initial damage rate is highest between the HAZ and the weld at the inner bore at the lifetime fraction of 0.12% in Figure 6.8. Due to the different material zones in this case, different body loads in the three regions are produced with the growth of creep deformation and this phenomenon causes the stresses to redistribute radially outwards. Thus, a first failed element is observed in Figure 6.9 and more failed elements are observed on the fusion boundary spreading both inward and outwards in Figure 6.10. Figure 6.11 and Figure 6.12 show that the damage distribution in the weld is more uniform and becomes wider towards to outer bore; the growth of the creep damage rate is beginning to stabilize at this stage because of the nature of the creep damage constitutive equations. With the increasing time, the centroid of the damaged zone moved into the weld metal in Figure 6.13 and the damaged zone on the fusion boundary and weld metal have obviously higher damage levels in Figure 6.14. Lastly, the crack has propagated through the pipe at lifetime fraction of 99% in Figure 6.15 and the weldment is called failure at this time.

The stress redistribution causes the different creep damage rates in weldment. Odqvist (1974) has presented the analysis of an internally pressurised thick walled pressure vessel due to Bailey (1935). The equations for the hoop, radial and axial stresses have been derived by Odqvist (1974) and the equations are shown as follows:

$$\sigma_\theta = \frac{P_0}{\left\{(\frac{a}{b})^{\frac{2}{n}} - 1\right\}} \left\{1 - (1 - \frac{2}{n})(\frac{r}{b})^{\frac{-2}{n}}\right\} \tag{6.4}$$

$$\sigma_r = \frac{P_0}{\left\{(\frac{a}{b})^{\frac{2}{n}} - 1\right\}} \left\{1 - (\frac{r}{b})^{\frac{-2}{n}}\right\} \tag{6.5}$$

$$\sigma_z = \frac{1}{2}\,(\sigma_r + \sigma_\theta) \tag{6.6}$$

Where n is the creep exponent of stress in Norton's law; σ_θ is the hoop stress; σ_r is the radial stress; σ_z is axial stress; P_0 is internal stress; r is radial distance and a/b the internal diameter ratio (Odqvist, 1974).

It is noted that Hall and Hayhurst (1991) used FEM to represent the thick pressure vessel of Odqvist (1974) in the analysis of creep damage behaviour of weldment

through the use of Damage XX. In order to verify the in-house FE software HITSI, the FE results such as hoop stress, shear stress, radial and axial stresses have been compared with the analytical results from Odqvist's equations. The FE simulated results such as the radial stress, axial stresses, shear stress and hoop stress are shown in Figure 6.24, Figure 6.25, Figure 6.26 and Figure 6.27. The FE results have been shown to be in good agreement with the analytical results by Odqvist (1974).

According to Table 6.1, the transverse cracks have been observed in the coarse columnar regions of the weld metal and HAZ; the cracks then spread into the weld metal to cause the rupture. Thus, good agreement of the creep damage evolution has been obtained by a comparison between Table 6.1 and Table 6.8 for the same weldment case and approximate damage distributions have been predicted on the centre line of the weld at the steam pipe surface. The lifetime prediction's relative error between the software HITSI and pressure vessel laboratory test by Coleman is shown in Table 6.9.

Table 6.9: The lifetime prediction's relative error between HITSI and pressure vessel laboratory test

Rupture time from in-house software HITSI	**Rupture time from pressure vessel laboratory test**
40680 hours	46000 hours
Rupture time relative error $= \left\lvert \frac{40680 - 46000}{46000} \right\rvert = 0.12$	

The FE simulated results for the evolution of creep damage distributions and the rupture time in Table 6.9 are seen to be in good agreement between the pressure vessel laboratory test and FE simulated results by HITSI.

According to Hall and Hayhurst (1991), the predicted damage distributions in the weldment FE model through the use of Damage XX are presented against the background of failure time at life fractions of 0.12%, 45.2%, 63.5%, 77.2%, 80.8%, 87.0%, 99% and 99.9%. By a comparison between Table 6.2 and Table 6.8, the damage evolution and distributions for the weldment case from the FE software Damage XX and HITSI show a similar description. The lifetime prediction's relative error between the FE model of the weldment case by Damage XX and HITSI is shown in Table 6.10.

Table 6.10: The lifetime prediction's relative error between the in-house FE software HITSI and the FE solver Damage XX

Rupture time from in-house software HITSI	**Rupture time from FE solver Damage XX**
40680 hours	39119 hours
Rupture time relative error $= \left\|\frac{40680 - 39119}{39119}\right\| = 0.04$	

The predicted failure time by HITSI is 40680 hours. Table 6.9 and Table 6.10 show that the results obtained from HITSI agree with the actual failure time of the pressure vessel laboratory test and the results obtained from Damage XX, respectively.

Table 6.3 shows the relative error between the simulated failure time from Damage XX and the failure time of the pressure vessel laboratory test is 0.15. In Table 6.9, the lifetime prediction's relative error between HITSI and the pressure vessel laboratory test is 0.12, which is closer the actual failure time of the pressure vessel laboratory test.

The in-house FE software HITSI has been shown to predict reasonably well the creep damage behaviour and failure history of the pressure vessel weldment.

6.7 Investigation of Different Numerical Integration Methods

6.7.1 Introduction

The FE solution critically depends on the selection of the size of time steps associated with an appropriate integration method. The Euler integration subroutine and the 4th order Runge-Kutta integration subroutine have been developed by the author's colleague Feng Tan in this research group. The two subroutines have been tested and applied to HITSI for the weldment case. The computational efficiency and accuracy have been investigated and discussed. The creep damage distribution at failure time with the Euler integration scheme is in Figure 6.15, while the creep damage distribution at failure time with the 4th order Runge-Kutta integration subroutine is shown in Figure 6.30.

The evaluation of creep damage fields for the 2.25Cr 1Mo: 0.5Cr 0.5Mo 0.25V thick steam pipe weldment case using the Euler and the Runge-Kutta integration methods is very similar; however, the biggest difference is that the cost of computing time by the

Runge-Kutta integration method for this case is less than that of the Euler integration method. The computational efficiency between the Euler and the Runge-Kutta integration methods has been summarised in Table 6.11.

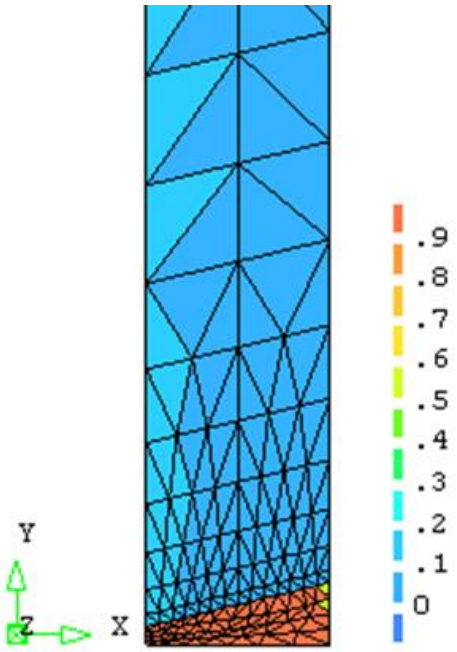


Figure 6.30: The creep damage distribution at failure time with Runge-Kutta integration method

Table 6.11: The computational efficiency between the Euler integration scheme and the Runge-Kutta integration scheme

Integration method	Computing time	Rupture time/hour
Euler	89s	40680
Runge-Kutta	81s	40510

According to Table 6.11, the cost of computing time for the thick steam pipe weldment case by the Runge-Kutta integration method is almost 10% less than that by the Euler integration method. The rupture time obtained by the Euler method is approximately 0.5% longer than that obtained by Runge-Kutta integration method.

6.7.2 Discussion

The well-known Euler method is only conditionally stable and the stability condition is rather stringent. It requires extremely small time steps to ensure the convergence of iterations and accuracy of calculations. The 4th order Runge-Kutta integration method gives a higher order of accuracy at intermediate points (Hagler, 1987), while the local

truncation error for the Runge-Kutta method is Δt^5 and for the Euler Method is Δt^2. The 4th order Runge-Kutta method requires solving the set of system equations four times in each integration step, while the Euler method requires solving the set of system equations only once. Thus, each integration method has its advantages and disadvantages; but for a large set of system equations problem, the Runge-Kutta method has obvious advantages because the fewer iterations can improve the computational efficiency significantly.

In this case, the number of elements and nodes are 233 and 140, respectively. The number of system equilibrium equations is 420. Although the Runge-Kutta method requires solving the set of system equations a factor of four times more than the Euler method (and the larger integration time per step may be a cost to the Runge-Kutta method) fewer iterations can still improve the computational efficiency significantly and large time steps can be employed with only slightly more computational effort than for the Euler method. Therefore, the total computation time cost can be reduced by the Runge-Kutta method in problems with a large set of system equations. In this weldment case, the use of Runge-Kutta integration method can save the cost of computing time.

6.8 Investigation of Normalized Creep Damage Constitutive Equation

6.8.1 Introduction

This section investigates the efficiency of the normalization of the constitutive and damage laws (Hayhurst et al., 1984) via the analysis of creep damage behaviour of the Cr-Mo-V steam pipe weldment case. Hayhurst's research group (Hayhurst et al., 1984) has proposed the algorithm of the normalization of the constitutive and damage laws; later on, Hall and Hayhurst (1991) reported that the normalization of the constitutive and damage laws can reduce the round-off error. The normalized Kachanov-Rabotnov creep damage constitutive equation's subroutine was developed by the author's colleague Feng Tan in this research group and this subroutine has been used in modelling the steam pipe weldment case for investigating the efficiency of the normalized creep damage constitutive equation. Comparisons are made between the creep damage behaviours predicted by the normalized creep damage constitutive equation and non-normalized creep damage constitutive equation. The results show that the normalized constitutive and damage laws can improve the computing efficiency.

6.8.2 The Normalized Creep Damage Constitutive Equation and Material Property

According to the normalization of the constitutive and damage laws (Hayhurst et al., 1984), the normalized stress and strain are defined as $\Sigma_{ij} = \sigma_{ij} / \sigma_0$, $S_{ij} = s_{ij} / \sigma_0$ and $V_{ij} = \varepsilon_{ij} / e_0$, where the e_0 is the uni-axial elastic strain at a constant stress of σ_0 and this constant stress has been selected as the internal pressure, given as $e_0 = \sigma_0 / E$ where E is Young's modulus. The Kachanov-Rabotnov creep damage constitutive equation can be rewritten as:

$$\frac{dV_{ij}}{dt} = \frac{3}{2}\frac{KE{\sigma_0}^{n-1}t^m}{(1-\omega)^n}\sum_{e}^{n-1}\frac{S_{ij}}{\sigma_0} \tag{6.7}$$

$$\frac{d\omega}{dt} = g\frac{M}{\emptyset+1}\frac{{\sigma_0}^{x}t^m}{(1-\omega)^{\emptyset}}\ (\frac{\Delta\left(\sigma_{ij}\right)}{\sigma_0})^n \tag{6.8}$$

The normalized time T_n and the constant V_u are shown as:

$$dT_n = KE{\sigma_0}^{n-1}t^m dt \tag{6.9}$$

$$V_u = \frac{KE}{M}{\sigma_0}^{(n-x-1)} \tag{6.10}$$

Thus, the Equation 6.7 and Equation 6.8 can be represented as:

$$\frac{dV_{ij}}{dT_n} = \frac{3}{2(1-\omega)^n}\sum_{e}^{n-1}S_{ij} \tag{6.11}$$

$$\frac{d\omega}{dT_n} = \frac{\Delta\, x}{V_u(\emptyset+1)}\frac{\Sigma_{ij}}{(1-\omega)^{\emptyset}} \tag{6.12}$$

Where K, n, m, M, Φ, χ and a are material constants used for the creep damage test of the 2.25Cr1Mo:0.5Cr0.5Mo0.25V steam pipe weldment; ω is creep damage, σ_0 is spherical stress tensor and S_{ij} is deviator stress tensor. Equation 6.11 and Equation 6.12 are used in the FE program for the analysis of steam pipe weldment case.

The normalized material constants used in this case have been defined by Hall and Hayhurst (1991) and they are shown in Table 6.12.

Table 6.12: The normalized material constants (Hall and Hayhurst, 1991)

Material	Stress Range	*K*	*n*	*m*	*M*	*Φ*	*χ*	*a*
Parent Metal	$\sigma \leq \sigma^{\wedge}$	2.8531d-14	4.8971	-0.2031	3.2641d-11	5.4141	3.0110	0.5955
	$\sigma > \sigma^{\wedge}$	1.3485d-25	10.3442	-0.2031	8.8846d-19	12.5486	6.9613	0.5955
HAZ (G.C.P.)	$\sigma \leq \sigma^{\wedge}$	1.0358d-7	1.3654	-0.1700	9.5176d-11	1.4231	2.7858	0.4298
	$\sigma > \sigma^{\wedge}$	8.7207d-25	8.9364	-0.1700	1.3459d-9	14.8589	9.0982	0.4298
Weld Metal (Fine)	$\sigma \leq \sigma^{\wedge}$	2.93965d-12	4.3680	-0.2031	1.9421d-10	4.9667	2.8554	0.4298
	$\sigma > \sigma^{\wedge}$	1.3485d-25	7.2496	-0.2031	1.7418d-15	8.9029	5.7669	0.4298

6.8.3 Damage Field and Macro-cracking

The weld model is a three-material model, namely the parent metal, the HAZ and the weld metal. The FE model has 140 nodes and 233 elements. The normalized Kachanov-Rabotnov creep damage constitutive equation and material constants are used in the FE modelling of the damage evolution for the weldment case. Firstly, the processes of the damage evolution obtained by HITSI are shown in Figure 6.31, Figure 6.32 and Figure 6.33 at life fractions of 0.12%, 20.4% and 99.9%. Then, the normalized radial stress, axial stress and hoop stress at the failure time are shown in Figure 6.34, Figure 6.35 and Figure 6.36. Lastly, the characteristics of creep damage fields by the normalized creep damage constitutive equation are summarised in Table 6.13.

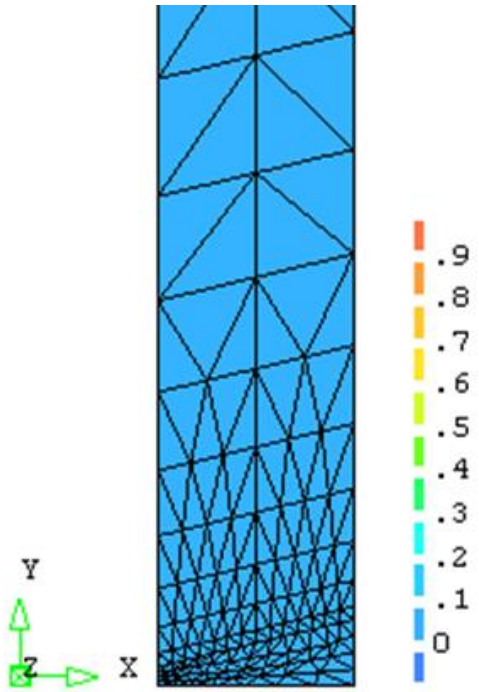


Figure 6.31: The damage distribution with normalized constitutive equation at life fractions of 0.12%

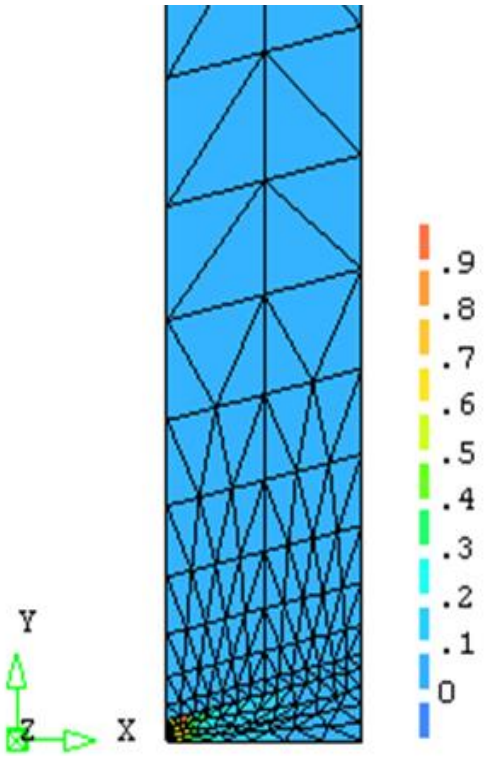


Figure 6.32: The damage distribution with normalized constitutive equation at life fractions of 20.4%

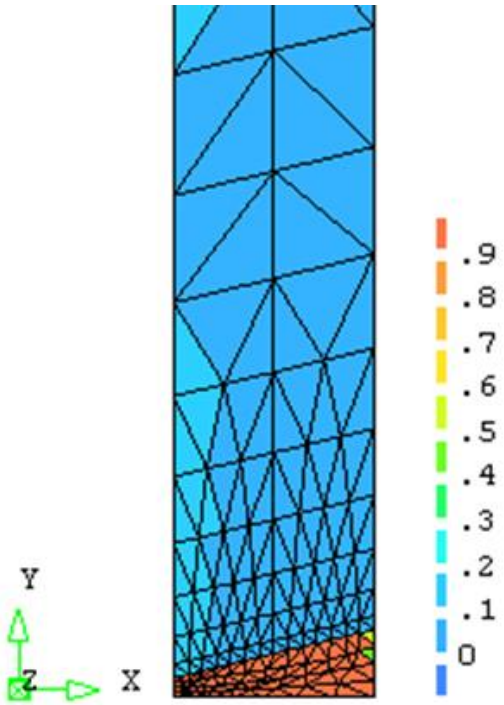


Figure 6.33: The damage distribution with normalized constitutive equation at life fractions of 99.9%

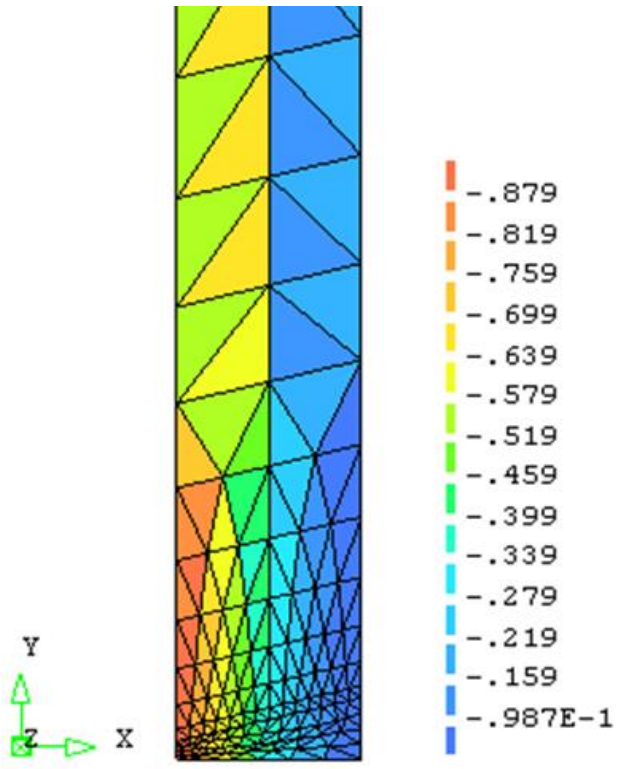


Figure 6.34: The normalized radial stress distribution at failure time

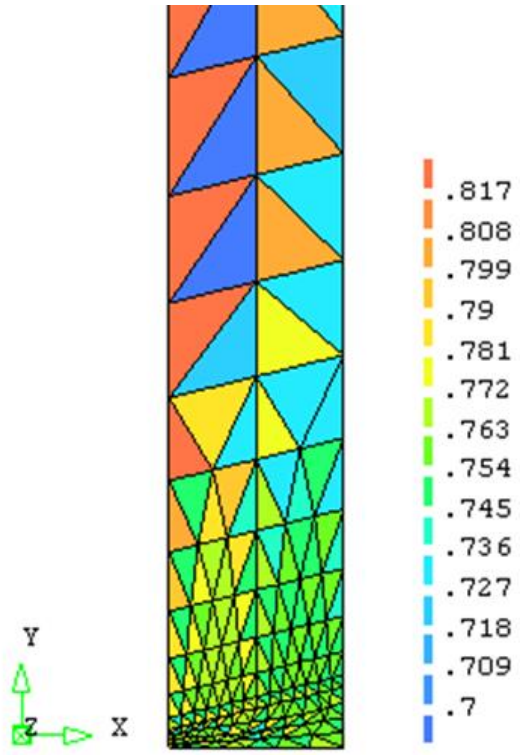


Figure 6.35: The normalized axial stress distribution at failure time

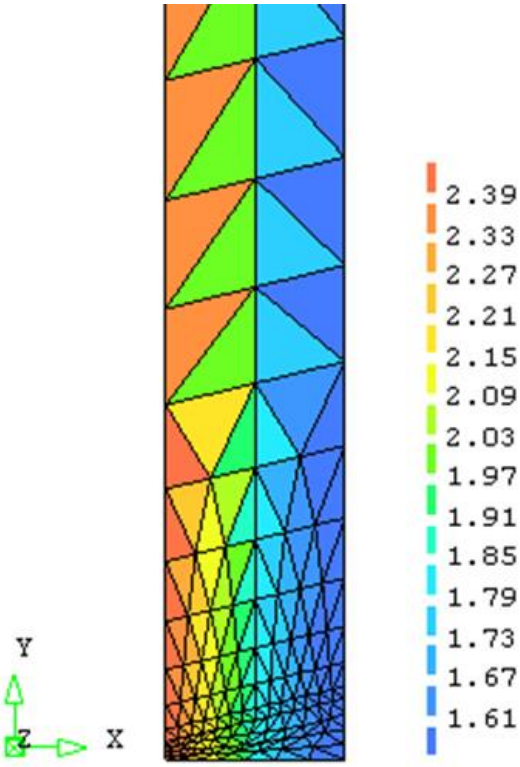


Figure 6.36: The normalized hoop stress distribution at failure time

Table 6.13: A brief summary of damage evolution of the 2.25Cr1Mo:0.5Cr0.5Mo0.25V steam pipe weldment by in-house FE software HITSI with normalized creep damage constitutive equation

Time/h	t/t_f	Characteristics
48	0.12%	The creep damage rate is increasing rapidly at this stage and the initial damage rate is highest between the heat affected zone (HAZ) and weld at the inner bore
7961	20.4%	The first failed element occurs between the weld metal and the heat affected zone (HAZ)
17818	45.2%	The damage rate is declining at this stage and maximum damaged region occurring along the fusion boundary between the weld metal and the heat affected zone (HAZ)
25032	63.5%	The damage distribution in the weld is more uniform and becomes wider
30432	77.2%	The maximum damaged zone on the fusion boundary are becoming more and more intense
31851	80.8%	The centroid of the damaged zone has moved into the weld metal
34295	87.0%	The damaged zone on the fusion boundary and weld metal now have higher damage levels
39420	99.9%	The coalescence of the most damaged zones into two main localized damaged regions

The creep damage rate increases rapidly at the primary stage and the initial damage rate is highest between the HAZ and the weld at the inner bore since the elements with maximum normalized stress are concentrated on the inner bore. This phenomenon causes the stresses to redistribute radially outwards. Thus, more damage has been observed in the weld metal and the first failed element occurred close the inner bore in Figure 6.32. With increasing time, the crack grows until it exists across the pipe from the inner bore to the outer bore and the weldment is called failure at this time. The damage evolution of the 2.25Cr1Mo:0.5Cr0.5Mo0.25V steam pipe weldment by the FE software HITSI with normalized creep damage constitutive equation has been summarized in Table 6.13.

6.8.4 Discussion

According to the damage evolution of the 2.25Cr1Mo:0.5Cr0.5Mo0.25V steam pipe weldment simulated by HITSI in Table 6.8 and Table 6.13, the results from the non-normalized constitutive equation and normalized constitutive equation give similar descriptions of the damage evolution in weldment where the non-normalized failure time is 40680 hours and normalized failure time is 39420 hours. Both agree with the experimental test by Coleman; however, the biggest difference is the cost of computing time where the non-normalized computing time is 89 seconds and the normalized computing time is 62 seconds.

Hall and Hayhurst (1991) reported that the normalization of the constitutive and damage laws can reduce the round-off error, which is the difference between the calculated approximation of a number and its exact mathematical value. In this FE model, the numerical analysis specifically tries to estimate the error by using the creep damage constitutive equation with a time integration algorithm.

Finite digits are used to represent real numbers. For example, the normalized axial stress is 0.78000000 MPa and it is rounded to two decimal places (0.78); the non-normalized axial stress is 34.627743691915313 MPa and it is also rounded to two decimal places (34.63). With the growth of the integrating calculations, numerical errors may accumulate due to the round-off error.

In order to reduce the round-off error, the number of digits should be increased to represent the real number. For example, the non-normalized axial stress 34.627743691915313 MPa can be rounded to ten decimal places (34.6277436919) and the numerical errors could be reduced with the growth of integrating calculations. However, the computer needs more storage to store the extra precision. As a result, the computing efficiency is reduced by increasing the number of digits. Thus, the use of normalized constitutive equation can significantly increase the computing efficiency.

6.9 Summary

This chapter presents the benchmark test of the computational FEM based CDM approach in-house FE software HITSI via the analysis of creep deformation and damage evolution of the 2.25Cr1Mo:0.5Cr0.5Mo0.25V steam pipe weldment case. The in-house FE software HITSI has been shown to predict reasonably well the failure history of the pressure vessel weldment.

The efficiency and accuracy of the numerical time integration schemes (Euler and Runge-Kutta) in FEM for creep damage analysis of weldment have been investigated in this chapter; the results reveal that the total computation time cost can be reduced by use of the Runge-Kutta method in problems with a large set of system equations.

The normalized creep damage laws (the Kachanov-Rabotnov creep damage constitutive equation) in FEM for creep damage analysis of weldment have also been investigated in this chapter; the results reveal that the computing efficiency can be increased through use of the normalized creep damage laws.

The author acknowledges that some important achievements and findings in this chapter have been reorganised and submitted to International Journal of Computational Materials Science.

Chapter 7 Conclusions and Future Work

This chapter summarizes the outcomes of this research and highlights the contributions in the relevant research topics, which were described in previous chapters. Future work relative to the development of in-house FE software HITSI for creep damage analysis are also discussed.

7.1 Contributions and Conclusions

This dissertation has documented the design and development of the in-house FE software HITSI for creep damage analysis. A novel in-house FE software prototype HITSI which allow the scientist to simulate the behaviour of creep damage in particular to analysis the evolution of creep damage in welds has provided. This research work can contribute to the computational creep damage mechanics in general and in particular to the structural design of components and the evolution of creep damage in weldment. It is perceived that the dissertation has made several contributions to the domain knowledge.

7.1.1 The Review of Computational FE software for Creep Damage Analysis

The first contribution of this project is a critical review of the current state of obtaining the computational capability for creep damage analysis. A brief overview and discussion on the problem domains relating to this project are presented. The mechanisms of creep deformation and creep fracture in metals and alloys are reviewed to understand the nature of the creep damage problem. The creep damage behaviour in weldment components has been identified. It also illustrates why this project needed to be done and why new techniques need to be involved. The current state of how computational capability is achieved for creep damage analysis and reasons to develop the in-house FE software have been demonstrated. It further reports on techniques such as CDM, FE algorithms, the OOP approach and numerical integration schemes that need to be involved in this project.

- The major advantages of the development and use of in-house FE software for creep damage analysis are presented. Three aspects are considered including the use of CDM, the removal of the failed element and the allowance for the stress redistribution due to tertiary creep and the multi-axial stress rupture criterion.

- Based on the CDM approach, a damage parameter is defined ranging from zero (no damage) to critical damage value (full damage) and is then controlled throughout the creep processes (primary, secondary and tertiary).

- Based on an explicit FE algorithm, the large storage demands can be reduced and the efficiency of computational capability can be improved in highly non-linear creep damage problem.

- Based on the OOP approach, the developer can simply create a new subroutine that inherits many of its features from existing subroutines making the program much easier to modify and maintain.

- Based on the Runge-Kutta scheme, improvement in stability and accuracy of results can be achieved; furthermore, the high efficiency is even more pronounced for large-scale problems in creep damage analysis.

7.1.2 The FEM in the Development of Computational Software for Creep Damage Analysis

The second contribution of this project is the outline of the use of FEM in the development of computational software for creep damage analysis. The general methodology consideration, the FE algorithm, the specific FE programming procedures and the existing standard FE subroutines that can be utilized in programming in-house FE software are stated.

- The fundamental requirement and the general methodology consideration in the development of in-house FE software for creep damage analysis are outlined; it further presents the FE algorithm involved with the creep damage constitutive equation, numerical integration method and explicit stress update algorithm for the development of HITSI.

- The specific FE methods such as the set of element data; the element stiffness assembly; the solution of equilibrium equation and results recovery at integrating points have been stated. Moreover, the relevant existing standard FE subroutines which can be utilized in the development of HITSI are reported to make the program work more efficiently.

7.1.3 Programming the FE Codes for in-house FE Software HITSI

The third contribution of this project is that the programming of the FE codes for the in-house FE software HITSI for creep damage analysis is presented. The general flow diagram and development strategy of the development of HITSI for creep damage analysis are proposed. The in-house FE software HITSI as developed involves the plane stress, plane strain, axisymmetric and three-dimensional version FE programs for creep damage problem. The use of Smith's standard FE library and Feng Tan's FE library is demonstrated in programming this software.

- The development of the linear elastic FE program contributes to specific FE techniques such as the input and initialisation, loop elements to find bandwidth and number of equations, element stiffness integration and assembly, equilibrium equation solution and stress recovery at the central gauss-point. The use of the relevant standard FE subroutines to achieve the computational capability for the linear elastic problem is also demonstrated.

- The development of the non-linear elastic-plastic FE program contributes to specific FE techniques such as how to add the load or displacement increment loop, execute the plastic iteration loop, check plastic convergence, update the gauss point stresses and compute the total body loads vector. The use of the relevant standard FE subroutines to achieve the computational capability for non-linear (material only and time independent) elastic-plastic problem is also demonstrated.

- The development of HITSI involves the plane stress, plane strain, axisymmetric and three-dimensional version FE programs for creep damage problem and contributes to specific FE techniques such as how to add the time increment loop, use of the creep damage constitutive equation, use of the numerical integration method, dealing with the stress redistribution, update of the stresses and creep damage field variables with the time integration, remove the failed element, dealing with the multi-material zones and different types (plane stress, plane strain, axisymmetric and three-dimensional) of problem. The use of the relevant standard FE subroutines to achieve the computational capability for creep damage analysis is also demonstrated.

7.1.4 Validation of the Finite Element Codes for in-house Software HITSI

The fourth contribution of this project is the validation of the FE codes of HITSI for creep damage analysis. A step by step validation according to the development strategy is proposed. The FE simulated results from HITSI (uni-axial case) are compared with the theoretical results to demonstrate the validity of the FE program. Adding to the domain knowledge, the FE codes of HITSI as validated involves the plane stress, plane strain, axisymmetric and three-dimensional version FE programs to satisfy the requirements of the development of HITSI for creep damage analysis.

- The validation of the linear elastic FE program contributes to ensure specific FE techniques, such as the read of the data information from FE model, the assembly of element stiffness matrix into global system, the integration of the gauss-point to find nodal coordinates and steering vector, the solution of the equilibrium equation and the recovery of stresses at central gauss-point, satisfy the requirements for the development of the in-house FE software.

- The validation of the non-linear elastic-plastic FE program contributes to ensure specific FE techniques, such as adding load or displacement increment loop, executing the plastic iteration loop, checking plastic convergence, updating the gauss point stresses and computing the total body loads vector, satify the requirements for the development of the in-house FE software.

- The validation of HITSI involved with the plane stress, plane strain, axisymmetric and three-dimensional version FE programs contributes to ensure specific FE techniques, such as how to add the time increment loop, use of the creep damage constitutive equation, use of the numerical integration method, dealing with the stress redistribution, update of the stresses and creep damage field variables with the time integration, remove the failed element, dealing with the multi-material zones and different types (plane stress, plane strain, axisymmetric and three-dimensional) of problem, satisfy the requirements for the development of the in-house FE software.

7.1.5 Benchmark Test of HITSI via the Numerical Investigation of Creep Damage Behaviour of a Steam Pipe Weldment Case

The fifth contribution of this project is the verification of HITSI via the numerical investigation of creep damage behaviour of a Cr-Mo-V steam pipe weldment case and

the investigation of the efficiency and accuracy of the integration algorithms (Euler and Runge-Kutta) and the normalized creep damage laws (Kachanov-Rabotnov creep damage constitutive equation).

- Verification of HITSI via the numerical investigation of creep damage behaviour of a steam pipe weldment case; the in-house software HITSI has been shown to predict reasonably well for the failure history of the pressure vessel weldment

- Investigation of the efficiency and accuracy of the numerical time integration algorithms (Euler and Runge-Kutta) in the FE method for creep damage analysis; the results reveal that the total computation time cost can be reduced significantly by the Runge-Kutta method in problems with a large set of system equations

- Investigation of the normalized creep damage laws (Kachanov-Rabotnov creep damage constitutive equation) in creep damage analysis of weldment; the results reveal that the computing efficiency can be increased through use of the normalized creep damage laws

7.2 Future Work

7.2.1 Disadvantages of the in-house FE Software HITSI

The computational FEM based CDM in-house software HITSI has been developed and applied for the analysis of creep deformation and damage; however, some disadvantages and limitations of this in-house software should be mentioned and outlined as follows:

1) Current FE codes provide limited element types and creep damage constitutive equations for the assessment of high temperature creep damage behaviour of weldment structures. In reality, weldment structure such as butt-welded pipework contains more complex structural features of a truly three-dimensional nature because of the existence of pipe intersections and branches. Therefore, the current research needs to be extended to cope with these features by, for example, the development of more complex element types such as tetrahedron elements.

2) Smith's standard subroutine FE library is utilized in the development of HITSI; however, the limitation imposed by using these subroutines is that the mesh generated by the subroutines should satisfy the order of node and freedom

numbering rule: the first node can be located at any corner, but subsequent corners and freedoms must follow in a clockwise sense. Furthermore, Smith's standard subroutine FE library was programmed based on the structured programming approach and in future the OOP approach should be considered.

3) The current version of the in-house FE software HITSI includes four main programs (plane stress, plane strain, axisymmetric and three-dimensional). The disadvantage of the use of HITSI is that user needs to define the analysis type for the problem; and then selects the relevant program for the analysis. This disadvantage increases complexity of interaction between the user and HITSI.

4) Current FE codes provide the computational capability for creep damage analysis; however, the calculation is performed for a continuum damage level from no damage to full damage and this requires a lot of storage on the computer. An output and restart control should be considered for reducing the storage requirement on the computer and increasing the computational efficiency.

7.2.2 Future Work

After the research work that has been done in this thesis, the author believes there are several ideas that should be taken forward:

1) The element types and creep damage constitutive equations in the assessment of high temperature creep damage behaviour of weldment structures should be extended to cope with more complex structural features and conditions.

2) The OOP approach has been considered in the development of HITSI; however, it not yet fully implemented in HITSI. Smith's standard subroutines can be reorganised and programmed based on the OOP approach.

3) The plane stress, plane strain, axisymmetric and three-dimensional version FE programs can be integrated into one program and an operation interface could be developed to increase the interaction between the user and this in-house FE software.

4) An output and restart control function should be developed for reducing the requirement of computer storage and increasing the computational efficiency for this in-house software.

5) The validation of the three-dimensional version FE program of HITSI has been conducted through a very simple uni-axial case; the data transfer interface between the FEMGV and three-dimensional version FE program of HITSI should be developed and the real three-dimensional (multi-axial) case should be used to validate the three-dimensional version FE program.

6) Butt-welded pipework may be subjected to more complicated loading and operating conditions that depend upon the location of the weldment in the pipework circuit; thus, it is important to develop the nodal loading calculator system for butt-welded pipework.

Reference

1. Altenbach, H., Kolarow, G., Morachkovsky, O. and Naumenko, K. (2000). On the accuracy of creep-damage predictions in thinwalled structures using the finite element method. *Computational mechanics, 25*(1), 87-98.

2. Archer, G. C. (1996). *Object-oriented finite element analysis.* (Doctoral dissertation, University of California at Berkeley).

3. Ashby, M. (1977). *Progress in the development of fracture mechanism maps.* Paper presented at the ICF4, Waterloo, Canada.

4. Ashby, M., Gandhi, C. and Taplin, D. (1979). Overview No. 3 Fracture-mechanism maps and their construction for fcc metals and alloys. *Acta Metallurgica, 27*(5), 699-729.

5. Ashby, M. F. (1972). A first report on deformation-mechanism maps. *Acta Metallurgica, 20*(7), 887-897.

6. Ashby, M. F. and Brown, L. M. (1983). *Perspectives in creep fracture*. Oxford and New York: Pergamon Press.

7. Bailey, R. (1935). The utilization of creep test data in engineering design. *Proceedings of The Institution of Mechanical Engineers, 131*(1), 131-349.

8. Barrett, C. and Nix, W. (1965). A model for steady state creep based on the motion of jogged screw dislocations. *Acta Metallurgica, 13*(12), 1247-1258.

9. Bauer, C. L. (1965). Polygonization of Rock Salt. *Trans. Metall. Soc. of AIME, 223*(4), 846-847.

10. Becker, A., Hyde, T., Sun, W. and Andersson, P. (2002). Benchmarks for finite element analysis of creep continuum damage mechanics. *Computational Materials Science, 25*(1), 34-41.

11. Becker, A., Hyde, T. and Xia, L. (1994). Numerical analysis of creep in components. *The Journal of Strain Analysis for Engineering Design, 29*(3), 185-192.

12. Bose, S. K. (2009). *Numeric computing in Fortran*. Oxford: Alpha Science International.

13. Briand, L. C. and Wieczorek, I. (2002). *Resource modeling in software engineering*. In: J. J. Marciniak (Ed.), Encyclopaedia of software engineering (2nd version). Chichester: John Wiley & Sons.

14. Cadek, J. (1988). *Creep in metallic materials (Materials Science Monographs)*. Michigan: Elsevier Science Ltd

15. Callister, W. D. (2001). *Fundamentals of Materials Science and Engineering: An Interactive eText*. New York: John Wiley & Sons.

16. Cao, J., Lin, J. and Dean, T. (2008). An implicit unitless error and step-size control method in integrating unified viscoplastic/creep ODE-type constitutive equations. *International journal for numerical methods in engineering, 73*(8), 1094-1112.

17. Chen, G. and Hsu, T. (1988). A mixed explicit-implicit (EI) algorithm for creep stress analysis. *International journal for numerical methods in engineering, 26*(2), 511-524.

18. Christiansen, J. (1970). Numerical solution of ordinary simultaneous differential equations of the 1st order using a method for automatic step change. *Numerische Mathematik, 14*(4), 317-324.

19. Coble, R. (1963). A model for boundary diffusion controlled creep in polycrystalline materials. *Journal of Applied Physics, 34*(6), 1679-1682.

20. Coleman, M. and Kimmins, S. (1990). *The behaviour of 1/2Crl/2Mol/4V pipe weldments in high temperature plant.* Paper presented at the Proc. Inst. Mech. Engrs. Conf. on Life of Welds at High Temperature, London, United Kingdom.

21. Coleman, M., Miller, D. and Stevens, R. (1998). *Reheat cracking and strategies to assure integrity of type 316 welded components.* Paper presented at the Integrity of High-Temperature Welds, International Conference, Nottingham, United Kingdom.

22. Coleman, M., Parker, J. and Walters, D. (1985). The behaviour of ferritic weldments in thick section 12Cr12Mo14V pipe at elevated temperature. *International Journal of Pressure Vessels and Piping, 18*(4), 277-310.

23. Conte, S. D., Dunsmore, H. E. and Shen, V. Y. (1986). *Software engineering metrics and models*. Redwood City: Benjamin-Cummings Publishing Co., Inc.

24. Cook, R. D. (2007). *Concepts and applications of finite element analysis*. New York: John Wiley & Sons.

25. Copenhaver, B. (1980). Jewish theologies of space in the scientific revolution: Henry More, Joseph Raphson, Isaac Newton and their predecessors. *Annals of Science, 37*(5), 489-548.

26. Cormeau, I. (1975). Numerical stability in quasi-static elasto/visco-plasticity. *International journal for numerical methods in engineering, 9*(1), 109-127.

27. Cottrell, A. and Jaswon, M. (1949). Distribution of solute atoms round a slow dislocation. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences, 199*(1056), 104-114.

28. Fierz, B., Spillmann, J. and Harders, M. (2011). *Element-wise mixed implicit-explicit integration for stable dynamic simulation of deformable objects.* Paper presented at the

Proceedings of the 2011 ACM Eurographics Symposium on Computer Animation, Vancouver, Canada.

29. Forde, B. W., Foschi, R. O. and Stiemer, S. F. (1990). Object-oriented finite element analysis. *Computers & Structures*, *34*(3), 355-374.

30. Frost, H. J. and Ashby, M. F. (1982). *Deformation mechanism maps: the plasticity and creep of metals and ceramics*. Oxford: Pergamon press.

31. Garofalo, F. (1965). *Fundamentals of creep and creep-rupture in metals*. London: Macmillan.

32. Gollapudi, S. (2007). *Creep mechanisms in titanium alloy tubing*. Michigan: ProQuest.

33. Gollapudi, S., Charit, I. and Murty, K. (2008). Creep mechanisms in Ti–3Al–2.5 V alloy tubing deformed under closed-end internal gas pressurization. *Acta Materialia, 56*(10), 2406-2419.

34. Gorash, Y., Altenbach, H. and Naumenko, K. (2008). Modeling of primary and secondary creep for a wide stress range. *PAMM, 8*(1), 10207-10208.

35. Goretta, K., Cruse, T., Koritala, R., Routbort, J., Mélendez-Martínez, J. and de Arellano-López, A. (2001). Compressive creep of polycrystalline $ZrSiO_4$. *Journal of the European Ceramic Society, 21*(8), 1055-1060.

36. Hagler, M. (1987). Spreadsheet solution of partial differential equations. *Education, IEEE Transactions on*(3), 130-134.

37. Hall, F. (1990). *Development of continuum damage mechanics models to predict the creep deformation and failure of high temperature structures.* (Doctoral dissertation, University of Manchester).

38. Hall, F. and Hayhurst, D. (1991). Continuum damage mechanics modelling of high temperature deformation and failure in a pipe weldment. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences, 433*(1888), 383-403.

39. Hall, F., Hayhurst, D. and Brown, P. (1996). Prediction of plane-strain creep-crack growth using continuum damage mechanics. *International Journal of Damage Mechanics, 5*(4), 353-383.

40. Harper, J. and Dorn, J. E. (1957). Viscous creep of aluminum near its melting temperature. *Acta Metallurgica, 5*(11), 654-665.

41. Hayhurst, C. J., Ranson, H. J., Gardner, D. J. and Birnbaum, N. K. (1995). Modelling of microparticle hypervelocity oblique impacts on thick targets. *International journal of impact engineering, 17*(1), 375-386.

42. Hayhurst, D. R. (1972). Creep rupture under multi-axial states of stress. *Journal of the Mechanics and Physics of Solids, 20*(6), 381-382.

43. Hayhurst, D. R., Brown, P. and Morrison, C. (1984). The role of continuum damage in creep crack growth. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 131-158.

44. Hayhurst, D. R., Hayhurst, R. J. and Vakili-Tahami, F. (2005). Continuum damage mechanics predictions of creep damage initiation and growth in ferritic steel weldments in a medium bore branched pipe under constant pressure at 590° C using a five-material weld model. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science, 461*(2060), 2303-2326.

45. Hayhurst, D. R. and Henderson, J. (1977). Creep stress redistribution in notched bars. *International Journal of Mechanical Sciences, 19*(3), 133-146.

46. Hayhurst, D. R. and Krzeczkowski, A. (1979). Numerical solution of creep problems. *Computer Methods in Applied Mechanics and Engineering, 20*(2), 151-171.

47. Hayhurst, D. R. (1973). Stress redistribution and rupture due to creep in a uniformly stretched thin plate containing a circular hole. *Journal of Applied Mechanics, 40*(1), 244-250.

48. Hayhurst, D. R., Dimmer, P. and Morrison, C. (1984). Development of continuum damage in the creep rupture of notched bars. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences, 311*(1516), 103-129.

49. Hayhurst, R. J. (2006). *Creep lifetime predictions of welded structures using parallel processing algorithms.* (Doctoral dissertation, University of Manchester).

50. Hayhurst, R. J., Vakili-Tahami, F. and Hayhurst, D. R. (2009). Verification of 3-D parallel CDM software for the analysis of creep failure in the HAZ region of Cr–Mo–V crosswelds. *International Journal of Pressure Vessels and Piping, 86*(8), 475-485.

51. Herring, C. (1950). Diffusional viscosity of a polycrystalline solid. *Journal of Applied Physics, 21*(5), 437-445.

52. Holdsworth, S. (2008). The European Creep Collaborative Committee (ECCC) approach to creep data assessment. *Journal of Pressure Vessel Technology, 130*(2), 024001.

53. Hyde, T. and Sun, W. (2002). Effect of bending load on the creep failure behaviour of a pressurised thick walled CrMoV pipe weldment. *International journal of pressure vessels and piping, 79*(5), 331-339.

54. Hyde, T., Sun, W. and Becker, A. (2000). Failure prediction for multi-material creep test specimens using a steady-state creep rupture stress. *International journal of mechanical sciences, 42*(3), 401-423.

55. Hyde, T., Sun, W. and Williams, J. (1999). Creep behaviour of parent, weld and HAZ materials of new, service-aged and repaired 1/2Cr1/2Mo1/4V: 2 1/4Cr1Mo pipe welds at 640 C. *Materials at high temperatures*, *16*(3), 117-129.

56. Hyde, T., Yehia, K. and Becker, A. (1993). Interpretation of impression creep data using a reference stress approach. *International journal of mechanical sciences*, *35*(6), 451-462.

57. James, M. L., Smith, G. M. and Wolford, J. (1985). *Applied numerical methods for digital computation* (Vol. 2). New York: Harper & Row.

58. Junhai, W. (2010). Numerical Simulation on Floor Heave Mechanism of Roadway. *Modern Mining*, *26*(12), 52-54.

59. Kachanov, L. (1958). Time of the rupture process under creep conditions. *Isv. Akad. Nauk. SSR. Otd Tekh. Nauk*, *8*, 26-31.

60. Kattan, P. I. and Voyiadjis, G. Z. (2002). *Damage mechanics with finite elements: practical applications with computer too*ls (Vol. 1). Berlin: Springer.

61. Keates, S., Clarkson, P. J., Harrison, L. A. and Robinson, P. (2000). *Towards a practical inclusive design approach.* Paper presented at the Proceedings on the 2000 conference on Universal Usability, Washington, United States.

62. Kern, T. U., Merckling, G. and Yagi, K. (2004). Introduction *Creep Properties of Heat Resistant Steels and Superalloys*: Berlin: Springer.

63. Kim, Y. J., Kim, J. S., Huh, N. S. and Kim, Y. J. (2002). Engineering C integral estimates for integral estimates for generalised creep behaviour and finite element validation. *International journal of pressure vessels and piping*, *79*(6), 427-443.

64. Kimmins, S., Walker, N. and Smith, D. (1996). Creep deformation and rupture of low alloy ferritic weldments under shear loading. *The Journal of Strain Analysis for Engineering Design*, *31*(2), 125-133.

65. Klenk, A., Schemmel, J. and Maile, K. (2003). Numerical modelling of ferritic welds and repair welds. *Numerical Modelling of Ferritic Welds and Repair Welds*. *2*(2), 1-14.

66. Krishnamohanrao, Y., Kutumbarao, V. and Rao, P. R. (1986). Fracture mechanism maps for titanium and its alloys. *Acta Metallurgica*, *34*(9), 1783-1806.

67. Krishnaswamy, P., Brust, F. and Ghadiali, N. (1995). A finite element algorithm to study creep cracks based on the creep hardening surface. *International journal for numerical methods in engineering*, *38*(6), 969-987.

68. Kun, F., Moreno, Y., Hidalgo, R. and Herrmann, H. (2003). Creep rupture has two universality classes. *EPL (Europhysics Letters)*, *63*(3), 347.

69. Lages, E. N., Paulino, G. H., Menezes, I. F. and Silva, R. R. (1999). Nonlinear finite element analysis using an object-oriented philosophy–Application to beam elements and to the Cosserat continuum. *Engineering with Computers*, *15*(1), 73-89.

70. Leckie, F. A. and Hayhurst, D. (1977). Constitutive equations for creep rupture. *Acta Metallurgica, 25*(9), 1059-1070.

71. Lee, S., Kim, B. and Lee, D. (1989). Fracture mechanism in coarse grained HAZ of HSLA steel welds. *Scripta metallurgica, 23*(6), 995-1000.

72. Lemaitre, J. (1972). Evaluation of dissipation and damage in metals submitted to dynamic loading. *Mechanical behavior of materials*, 540-549.

73. Lemaitre, J. (1985). Coupled elasto-plasticity and damage constitutive equations. *Computer Methods in Applied Mechanics and Engineering*, *51*(1), 31-49.

74. Lemaitre, J. and Desmorat, R. (2005). *Engineering damage mechanics: ductile, creep, fatigue and brittle failures*. Springer Science & Business Media.

75. Leonard, J. W. (1979). Newton-Raphson iterative method applied to circularly towed cable-body system. *Engineering Structures, 1*(2), 73-80.

76. Li, X. C. and Wu, S. X. (2008). Simulation of Early-age Concrete Creep Stress Based on ANSYS . *Journal of System Simulation, 20*(15), 3944-3947.

77. Ling, X., Tu, S. T. and Gong, J. M. (2000). Application of Runge–Kutta–Merson algorithm for creep damage analysis. *International journal of pressure vessels and piping, 77*(5), 243-248.

78. Liu, A. F. (2005). *Mechanics and mechanisms of fracture: an introduction*. Ohio: ASM International.

79. Liu, D. Z., Xu, Q. and Lu, Z. Y. (2013e). Research in the development of finite element software for creep damage analysis. *Journal of communication and computer, 10*(8), 1019-1030.

80. Liu, D. Z., Xu, Q., Lu, Z. Y., Barrans, S. and Glover, I. (2013c). *The development of finite element software for creep deformation and damage analysis of weldment*. Paper presented at the 6th International 'HIDA' Conference: Life/Defect Assessment & Failures in High Temperature Plant, Nagasaki, Japan.

81. Liu, D. Z., Xu, Q., Lu, Z. Y. and Xu, D. L. (2012b). *Research in the development of computational FE software for creep damage mechanics.* Paper presented at the 18th International Conference on Automation and Computing (ICAC), Loughborough, United Kingdom.

82. Liu, D. Z., Xu, Q., Lu, Z. Y., Xu, D. L. and Tan, F. (2013a). *The development of finite element analysis software for creep damage analysis.* Paper presented at the 2013 World Congress in Computer Science and Computer Engineering and Application, Las Vegas, United States.

83. Liu, D. Z., Xu, Q., Lu, Z. Y. and Xu, D. L. (2012a). The review of computational FE software for creep damage mechanics. *Advanced Materials Research, 510,* 495-499.

84. Liu, D. Z., Xu, Q., Lu, Z. Y., Xu, D. L. and Tan, F. (2013d). The validation of computational FE software for creep damage mechanics. *Advanced Materials Research, 744*, 205-210.

85. Liu, D. Z., Xu, Q., Lu, Z. Y., Xu, D. L. and Xu, Q. H. (2013b). The techniques in developing finite element software for creep damage analysis. *Advanced Materials Research, 744*, 199-204.

86. Machiels, L. and Deville, M. (1997). Fortran 90: an entry to object-oriented programming for the solution of partial differential equations. *ACM Transactions on Mathematical Software (TOMS), 23*(1), 32-49.

87. Mackie, R. I. (2008). *Programming distributed finite element analysis: an object oriented approach*. Stirling: Saxe-Coburg Publications.

88. Matsui, M., Tabuchi, M., Watanabe, T., Kubo, K., Kinugawa, J. and Abe, F. (2001). Degradation of creep strength in welded joint of 9% Cr steel. *ISIJ international, 41*, S126-S130.

89. Miller, G. R., & Rucki, M. D. (1993). *A program architecture for interactive nonlinear dynamic analysis of structures*. Paper presented at the ASCE Conference on Computing in Civil and Building Engineering, Anaheim, United States.

90. Moberg, F. (1995). *Implementation of constitutive equations for creep damage mechanics into the ABAQUS finite element code*. Sweden: SAQ Kontroll AB Certifiering.

91. Murakami, S. (1983). Notion of continuum damage mechanics and its application to anisotropic creep damage theory. *Journal of Engineering Materials and Technology, 105*(2), 99-105.

92. Murakami, S. and Liu, Y. (1995). Mesh-dependence in local approach to creep fracture. *International Journal of Damage Mechanics, 4*(3), 230-250.

93. Murti, K., and Sundaresan, S. (1985). Thermal Behavior of Austenitic-Ferritic Transition Joints Made by Friction Welding. *Welding Journal, 64*(12), S327-S334.

94. Mustata, R., Hayhurst, R., Hayhurst, D. and Vakili-Tahami, F. (2006). CDM predictions of creep damage initiation and growth in ferritic steel weldments in a medium-bore branched

pipe under constant pressure at 590 C using a four-material weld model. *Archive of Applied Mechanics, 75*(8-9), 475-495.

95. Nabarro, F. R. (1948). Report of a Conference on the Strength of Solids. *The Physical Society, London, 18*, 524.

96. Noels, L., Stainier, L. and Ponthot, J. P. (2004). Combined implicit/explicit time-integration algorithms for the numerical simulation of sheet metal forming. *Journal of Computational and Applied Mathematics, 168*(1), 331-339.

97. Norton, F. H. (1929). *The creep of steel at high temperatures*. New York: McGraw-Hill Book Company, Incorporated.

98. Oden, J. T. and Reddy, J. N. (2012). *An introduction to the mathematical theory of finite elements*. New York: Courier Corporation.

99. Odqvist, F. K. G. (1974). *Mathematical theory of creep and creep rupture*. Oxford: Clarendon Press Oxford.

100. Panait, C., Bendick, W., Fuchsmann, A., Gourgues-Lorenzon, A. F. and Besson, J. (2010). Study of the microstructure of the Grade 91 steel after more than 100,000 h of creep exposure at 600 °C. *International journal of pressure vessels and piping, 87*(6), 326-335.

101. Parker, J. and Parsons, A. (1995). High temperature deformation and fracture processes in 214Cr1Mo-12Cr12Mo14V weldments. *International journal of pressure vessels and piping, 63*(1), 45-54.

102. Parker, J. and Stratford, G. (1996). Strain localization in creep testing of samples with heterogeneous microstructures. *International journal of pressure vessels and piping, 68*(2), 135-143.

103. Penny, R. K. and Marriott, D. L. (1995). *Design for creep*. London: Chapman & Hall.

104. Perrin, I. and Hayhurst, D. (1996a). Creep constitutive equations for a 0.5 Cr–0.5 Mo–0.25 V ferritic steel in the temperature range 600–675 C. *The Journal of Strain Analysis for Engineering Design, 31*(4), 299-314.

105. Perrin, I. and Hayhurst, D. (1996b). A method for the transformation of creep constitutive equations. *International journal of pressure vessels and piping, 68*(3), 299-309.

106. Poirier, J. P. (1985). *Creep of crystals: high-temperature deformation processes in metals, ceramics and minerals*. Cambridge: Cambridge University Press.

107. Poirier, J. P. and Nicolas, A. (1976). *Crystalline plasticity and solid state flow in metamorphic rocks*. London: John Wiley & Sons.

108. Porter, D. A. and Easterling, K. E. (1992). *Phase Transformations in Metals and Alloys, (Revised Reprint)*. Florida: CRC press.

109. Rabotnov, Y. N. (1969). *Creep problems in structural members*. London: John Wiley & Sons.

110. Ralph, P. and Wand, Y. (2009). A proposal for a formal definition of the design concept. *Design requirements engineering: A ten-year perspective*, *14*, 103-136.

111. Rebelo, N., Nagtegaal, J., Taylor, L. and Passman, R. (1992). *Comparison of implicit and explicit finite element methods in the simulation of metal forming processes.* Paper presented at the ABAQUS Users Conf., Newport, Rhode Island.

112. Rehak, D. R. and Baugh, J. W. (1989). *Alternative programming techniques for finite element program development*. Paper presented at the IABSE Colloquium on Expert Systems in Civil Engineering, Bergamo, Italy.

113. Rice, J. R. and Thomson, R. (1974). Ductile versus brittle behaviour of crystals. *Philosophical magazine*, *29*(1), 73-97.

114. Riedel, H. (1987). *Fracture at high temperatures*. Berlin: Springer-Verlag.

115. Riedel, H. (1990). Creep crack growth under small-scale creep conditions. *International Journal of Fracture, 42*, 173-188.

116. Rollason, E. C. (1973). *Metallurgy for engineers*. London: Edward Arnold London.

117. Sasaki, S., Tateishi, M., Ishikawa, I. and Vanderwalt, P. (2005). *Case studies of reliability analysis by stochastic methodology in BGA creep analysis.* Paper presented at the 2005 International Symposium on Electronics Materials and Packaging, Tokyo, Japan.

118. Segle, P. (2002). Numerical simulation of weldment creep response (Doctoral dissertation, Department of Materials Science and Engineering Royal Institute of Technology, Swedish Institute for Metals Research Drottning Kristinas).

119. Smith, I. M. and Griffiths, D. V. (2005). *Programming the finite element method* (4th version). London: John Wiley & Sons.

120. Smith, I. M., Griffiths, D. V. and Margetts, L. (2013). *Programming the finite element method* (5th version).Oxford: Wiley-Blackwell.

121. Sun, J., Lee, K. and Lee, H. (2000). Comparison of implicit and explicit finite element methods for dynamic problems. *Journal of Materials Processing Technology*, *105*(1), 110-118.

122. Svensson, L.-E., & Dunlop, G. (1981). Growth of intergranular creep cavities. *International Metals Reviews*, *26*(1), 109-131.

123. Szabo, B. A. and Babuška, I. (1991). *Finite element analysis*. London: John Wiley & Sons.

124. Tu, S. T., Segle, P. and Gong, J. M. (2004). Creep damage and fracture of weldments at high temperature. *International journal of pressure vessels and piping, 81*(2), 199-209.

125. Tu, S. T., Wu, R. and Sandström, R. (1994). Design against creep failure for weldments in 0.5Cr0.5Mo0.25V pipe. *International journal of pressure vessels and piping, 58*(3), 345-354.

126. Venkatramani, G., Ghosh, S. and Mills, M. (2007). A size-dependent crystal plasticity finite-element model for creep and load shedding in polycrystalline titanium alloys. *Acta Materialia, 55*(11), 3971-3986.

127. Vignjevic, R., Campbell, J. C. and Lepage, S. (2004). *Numerical simulation of high velocity impacts on thin metallic targets I and II*. Paper presented at the 6th International Conference on Dynamics and Control of Systems and Structures in Space (DCSSS), Riomaggiore, Italy.

128. Viswanathan, G., Vasudevan, V. and Mills, M. (1999). Modification of the jogged-screw model for creep of γ-TiAl. *Acta Materialia, 47*(5), 1399-1411.

129. Viswanathan, R. (1989). *Damage mechanisms and life assessment of high temperature components*. Ohio: ASM international.

130. Wang, Z. and Hayhurst, D. (1994). The use of supercomputer modelling of high-temperature failure in pipe weldments to optimize weld and heat affected zone materials property selection. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences, 446*(1926), 127-148.

131. Weertman, J. (1955). Theory of Steady-State Creep Based on Dislocation Climb. *Journal of Applied Physics, 26*(10), 1213-1217.

132. Wilson, D. G. and Korakianitis, T. (2014). *The design of high-efficiency turbomachinery and gas turbines* (2nd version). Cambridge: MIT press.

133. Wong, M. T. (1999). *Three-dimensional finite element analysis of creep continuum damage growth and failure in weldments.* (Doctoral dissertation, University of Manchester).

134. Xu, Q. (2001). Creep damage constitutive equations for multi-axial states of stress for 0.5 Cr0. 5Mo0. 25V ferritic steel at 590 C. *Theoretical and applied fracture mechanics, 36*(2), 99-107.

135. Yao, H. T., Xuan, F. Z., Wang, Z. and Tu, S. T. (2007). A review of creep analysis and design under multi-axial stress states. *Nuclear Engineering and Design, 237*(18), 1969-1986.

136. Yu, T., Yatomi, M. and Shi, H. J. (2009). Numerical investigation on the creep damage induced by void growth in heat affected zone of weldments. *International Journal of Pressure Vessels and Piping*, *9*(86), 578-584.

137. Yuan, H. P., Zhu, L. G., Zhai, Y. J. and Chen, S. M. (2012). Numerical Test Study on the Mechanical Behavior of Rock Creep Fracture. *Applied Mechanics and Materials*, *204*, 526-533.

138. Zienkiewicz, O. and Cheung, Y. (1967). *The finite element method in structural and continuum mechanics*. New York: McGraw-Hill.

139. Zienkiewicz, O. and Cormeau, I. (1974). Visco-plasticity-plasticity and creep in elastic solids-a unified numerical solution approach. *International journal for numerical methods in engineering*, *8*(4), 821-845.

140. Zienkiewicz, O. C. and Taylor, R. L. (2000). *The finite element method: Solid mechanics* (Vol. 2). Oxford: Butterworth-heinemann.

141. Zienkiewicz, O. C. and Taylor, R. L. (2005). *The finite element method for solid and structural mechanics*. Oxford: Butterworth-heinemann.

142. Zimmermann, T., Dubois, Y. and Bomme, P. (1992). Object-oriented finite element programming: I. Governing principles. *Computer methods in applied mechanics and engineering*, *98*(2), 291-303.

143. Zolochevsky, A., Sklepus, S., Hyde, T., Becker, A. and Peravali, S. (2009). Numerical modeling of creep and creep damage in thin plates of arbitrary shape from materials with different behavior in tension and compression under plane stress conditions. *International journal for numerical methods in engineering*, *80*(11), 1406-1436.

Appendix A: Source Codes of the Main Program for FE Software HITSI

```
!< Source codes of the main program for FE software HITSI
!>-----------------------------------------------------------------------------------------------------
!< The in-house FE software HITSI has been developed and the current version
!< includes four main programs (plane stress, plane strain, axisymmetric and
!<three-dimensional)
!>-----------------------------------------------------------------------------------------------------
!< The existing standard FE subroutines from I M Smith's book: Programming
!< the Finite Element Method are modified and utilized in HITSI for the spatial
!< discretisation by finite elements, element stiffness integration and assembly,
!<  solution of equilibrium equation and recover results at integrating points.
!>-----------------------------------------------------------------------------------------------------
!< The specific subroutine library is provided by Feng Tan for HITSI and it contains
!< the creep damage constitutive equation's subroutine, the time integration's
!< subroutine, a nodal force calculator for axisymmetric FE program and a
!< data transfer interface between the in-house FE software HITSI and the
!< pre- and post-processor FE software FEMGV.
!>-----------------------------------------------------------------------------------------------------
program planestress
!>-------------------------------------------introdution-----------------------------------------------
!< This  main program is developed for solving plane stress creep damage problem
use new_library;  use  geometry_lib;  use lib_add;  implicit none
99998 format(1X,I4)
!>------------------------ --------The declaration of variables  ----------------------------------
integer::    nels,neq,nband,nn,nr,nip,nodof,nod,nst,ndof,oppo,              &
              i,k,iel,ndim,loaded_nodes,nprops,np_types,iy,j,ix,             &
             iters,ii,ij,key1=1,key2=2,key3=3,key4=4,key5=5,key6=6,         &
```

```
        key7=7,key8=8,key9=9,key10=10,key11=9999
 doubleprecision::   ESS, MPSS, T, t0, e,v,det
!>---------------------------------- The declaration of arrays  ----------------------------------
 doubleprecision, dimension (5)::ABV,crate
 character(len=15) :: element
!---------------------------------------- dynamic arrays-----------------------------------------
 doubleprecision    ,allocatable :: g_coord(:,:),points(:,:),weights(:),kv(:),          &
                 km(:,:),dee(:,:),fun(:),der(:,:),jac(:,:),deriv(:,:),tabv(:,:,:),       &
                 bee(:,:),coord(:,:),loads(:),eld(:),sigma(:), tevp(:,:,:),              &
                 prop(:,:), eps(:), evp(:),devp(:),  teps(:,:,:)                         &
                 bload(:),eload(:),evpt(:,:,:),bdylds(:),                                &
                 material(:),storkv(:,:),tsigma(:,:,:),                                  &
                 tdevp(:,:,:),gc(:),tgc(:,:,:),
 integer, allocatable :: g_num(:,:) ,nf(:,:), g(:)  , num(:)  ,  g_g(:,:),              &
                 etype(:), no(:)
!-----------------------------------input and initialisation-------------------------------------
  open (10,file='p1.dat',status='old',    action='read')
  open (11,file='p1.res',status='replace',action='write')
  read (10,*) element,nels,nn,nip,nodof,nod,nst,ndim,oppo
  ndof=nod*nodof
  allocate ( g_coord(ndim,nn),g_num(nod,nels),nf(nodof,nn), g_g(ndof,nels),    &
         g(ndof),points(nip,ndim),weights(nip),coord(nod,ndim),fun(nod),       &
         jac(ndim,ndim),dee(nst,nst),der(ndim,nod),deriv(ndim,nod),            &
        num(nod),km(ndof,ndof), eld(ndof),  sigma(nst),etype(nels),            &
        eps(nst), evp(nst), devp(nst), bload(ndof),eload(ndof),                &
        evpt(nst,nip,nels), tabv(5,nip,nels), material(oppo),                  &
        tsigma(nst,nip,nels),tevp(nst,nip,nels), bee(nst,ndof),                &
        tdevp(nst,nip,nels),gc(ndim),tgc(ndim,nip,nels),                       &
        teps(nst,nip,nels))
 read(10,*) nprops , np_types
 allocate(prop(nprops,np_types)) ; read(10,*) prop
```

```
      etype = 1 ; if(np_types>1) read(10,*) etype
        do i=1, nn
           read (10,*)k,g_coord(:,i)
        end do
        do i=1, nels
           read (10,*)k, g_num(:,i)
        end do
  nf=1; read(10,*) nr ; if(nr>0) read(10,*)(k,nf(:,k),i=1,nr)
  call formnf (nf);neq=maxval(nf); nband = 0
!-----------------------loop the elements to find bandwidth and neq----------------------
    elements_1   : do iel =1,nels
              num=g_num(:,iel)
                 call num_to_g ( num , nf , g );
                   g_g(:,iel)=g
                 e=prop(1,etype(iel)); v=prop(2,etype(iel))
            if(nband<bandwidth(g))nband=bandwidth(g)
   end do elements_1
  dee=.0; dee(1,1)=e/(1.-v*v);dee(2,2)=dee(1,1);dee(3,3)=.5*e/(1.+v)
  dee(1,2)=v*dee(1,1);dee(2,1)=dee(1,2)
  call sample(element,points,weights)
  allocate( kv(neq*(nband+1)),loads(0:neq),bdylds(0:neq)); kv=0.0
!---------------------- element stiffness integration and assembly--------------------------
 elements_2: do iel = 1 , nels
          num = g_num(:, iel);    g = g_g( : , iel )
          coord = transpose(g_coord(:, num)) ;      km=0.0
        gauss_pts_1: do i = 1 , nip
           call shape_fun(fun,points,i)
            call shape_der(der,points,i) ; jac = matmul(der,coord)
            det = determinant(jac); call invert(jac)
            gc=matmul(fun,coord);  tgc(:,i,iel)=gc
```

```
          deriv = matmul(jac,der) ; call beemat (bee,deriv)
          km = km + matmul(matmul(transpose(bee),dee),bee) *det* weights(i)
       end do gauss_pts_1
    call formkv (kv,km,g,neq)
 end do elements_2
!------------------------ solution of equilibrium equation---------------------------------
 bdylds=.0; evpt(1,nip,nels)=0.0;  evpt(2,nip,nels)=0.0
 evpt(3,nip,nels)=0.0;  evpt(4,nip,nels)=0.0
read (10,*) loaded_nodes;allocate(no(loaded_nodes),storkv(loaded_nodes,ndim))
read (10,*)(no(i),storkv(i,:),i=1,loaded_nodes)
   call banred(kv,neq)
       T=1;  t0=0
         do i=1,nels; do j=1,nip;  do k=1,5
                tabv(k,j,i)=0
          end do; end do; end do
  tsigma=0; tevp=0; tdevp=0
   do ii=1,23774; ij=ii*iy; do iy=0,1
     t0=t0+t; iters=0;bdylds=0;evpt=0
      do i=1, loaded_nodes; loads(nf(:,no(i)))=storkv(i,:);end do
      loads = loads + bdylds
 call bacsub(kv,loads)
!---------------------------- recover initial elastic stress---------------------------------
elements_3:do iel=1,nels; bload=0
          num = g_num(:,iel) ; coord = transpose(g_coord(:,num))
          g = g_g( : , iel)  ;     eld=loads(g)
         integrating_pts_2: do i = 1 , nip
          call shape_fun(fun,points,i); call shape_der(der,points,i)
          jac=matmul(der,coord); call invert(jac)
         deriv=matmul(jac,der); call beemat(bee,deriv)
         eps=matmul(bee,eld); teps(:,i,iel)=eps
         det = determinant(jac); eps=eps-evpt(:,i,iel)
```

```
      sigma=matmul(dee,eps); tsigma(:,i,iel)=sigma
!---------- creep damage variables and stress updating ----------
abv=tabv(:,i,iel); call rdmpes (sigma,mpss,ess)
do ix=1, oppo; material(ix)=prop(ix+2,etype(iel));  end do
  call EULER_KR (abv,crate,t,t0,sigma,ess,mpss,material)
   tabv(:,i,iel)=abv; evp(1)=crate(1)*t;evp(2)=crate(2)*t;
   evp(3)=crate(3)*2*t; evp(4)=crate(4)*t; tevp(:,i,iel)=evp
evpt(:,i,iel)=evpt(:,i,iel)+evp
   devp=matmul(dee,evp); tdevp(:,i,iel)=devp
      eload=matmul(devp,bee)
            bload=bload+eload*det*weights(i)
   if(tabv(5,i,iel)>=0.99)then
          tabv(5,i,iel)=0.99; tevp(:,i,iel)=0.0; km=0.0
              else
    tabv(:,i,iel)=abv; tevp(:,i,iel)=evp
           evpt(:,i,iel)=evpt(:,i,iel)+evp
             end if
    end do integrating_pts_2
  bdylds( g ) = bdylds( g )+ bload     ; bdylds(0) = 0
 end do elements_3
end do; end do
!---------- output of all calculated results for post-processing----------
write(11,99998) key1; write(11,*)ndim,nn,nod,nels,element,nst,nip,t0,e,v
write(11,99998) key2;  do k=1,nn; write(11,*) k ,g_coord(:,k);end do
write(11,99998) key3;  do k = 1 , nels; write(11,*) k ,g_num(:,k), key1; end do
write(11,99998) key4; do k=1,nn; write(11,*) k,loads(nf(:,k)); end do
write(11,99998) key5; do k=1,nn; write(11,*) k,bdylds(nf(:,k)); end do
write(11,99998) key7;  do i=1,nels; write(11,*) i
       do j=1,nip; write(11,*) j, tsigma(:,j,i); end do;  end do
write(11,99998) key8; do i=1,nels; write(11,*) i
       do j=1,nip; write(11,*)  j, teps(:,j,i); end do; end do
```

```
write(11,99998) key9; do i=1,nels;  write(11,*) i
      do j=1,nip; write(11,*)  j, evpt(:,j,i);  end do; end do
write(11,99998) key10;  do i=1,nels; write(11,*) i
     do j=1,nip; write(11,*)  j, tabv(5,j,i); end do;  end do
 write(11,99998) key1
 write(11,*)ndim,nn,nod,nels,element,nst,nip,t0,e,v, key1
 write(11,99998) key2
 do k=1,nn; write(11,*) k ,g_coord(:,k);end do
 write(11,99998) key3
 do k = 1 , nels; write(11,*) k ,g_num(:,k), key1; end do
 write(11,99998) key11
end program planestress

!>-----------------------------------------------------------------------------------------------
program planestrain
!>-----------------------------------------introdution-----------------------------------------
!< This  main program is developed for solving plane strain creep damage problem
use new_library   ;   use  geometry_lib  ; use lib_add;     implicit none
99998 format(1X,I4)
 integer::    nels,neq,nn,nr,nip,nodof,nod,nst,ndof,oppo,                    &
          i,k,iel,ndim,loaded_nodes,nprops,np_types,iy,j,ix,                 &
         iters,ii,ij,key1=1,key2=2,key3=3,key4=4,key5=5,key6=6,             &
         key7=7,key8=8,key9=9,key10=10,key11=9999
 doubleprecision::   ESS, MPSS, T, t0, e,v,det
doubleprecision, dimension (5)::ABV,crate
 character(len=15) :: element
!---------------------------------------- dynamic arrays-----------------------------------------
 doubleprecision   ,allocatable :: g_coord(:,:),points(:,:),weights(:),kv(:),      &
                km(:,:),dee(:,:),fun(:),der(:,:),jac(:,:),deriv(:,:),              &
                bee(:,:),coord(:,:),loads(:),eld(:),sigma(:),                      &
                prop(:,:), eps(:), evp(:),devp(:),                                 &
```

```
                bload(:),eload(:),evpt(:,:,:),bdylds(:),tabv(:,:,:),          &
                material(:),storkv(:,:),tsigma(:,:,:),tevp(:,:,:),            &
                tdevp(:,:,:),gc(:),tgc(:,:,:),teps(:,:,:)
 integer, allocatable :: g_num(:,:) ,nf(:,:), g(:)  , num(:)  ,  g_g(:,:),    &
               etype(:), no(:),kdiag(:)
!--------------------------------------input and initialisation---------------------------------------
 open (10,file='p2.dat',status='old',    action='read')
 open (11,file='p2.res',status='replace',action='write')
 read (10,*) element,nels,nn,nip,nodof,nod,nst,ndim,oppo
 ndof=nod*nodof
 allocate ( g_coord(ndim,nn),g_num(nod,nels),nf(nodof,nn), g_g(ndof,nels),  &
       g(ndof),points(nip,ndim),weights(nip),coord(nod,ndim),fun(nod),      &
       jac(ndim,ndim),dee(nst,nst),der(ndim,nod),deriv(ndim,nod),           &
       num(nod),km(ndof,ndof), eld(ndof),  sigma(nst),etype(nels),          &
       eps(nst), evp(nst), devp(nst), bload(ndof),eload(ndof),              &
       evpt(nst,nip,nels), tabv(5,nip,nels), material(oppo),                &
       tsigma(nst,nip,nels),tevp(nst,nip,nels), bee(nst,ndof),              &
       tdevp(nst,nip,nels),gc(ndim),tgc(ndim,nip,nels),                     &
       teps(nst,nip,nels))
  read(10,*) nprops , np_types
  allocate(prop(nprops,np_types)) ; read(10,*) prop
   etype = 1 ; if(np_types>1) read(10,*) etype
       do i=1, nn
            read (10,*)k,g_coord(:,i)
       end do
       do i=1, nels
            read (10,*)k, g_num(:,i)
       end do
 nf=1; read(10,*) nr ; if(nr>0) read(10,*)(k,nf(:,k),i=1,nr)
```

```
call formnf (nf);neq=maxval(nf); allocate(kdiag(neq))    ; kdiag = 0
!------------------------loop the elements to find bandwidth and neq----------------------
   elements_1   : do iel =1,nels
                num=g_num(:,iel)
                   call num_to_g ( num , nf , g );
                       g_g(:,iel)=g
                    call fkdiag(kdiag,g)
       end do elements_1
        kdiag(1)=1; do i=2,neq; kdiag(i)=kdiag(i)+kdiag(i-1); end do
            call sample(element,points,weights)
     allocate( kv(kdiag(neq)),loads(0:neq),bdylds(0:neq)); kv=0.0
     call sample(element,points,weights)
!----------------------- element stiffness integration and assembly-------------------------
 elements_2: do iel = 1 , nels
          num = g_num(:, iel);    g = g_g( : , iel )
          coord = transpose(g_coord(:, num)) ;      km=0.0
       gauss_pts_1: do i = 1 , nip
           e=prop(1,etype(iel)); v=prop(2,etype(iel)); call deemat(dee,e,v)
           call shape_fun(fun,points,i);   call shape_der(der,points,i)
            jac = matmul(der,coord);     det = determinant(jac)
            call invert(jac);  gc=matmul(fun,coord)
              tgc(:,i,iel)=gc
            deriv = matmul(jac,der) ; call beemat (bee,deriv)
         km = km + matmul(matmul(transpose(bee),dee),bee) *det* weights(i)
       end do gauss_pts_1
   call fsparv (kv,km,g,kdiag)
 end do elements_2
!----------------------- solution of equilibrium equation---------------------------------
 bdylds=.0;   evpt(1,nip,nels)=0.0;  evpt(2,nip,nels)=0.0
 evpt(3,nip,nels)=0.0;  evpt(4,nip,nels)=0.0
```

```
read (10,*) loaded_nodes;allocate(no(loaded_nodes),storkv(loaded_nodes,ndim))
 read (10,*)(no(i),storkv(i,:),i=1,loaded_nodes)
    call sparin (kv,kdiag)
     T=1;  t0=0
          do i=1,nels; do j=1,nip; do k=1,5
                tabv(k,j,i)=0
           end do; end do; end do
             tsigma=0; tevp=0; tdevp=0
           do ii=0,7038;   ij=ii*iy; do iy=0,0;   t0=t0+t
             iters=0;bdylds=0;evpt=0
        do i=1, loaded_nodes; loads(nf(:,no(i)))=storkv(i,:);end do
       loads = loads + bdylds
    call spabac(kv,loads,kdiag)
!---------------------------- recover initial elastic stress----------------------------------
elements_3:do iel=1,nels;     bload=0
           num = g_num(:,iel) ; coord = transpose(g_coord(:,num))
           g = g_g( : , iel)  ;     eld=loads(g)
         integrating_pts_2: do i = 1 , nip
           call shape_fun(fun,points,i);  call shape_der(der,points,i)
            jac=matmul(der,coord); det = determinant(jac)
              call invert(jac);     deriv=matmul(jac,der)
             call beemat(bee,deriv)  ; eps=matmul(bee,eld)
            eps=eps-evpt(:,i,iel); sigma=matmul(dee,eps)
           tsigma(:,i,iel)=sigma
!-------------------- creep damage variables and stress updating ----------------------
       abv=tabv(:,i,iel); call rdmpes (sigma,mpss,ess)
       do ix=1, oppo; material(ix)=prop(ix+2,etype(iel)); end do
        call EULER_KR (abv,crate,t,t0,sigma,ess,mpss,material)
         tabv(:,i,iel)=abv; evp(1)=crate(1)*t;evp(2)=crate(2)*t
```

```
            evp(3)=crate(3)*2*t; evp(4)=crate(4)*t
            tevp(:,i,iel)=evp; evpt(:,i,iel)=evpt(:,i,iel)+evp
              devp=matmul(dee,evp); tdevp(:,i,iel)=devp
              eload=matmul(devp,bee)
                        bload=bload+eload*det*weights(i)
   if(tabv(5,i,iel)>=0.99)then
            tabv(5,i,iel)=0.99; tevp(:,i,iel)=0.0; km=0.0
                else
    tabv(:,i,iel)=abv; tevp(:,i,iel)=evp
             evpt(:,i,iel)=evpt(:,i,iel)+evp
               end if
     end do integrating_pts_2
  bdylds( g ) = bdylds( g )+ bload      ; bdylds(0) = 0
 end do elements_3
end do; end do
!----------------- output of all calculated results for post-processing----------
write(11,99998) key1; write(11,*)ndim,nn,nod,nels,element,nst,nip,t0,e,v
write(11,99998) key2;  do k=1,nn; write(11,*) k ,g_coord(:,k);end do
write(11,99998) key3;  do k = 1 , nels; write(11,*) k ,g_num(:,k), key1; end do
write(11,99998) key4; do k=1,nn; write(11,*) k,loads(nf(:,k)); end do
write(11,99998) key5; do k=1,nn; write(11,*) k,bdylds(nf(:,k)); end do
write(11,99998) key7;  do i=1,nels; write(11,*) i
        do j=1,nip; write(11,*) j, tsigma(:,j,i); end do;  end do
write(11,99998) key8; do i=1,nels; write(11,*) i
        do j=1,nip; write(11,*)  j, teps(:,j,i); end do; end do
write(11,99998) key9; do i=1,nels;  write(11,*) i
        do j=1,nip; write(11,*)  j, evpt(:,j,i);  end do; end do
write(11,99998) key10;  do i=1,nels; write(11,*) i
       do j=1,nip; write(11,*)  j, tabv(5,j,i); end do;  end do
 write(11,99998) key1
 write(11,*)ndim,nn,nod,nels,element,nst,nip,t0,e,v, key1
```

```
 write(11,99998) key2
 do k=1,nn; write(11,*) k ,g_coord(:,k);end do
 write(11,99998) key3
 do k = 1 , nels; write(11,*) k ,g_num(:,k), key1; end do
 write(11,99998) key11
end program planestrain

!>-----------------------------------------------------------------------------------------------
program axisy
!>-------------------------------------------introdution-----------------------------------------
!< This  main program is developed for solving axisymmetric creep damage problem
 use new_library  ;  use  geometry_lib  ; use lib_add; implicit none
 99998 format(1X,I4)
 integer::    nels,neq,nband,nn,nr,nip,nodof,nod,nst,ndof,oppo,                  &
          i,k,iel,ndim,loaded_nodes,nprops,np_types,iy,j,ix,                     &
         iters,ii,ij,key1=1,key2=2,key3=3,key4=4,key5=5,key6=6,                 &
         key7=7,key8=8,key9=9,key10=10,key11=9999
 doubleprecision::   ESS, MPSS, T, t0, e,v,det ,radius
doubleprecision, dimension (5)::ABV,crate
doubleprecision, dimension (4)::S
 character(len=15) :: element
!----------------------------- --------- dynamic arrays----------------------------------------
 doubleprecision   ,allocatable :: g_coord(:,:),points(:,:),weights(:),kv(:),       &
                km(:,:),dee(:,:),fun(:),der(:,:),jac(:,:),deriv(:,:),               &
                bee(:,:),coord(:,:),loads(:),eld(:),sigma(:),                       &
                prop(:,:), eps(:), evp(:),devp(:), bload(:),                        &
                eload(:),evpt(:,:,:),bdylds(:),tabv(:,:,:),                         &
                material(:),storkv(:,:),tsigma(:,:,:),                              &
                tevp(:,:,:),  tdevp(:,:,:),gc(:),                                   &
                tgc(:,:,:),teps(:,:,:)
```

```
 integer, allocatable :: g_num(:,:) ,nf(:,:), g(:)  , num(:)  ,  g_g(:,:),                  &
                         etype(:), no(:)
!---------------------------------input and initialisation-------------------------------------
 open (10,file='p3.dat',status=    'old',action='read')
 open (11,file='p3.res',status='replace',action='write')
 read (10,*) element,nels,nn,nip,nodof,nod,nst,ndim,oppo
 ndof=nod*nodof
 allocate ( g_coord(ndim,nn),g_num(nod,nels),nf(nodof,nn), g_g(ndof,nels), &
       g(ndof),points(nip,ndim),weights(nip),coord(nod,ndim),fun(nod),      &
       jac(ndim,ndim),dee(nst,nst),der(ndim,nod),deriv(ndim,nod),           &
       num(nod),km(ndof,ndof), eld(ndof),  sigma(nst),etype(nels),          &
       eps(nst), evp(nst), devp(nst), bload(ndof),eload(ndof),              &
       evpt(nst,nip,nels), tabv(5,nip,nels), material(oppo),                &
       tsigma(nst,nip,nels),tevp(nst,nip,nels), bee(nst,ndof),              &
       tdevp(nst,nip,nels),gc(ndim),tgc(ndim,nip,nels),                     &
       teps(nst,nip,nels))
   read(10,*) nprops , np_types
   allocate(prop(nprops,np_types)) ; read(10,*) prop
    etype = 1 ; if(np_types>1) read(10,*) etype
        do i=1, nn
              read (10,*)k,g_coord(:,i)
        end do
       do i=1, nels
              read (10,*)k, g_num(:,i)
       end do
   nf=1; read(10,*) nr ;if(nr>0) read(10,*) (k,nf(:,k),i=1,nr)
  call formnf(nf); neq=maxval(nf)
!---------------- loop the elements to find nband and set up global arrays ------------
    nband=0
    elements_1   : do iel =1,nels
             num=g_num(:,iel); call num_to_g ( num , nf , g ); g_g(:,iel)=g
```

```
        if(nband<bandwidth(g))nband=bandwidth(g)
       end do elements_1
        call sample(element,points,weights)
   allocate( kv(neq*(nband+1)),loads(0:neq),bdylds(0:neq)); kv=0.0
!-------------------- element stiffness integration and assembly-----------------------
 elements_2: do iel=1,nels
        num=g_num(:,iel)  ; coord =transpose( g_coord(:,num))
        g = g_g(: ,iel)   ;    km=0.0
        e=prop(1,etype(iel)); v=prop(2,etype(iel))
        call deemat(dee,e,v);  do ix=1, oppo
        material(ix)=prop(ix+2,etype(iel)); end do
    integrating_pts_1:  do i=1,nip
        call shape_fun(fun,points,i)  ;  call shape_der(der,points,i)
        jac=matmul(der,coord)   ;   det= determinant(jac)
        call invert(jac); gc=matmul(fun,coord)
          tgc(:,i,iel)=gc;  deriv = matmul(jac,der)
          call bmataxi(bee,radius,coord,deriv,fun)
            det =det*radius
       km= km+matmul(matmul(transpose(bee),dee),bee)*det*weights(i)
      end do integrating_pts_1
     call formkv (kv,km,g,neq)
  end do elements_2
!------------------------ solution of equilibrium equation---------------------------------
  read (10,*) loaded_nodes;allocate(no(loaded_nodes),storkv(loaded_nodes,ndim))
  read (10,*)(no(i),storkv(i,:),i=1,loaded_nodes)
    call banred(kv,neq)
    bdylds=.0;  T=1;  t0=0
      do i=1,nels; do j=1,nip; do k=1,5
            tabv(k,j,i)=0
       end do; end do; end do
```

```
            tsigma=0; tevp=0; tdevp=0
       do ii=0,10692;  ij=ii*iy;  do iy=0,0;   t0=t0+t
          iters=0;bdylds=0;evpt=0
        do i=1, loaded_nodes; loads(nf(:,no(i)))=storkv(i,:);end do
       loads = loads + bdylds
    call bacsub(kv,loads)
!---------------------------- recover initial elastic stress----------------------------------
 elements_3:  do iel=1, nels;  bload=0
          num = g_num(:,iel) ; coord = transpose(g_coord(:,num))
          g = g_g( : , iel)  ;     eld=loads(g)
         integrating_pts_2: do i = 1 , nip
          call shape_fun(fun,points,i);   call shape_der(der,points,i)
          jac=matmul(der,coord);     call invert(jac)
            deriv=matmul(jac,der)
              call bmataxi(bee,radius,coord,deriv,fun)
              eps=matmul(bee,eld);   teps(:,i,iel)=eps
           det=det*radius;   eps=eps-evpt(:,i,iel)
        sigma=matmul(dee,eps); tsigma(:,i,iel)=sigma
!-------------------- creep damage variables and stress updating ----------------------
   abv=tabv(:,i,iel); do ix=1, oppo
   material(ix)=prop(ix+2,etype(iel));  end do
     call STRESS_DEVIATOR_2D (sigma,S)
       call equivalent_stress_2D (S,ESS)
         call  max_PRINCIPAL_STRESS_2D (sigma,MPSS)
     call EULER_KR (abv,crate,t,t0,sigma,ess,mpss,material)
            if(tabv(5,i,iel)>=0.99)then
            tabv(5,i,iel)=0.99; tevp(:,i,iel)=0.0; km=0.0
                else
    tabv(:,i,iel)=abv; tevp(:,i,iel)=evp
             evpt(:,i,iel)=evpt(:,i,iel)+evp
               end if
```

```
    devp=matmul(dee,evp);  tdevp(:,i,iel)=devp
         eload=matmul(devp,bee)
        bload=bload+eload*det*weights(i)
      end do integrating_pts_2
  bdylds( g ) = bdylds( g )+ bload     ; bdylds(0) = 0
 end do elements_3
end do; end do
!----------------- output of all calculated results for post-processing----------
write(11,99998) key1; write(11,*)ndim,nn,nod,nels,element,nst,nip,t0,e,v
write(11,99998) key2;  do k=1,nn; write(11,*) k ,g_coord(:,k);end do
write(11,99998) key3;  do k = 1 , nels; write(11,*) k ,g_num(:,k), key1; end do
write(11,99998) key4; do k=1,nn; write(11,*) k,loads(nf(:,k)); end do
write(11,99998) key5; do k=1,nn; write(11,*) k,bdylds(nf(:,k)); end do
write(11,99998) key7;  do i=1,nels; write(11,*) i
        do j=1,nip; write(11,*) j, tsigma(:,j,i); end do;  end do
write(11,99998) key8; do i=1,nels; write(11,*) i
        do j=1,nip; write(11,*)  j, teps(:,j,i); end do; end do
write(11,99998) key9; do i=1,nels;  write(11,*) i
        do j=1,nip; write(11,*)  j, evpt(:,j,i);  end do; end do
write(11,99998) key10;  do i=1,nels; write(11,*) i
       do j=1,nip; write(11,*)  j, tabv(5,j,i); end do;  end do
 write(11,99998) key1
 write(11,*)ndim,nn,nod,nels,element,nst,nip,t0,e,v, key1
 write(11,99998) key2
 do k=1,nn; write(11,*) k ,g_coord(:,k);end do
 write(11,99998) key3
 do k = 1 , nels; write(11,*) k ,g_num(:,k), key1; end do
 write(11,99998) key11
end program axisy

!>-------------------------------------------------------------------------------------------------
```

```
program D3
!>----------------------------------------introdution----------------------------------------
!< This main program is developed for solving three-dimensional creep damage problem
use new_library  ;  use  geometry_lib  ; use lib_add;    implicit none
99998 format(1X,I4)
 integer::    nels,neq,nn,nr,nip,nodof,nod,nst,ndof,oppo,                    &
          i,k,iel,ndim,loaded_nodes,nprops,np_types,iy,j,ix,                 &
          iters,ii,ij,fixed_nodes,key1=1,key2=2,key3=3,key4=4,               &
         key5=5,key6=6, key7=7,key8=8,key9=9,key10=10,key11=9999
doubleprecision::   ESS, MPSS, T, t0, e,v,det
doubleprecision, dimension (7):: ABV,crate
 character(len=15) :: element
!---------------------------------------- dynamic arrays----------------------------------------
 doubleprecision    ,allocatable :: g_coord(:,:),points(:,:),weights(:),kv(:),      &
                   km(:,:),dee(:,:),fun(:),der(:,:),jac(:,:),deriv(:,:),           &
                   bee(:,:),coord(:,:),loads(:),eld(:),sigma(:),                   &
                   prop(:,:), eps(:), evp(:),devp(:), bload(:),                    &
                  eload(:),evpt(:,:,:),bdylds(:),tabv(:,:,:),                      &
                  material(:),storkv(:),tsigma(:,:,:),value(:) ,                   &
                 tdevp(:,:,:),load_store(:), ,gc(:),tevp(:,:,:)
                 tgc(:,:,:),teps(:,:,:)
 integer, allocatable :: g_num(:,:) ,nf(:,:), g(:)  , num(:)  ,  g_g(:,:),        &
                 etype(:), no(:),kdiag(:),sense(:), node(:)
!----------------------------------input and initialisation----------------------------------------
  open (10,file='p4.dat',status=    'old',action='read')
  open (11,file='p4.res',status='replace',action='write')
  read (10,*) element,nels,nn,nip,nodof,nod,nst,ndim,oppo
  ndof=nod*nodof
 allocate ( g_coord(ndim,nn),g_num(nod,nels),nf(nodof,nn), g_g(ndof,nels),   &
         g(ndof),points(nip,ndim),weights(nip),coord(nod,ndim),fun(nod),      &
         jac(ndim,ndim), der(ndim,nod),deriv(ndim,nod),bee(nst,ndof),         &
```

```
        num(nod),km(ndof,ndof), eld(ndof),  sigma(nst),etype(nels),          &
        eps(nst), evp(nst), devp(nst), bload(ndof),eload(ndof),              &
        evpt(nst,nip,nels), tabv(7,nip,nels), material(oppo),                &
        tsigma(nst,nip,nels),tevp(nst,nip,nels),dee(nst,nst),                &
        tdevp(nst,nip,nels),gc(ndim),tgc(ndim,nip,nels),                     &
        teps(nst,nip,nels))
  read(10,*) nprops , np_types
 allocate(prop(nprops,np_types)) ; read(10,*) prop
    etype = 1 ; if(np_types>1) read(10,*) etype
        do i=1, nn
          read (10,*)k,g_coord(:,i)
        end do
        do i=1, nels
          read (10,*)k, g_num(:,i)
       end do
 nf=1; read(10,*) nr ; if(nr>0) read(10,*)(k,nf(:,k),i=1,nr)
call formnf (nf);neq=maxval(nf); allocate(kdiag(neq))    ; kdiag = 0
!----------------- loop the elements to set up global arrays and kdiag --------------------
    elements_1   : do iel =1, nels
                num=g_num(:,iel)  ;  call num_to_g ( num , nf , g )
                g_g(:,iel)=g;    call fkdiag(kdiag,g)
               end do elements_1
       kdiag(1)=1; do i=2,neq; kdiag(i)=kdiag(i)+kdiag(i-1); end do
allocate( kv(kdiag(neq)),loads(0:neq),bdylds(0:neq), load_store(0:neq)); kv=0.0
     call sample(element,points,weights)
!---------------- element stiffness integration and assembly & set stresses----------------
elements_2: do iel = 1 , nels
          num = g_num(:, iel);    g = g_g( : , iel )
          coord = transpose(g_coord(:, num)) ;      km=0.0
       gauss_pts_1: do i = 1 , nip
```

```
          e=prop(1,etype(iel)); v=prop(2,etype(iel)); call deemat(dee,e,v)
            call shape_der(der,points,i) ; jac = matmul(der,coord)
              det = determinant(jac); call invert(jac)
              gc=matmul(fun,coord); tgc(:,i,iel)=gc
          deriv = matmul(jac,der) ; call beemat (bee,deriv)
       km = km + matmul(matmul(transpose(bee),dee),bee) *det* weights(i)
    end do gauss_pts_1
  call fsparv (kv,km,g,kdiag)
 end do elements_2
!------------------------ solution of equilibrium equation---------------------------------
  bdylds=.0;   evpt(1,nip,nels)=0.0;  evpt(2,nip,nels)=0.0
  evpt(3,nip,nels)=0.0;  evpt(4,nip,nels)=0.0
    read(10,*) loaded_nodes;   if(loaded_nodes/=0) then
    read(10,*)(k,loads(nf(:,k)),i=1,loaded_nodes); load_store = loads
   end if
   read(10,*) fixed_nodes;  if(fixed_nodes /=0) then
    allocate(node(fixed_nodes),sense(fixed_nodes),value(fixed_nodes),       &
          no(fixed_nodes),storkv(fixed_nodes))
    read(10,*) (node(i), sense(i), value(i),i=1,fixed_nodes)
    do i=1,fixed_nodes; no(i)=nf(sense(i),node(i)); end do
    kv(kdiag(no)) = kv(kdiag(no)) + 1.e20  ; storkv = kv(kdiag(no))
   end if
      call sparin (kv,kdiag); bdylds=.0; T=1;  t0=0
         do i=1,nels; do j=1,nip; do k=1,7
               tabv(k,j,i)=0
          end do; end do; end do
       tsigma=0; tevp=0; tdevp=0
   do ii=1,1603;  ij=ii*iy; do iy=0,0;  t0=t0+t
        iters=0;bdylds=0;evpt=0; loads =.0
     if(loaded_nodes/=0) loads = load_store
    if(fixed_nodes/=0) loads(no) = storkv * value
```

```
      loads = loads + bdylds
     call spabac(kv,loads,kdiag)
!---------------------------- recover initial elastic stress----------------------------------
 elements_3:do iel=1,nels; bload=0
          num = g_num(:,iel) ; coord = transpose(g_coord(:,num))
          g = g_g( : , iel)  ;     eld=loads(g)
       integrating_pts_2: do i = 1 , nip
         call shape_fun(fun,points,i); call shape_der(der,points,i)
          jac=matmul(der,coord); det = determinant(jac)
              call invert(jac);  deriv=matmul(jac,der)
              call beemat(bee,deriv); eps=matmul(bee,eld)
           eps=eps-evpt(:,i,iel);  sigma=matmul(dee,eps)
        tsigma(:,i,iel)=sigma
!--------------------- creep damage variables and stress updating ----------------------
 abv=tabv(:,i,iel);   call rdmpes (sigma,mpss,ess)
 do ix=1, oppo; material(ix)=prop(ix+2,etype(iel));  end do
   call EULER_KR (abv,crate,t,t0,sigma,ess,mpss,material)
      tabv(:,i,iel)=abv; tevp(:,i,iel)=evp
       evpt(:,i,iel)=evpt(:,i,iel)+evp
          devp=matmul(dee,evp); tdevp(:,i,iel)=devp
            eload=matmul(devp,bee)
          bload=bload+eload*det*weights(i)
      if(tabv(7,i,iel)>=0.99)then
             tabv(7,i,iel)=0.99; tevp(:,i,iel)=0.0; km=0.0
                 else
    tabv(:,i,iel)=abv; tevp(:,i,iel)=evp
             evpt(:,i,iel)=evpt(:,i,iel)+evp
               end if
    end do integrating_pts_2
 bdylds( g ) = bdylds( g )+ bload     ; bdylds(0) = 0
```

```
end do elements_3; end do; end do
!----------------- output of all calculated results for post-processing----------------
 write(11,99998) key1; write(11,*)ndim,nn,nod,nels,element,nst,nip,t0,e,v
write(11,99998) key2;  do k=1,nn; write(11,*) k ,g_coord(:,k);end do
write(11,99998) key3;  do k = 1 , nels; write(11,*) k ,g_num(:,k), key1; end do
write(11,99998) key4; do k=1,nn; write(11,*) k,loads(nf(:,k)); end do
write(11,99998) key5; do k=1,nn; write(11,*) k,bdylds(nf(:,k)); end do
write(11,99998) key7;  do i=1,nels; write(11,*) i
        do j=1,nip; write(11,*) j, tsigma(:,j,i); end do;  end do
write(11,99998) key8; do i=1,nels; write(11,*) i
        do j=1,nip; write(11,*)  j, teps(:,j,i); end do; end do
write(11,99998) key9; do i=1,nels;  write(11,*) i
        do j=1,nip; write(11,*)  j, evpt(:,j,i);  end do; end do
write(11,99998) key10;  do i=1,nels; write(11,*) i
       do j=1,nip; write(11,*)  j, tabv(7,j,i); end do;  end do
write(11,99998) key11
end program D3
```

Appendix B: Source Codes of Smith's Subroutine Library for HITSI

```
!------------------------------------------  Declaration  ----------------------------------------------------
!---------- geotech / software / prog_fe / GEOMETRY.F90 in (Smith and Griffiths, 2005) --------
!---------- geotech / software / prog_fe /NEW_LIBR.F90 in (Smith and Griffiths, 2005) --------
!---------------- Source codes of Smith's subroutines that have been used in HITSI -----------------
module geometry_lib
contains
!----------------Node to freedom number conversion ----------------------------
 subroutine num_to_g(num,nf,g)
 !finds the g vector from num and nf
 implicit none
 integer,intent(in)::num(:),nf(:,:)  ; integer,intent(out):: g(:)
 integer::i,k,nod,nodof ; nod=ubound(num,1) ; nodof=ubound(nf,1)
  do i = 1 , nod
     k = i*nodof  ; g(k-nodof+1:k) = nf( : , num(i) )
  end do
 return
 end subroutine num_to_g
!----------------------------- Triangles -------------------------------------
subroutine geometry_3tx(iel,nxe,aa,bb,coord,num)
!     this subroutine forms the coordinates and node vector
!     for a rectangular mesh of uniform 3-node triangles
!     counting in the x-direction    ; local numbering clockwise
  implicit none
 doubleprecision,intent(in):: aa,bb; integer,intent(in):: iel,nxe
 doubleprecision,intent(out) :: coord(:,:); integer,intent(out):: num(:)
  integer::ip,iq,jel
     jel= (2*nxe+iel-1)/(2*nxe)
```

```
    if(iel/2*2==iel)then; iq=2*jel; else; iq=2*jel-1; end if
    ip= (iel-2*nxe*(jel-1)+1)/2
    if(mod(iq,2)/=0) then
     num(1)=(nxe+1)*(iq-1)/2+ip  ;  num(3)=(nxe+1)*(iq+1)/2+ip
     num(2)=num(1)+1          ;  coord(1,1)=(ip-1)*aa
     coord(1,2)=-(iq-1)/2*bb    ;  coord(3,1)=(ip-1)*aa
     coord(2,2)=coord(1,2)      ;  coord(2,1)=ip*aa
     coord(3,2)=-(iq+1)/2*bb
    else
     num(1)=(nxe+1)*iq/2+ip+1    ;  num(3)=(nxe+1)*(iq-2)/2+ip+1
     num(2)=num(1)-1          ;  coord(1,1)=ip*aa
     coord(1,2)=-iq/2*bb        ;  coord(3,1)=ip*aa
     coord(3,2)=-(iq-2)/2*bb    ;  coord(2,1)=(ip-1)*aa
     coord(2,2)=coord(1,2)
    end if
   return
  end subroutine geometry_3tx

subroutine geometry_6tx(iel,nxe,aa,bb,coord,num)
!    this subroutine forms the coordinates and nodal vector
!    for a rectangular mesh of uniform 6-node triangles
!    counting in the x-direction ; local numbering clockwise
  implicit none
  doubleprecision ,intent(in):: aa,bb; integer,intent(in):: iel,nxe
  doubleprecision,intent(out) :: coord(:,:); integer,intent(out):: num(:)
  integer::ip,iq,jel,i
    jel= (2*nxe+iel-1)/(2*nxe)
    if(iel/2*2==iel)then; iq=2*jel; else; iq=2*jel-1; end if
    ip= (iel-2*nxe*(jel-1)+1)/2
    if(mod(iq,2)/=0) then
     num(1)=(iq-1)*(2*nxe+1)+2*ip-1    ;  num(2)=num(1)+1
```

```
    num(3)=num(1)+2              ;   num(4)=num(2)+1
    num(6)=(iq-1)*(2*nxe+1)+2*nxe+2*ip ;   num(5)=(iq+1)*(2*nxe+1)+2*ip-1
    coord(1,1)=(ip-1)*aa         ;   coord(1,2)=-(iq-1)/2*bb
    coord(5,1)=(ip-1)*aa         ;   coord(5,2)=-(iq+1)/2*bb
    coord(3,1)=ip*aa             ;   coord(3,2)=coord(1,2)
    else
    num(1)=iq*(2*nxe+1)+2*ip+1      ; num(6)=(iq-2)*(2*nxe+1)+2*nxe+2*ip+2
    num(5)=(iq-2)*(2*nxe+1)+2*ip+1    ; num(4)=num(2)-1
    num(3)=num(1)-2              ; num(2)=num(1)-1
    coord(1,1)=ip*aa             ; coord(1,2)=-iq/2*bb
    coord(5,1)=ip*aa             ; coord(5,2)=-(iq-2)/2*bb
    coord(3,1)=(ip-1)*aa         ; coord(3,2)=coord(1,2)
   end if
   do  i=1,2
   coord(2,i)=.5*(coord(1,i)+coord(3,i))
   coord(4,i)=.5*(coord(3,i)+coord(5,i))
   coord(6,i)=.5*(coord(5,i)+coord(1,i))
   end do
   return
  end subroutine geometry_6tx

 subroutine geometry_15tyv(iel,nye,width,depth,coord,num)
!    this subroutine forms the coordinates and node vector
!    for a rectangular mesh of nonuniform 15-node triangles
!    counting in the y-direction ; local numbering clockwise
 implicit none
 doubleprecision,intent(in) :: width(:),depth(:)
 doubleprecision,intent(out) :: coord(:,:)
 integer, intent(in) ::  iel,nye; integer,intent(out)::num(:)
 integer::ip,iq,jel,i , fac1,fac2
  jel = (iel - 1)/nye; ip= jel+1; iq=iel-nye*jel
```

```
if(mod(iq,2)/=0) then
 fac1=4*(2*nye+1)*(ip-1)+2*iq-1 ; num(1)=fac1;  num(12)=fac1+1
 num(11)=fac1+2 ;  num(10)=fac1+3 ; num(9)=fac1+4 ; num(8)=fac1+2*nye+4
 num(7)=fac1+4*nye+4 ;  num(6)=fac1+6*nye+4 ;  num(5)=fac1+8*nye+4
 num(4)=fac1+6*nye+3 ;  num(3)=fac1+4*nye+2 ;  num(2)=fac1+2*nye+1
 num(13)=fac1+2*nye+2 ;  num(15)=fac1+2*nye+3  ;  num(14)=fac1+4*nye+3
 coord(1,1)=width(ip) ;  coord(1,2)=depth((iq+1)/2)
 coord(9,1)=width(ip)   ;  coord(9,2)=depth((iq+3)/2)
 coord(5,1)=width(ip+1)   ;  coord(5,2)=depth((iq+1)/2)
else
 fac2=4*(2*nye+1)*(ip-1)+2*iq+8*nye+5 ;  num(1)=fac2 ;  num(12)=fac2-1
 num(11)=fac2-2 ;  num(10)=fac2-3 ;  num(9)=fac2-4; num(8)=fac2-2*nye-4
 num(7)=fac2-4*nye-4 ; num(6)=fac2-6*nye-4  ;  num(5)=fac2-8*nye-4
 num(4)=fac2-6*nye-3 ; num(3)=fac2-4*nye-2 ; num(2)=fac2-2*nye-1
 num(13)=fac2-2*nye-2  ; num(15)=fac2-2*nye-3 ; num(14)=fac2-4*nye-3
 coord(1,1)=width(ip+1) ;  coord(1,2)=depth((iq+2)/2)
 coord(9,1)=width(ip+1) ;  coord(9,2)=depth(iq/2)
 coord(5,1)=width(ip)   ;   coord(5,2)=depth((iq+2)/2)
end if
do  i=1,2
 coord(3,i)=.5*(coord(1,i)+coord(5,i))
 coord(7,i)=.5*(coord(5,i)+coord(9,i))
 coord(11,i)=.5*(coord(9,i)+coord(1,i))
 coord(2,i)=.5*(coord(1,i)+coord(3,i))
 coord(4,i)=.5*(coord(3,i)+coord(5,i))
 coord(6,i)=.5*(coord(5,i)+coord(7,i))
 coord(8,i)=.5*(coord(7,i)+coord(9,i))
 coord(10,i)=.5*(coord(9,i)+coord(11,i))
 coord(12,i)=.5*(coord(11,i)+coord(1,i))
 coord(15,i)=.5*(coord(7,i)+coord(11,i))
 coord(14,i)=.5*(coord(3,i)+coord(7,i))
```

```
    coord(13,i)=.5*(coord(2,i)+coord(15,i))
   end do
  return
 end subroutine geometry_15tyv
!---------------------- Quadrilaterals ---------------------------------------
subroutine geometry_4qx(iel,nxe,aa,bb,coord,num)
! coordinates and nodal vectors for equal four node quad
! elements, numbering in x
implicit none
integer,intent(in)::iel,nxe; doubleprecision,intent(in)::aa,bb
doubleprecision,intent(out)::coord(:,:); integer,intent(out)::num(:)
integer :: ip,iq    ; iq=(iel-1)/nxe+1; ip=iel-(iq-1)*nxe
  num=(/iq*(nxe+1)+ip,(iq-1)*(nxe+1)+ip,   &
       (iq-1)*(nxe+1)+ip+1, iq*(nxe+1)+ip+1/)
  coord(1:2,1)=(ip-1)*aa; coord(3:4,1)=ip*aa
  coord(1:4:3,2)=-iq*bb; coord(2:3,2)=-(iq-1)*bb
 return
end subroutine geometry_4qx

  subroutine geometry_4qy(iel,nye,aa,bb,coord,num)
  ! rectangles of equal 4-node quads numbered in y
  implicit none
  integer,intent(in)::iel,nye; doubleprecision,intent(in)::aa,bb
  doubleprecision,intent(out)::coord(:,:);integer,intent(out)::num(:)
     num=(/iel+(iel-1)/nye+1,iel+(iel-1)/nye,iel+(iel-1)/nye+nye+1,  &
          iel+(iel-1)/nye+nye+2/)
     coord(1:2,1)= aa*((iel-1)/nye); coord(3:4,1)=aa*((iel-1)/nye+1)
     coord(1:4:3,2)=-(iel-((iel-1)/nye)*nye)*bb; coord(2:3,2)=coord(1,2)+bb
    return
  end subroutine geometry_4qy
```

```
subroutine geometry_4qyv(iel,nye,width,depth,coord,num)
!  coordinates and steering vector for a variable rectangular
!  mesh of 4-node quad elements numbering in the y-direction
  implicit none
  doubleprecision,intent(in)::width(:),depth(:); integer,intent(in)::iel,nye
  doubleprecision,intent(out)::coord(:,:); integer,intent(out):: num(:)
  integer:: ip,iq; ip=(iel-1)/nye+1; iq=iel-(ip-1)*nye
  num(1)=(ip-1)*(nye+1)+iq+1; num(2)=num(1)-1
  num(3)=ip*(nye+1)+iq;num(4)= num(3) + 1
  coord(1:2,1)=width(ip); coord(3:4,1)=width(ip+1)
  coord(1,2)=depth(iq+1); coord(2:3,2)=depth(iq); coord(4,2)=coord(1,2)
 return
end subroutine geometry_4qyv
subroutine geometry_8qx(iel,nxe,aa,bb,coord,num)
! coordinates and steering vector for a rectangular mesh of
! equal  8-node  elements  numbering in x
implicit none
 doubleprecision,intent(out)::coord(:,:); integer,intent(out)::num(:)
 integer,intent(in)::iel,nxe; doubleprecision,intent(in)::aa,bb
 integer:: ip,iq ; iq=(iel-1)/nxe+1; ip=iel-(iq-1)*nxe
 num(1)=iq*(3*nxe+2)+2*ip-1; num(2)=iq*(3*nxe+2)+ip-nxe-1
 num(3)=(iq-1)*(3*nxe+2)+2*ip-1; num(4)=num(3)+1
 num(5)=num(4)+1; num(6)=num(2)+1; num(7)=num(1)+2; num(8)=num(1)+1
 coord(1:3,1)=aa*(ip-1); coord(5:7,1)=aa*ip
 coord(4,1)=.5*(coord(3,1)+coord(5,1))
 coord(8,1)=.5*(coord(7,1)+coord(1,1))
 coord(1,2)=-bb*iq; coord(7:8,2)=-bb*iq
 coord(3:5,2)=-bb*(iq-1); coord(2,2)=.5*(coord(1,2)+coord(3,2))
 coord(6,2)=.5*(coord(5,2)+coord(7,2))
return
end subroutine geometry_8qx
```

```
subroutine geometry_8qy(iel,nye,aa,bb,coord,num)
!  coordinates and steering vector for a constant rectangular
!  mesh of 8-node quad elements numbering in the y-direction
  implicit none
  doubleprecision,intent(in):: aa,bb ; integer,intent(in):: iel,nye
  doubleprecision,intent(out)::coord(:,:); integer,intent(out):: num(:)
  integer:: ip,iq; ip=(iel-1)/nye+1; iq=iel-(ip-1)*nye
  num(1)=(ip-1)*(3*nye+2)+2*iq+1; num(2)=num(1)-1; num(3)=num(1)-2
  num(4)=(ip-1)*(3*nye+2)+2*nye+iq+1;num(5)=ip*(3*nye+2)+2*iq-1
  num(6)=num(5)+1; num(7)=num(5)+2; num(8)=num(4)+1
  coord(1:3,1)=(ip-1)*aa; coord(5:7,1)=ip*aa
  coord(4,1)=.5*(coord(3,1)+coord(5,1))
  coord(8,1)=.5*(coord(7,1)+coord(1,1))
  coord(1,2)=-iq*bb; coord(7:8,2)=-iq*bb; coord(3:5,2)=-(iq-1)*bb
  coord(2,2)=.5*(coord(1,2)+coord(3,2))
  coord(6,2)=.5*(coord(5,2)+coord(7,2))
 return
end subroutine geometry_8qy

subroutine geometry_8qxv(iel,nxe,width,depth,coord,num)
! nodal coordinates and node vector for a variable mesh of
! 8-node quadrilaterals numbering in the x-direction
implicit none
  integer,intent(in)::iel,nxe; doubleprecision,intent(in)::width(:),depth(:)
  doubleprecision,intent(out)::coord(:,:); integer,intent(out)::num(:)
  integer::ip,iq; iq=(iel-1)/nxe+1; ip=iel-(iq-1)*nxe
  num(1)=iq*(3*nxe+2)+2*ip-1; num(2)=iq*(3*nxe+2)+ip-nxe-1
  num(3)=(iq-1)*(3*nxe+2)+2*ip-1; num(4)=num(3)+1;num(5)=num(4)+1
  num(6)=num(2)+1; num(7)=num(1)+2; num(8)=num(1)+1
  coord(1:3,1)=width(ip); coord(5:7,1)=width(ip+1)
```

```
 coord(4,1)=.5*(coord(3,1)+coord(5,1));coord(8,1)=.5*(coord(7,1)+coord(1,1))
 coord(1,2)=depth(iq+1); coord(7:8,2)=depth(iq+1); coord(3:5,2)=depth(iq)
 coord(2,2)=.5*(coord(1,2)+coord(3,2));coord(6,2)=.5*(coord(5,2)+coord(7,2))
 return
end subroutine geometry_8qxv

subroutine geometry_8qyv(iel,nye,width,depth,coord,num)
!  coordinates and steering vector for a variable rectangular
!  mesh of 8-node quad elements numbering in the y-direction
 implicit none
 doubleprecision,intent(in)::width(:),depth(:); integer,intent(in)::iel,nye
 doubleprecision,intent(out)::coord(:,:); integer,intent(out)::num(:)
 integer::ip,iq; ip=(iel-1)/nye+1; iq=iel-(ip-1)*nye
 num(1)=(ip-1)*(3*nye+2)+2*iq+1; num(2)=num(1)-1; num(3)=num(1)-2
 num(4)=(ip-1)*(3*nye+2)+2*nye+iq+1;num(5)=ip*(3*nye+2)+2*iq-1
 num(6)=num(5)+1; num(7)=num(5)+2; num(8)=num(4)+1
 coord(1:3,1)=width(ip); coord(5:7,1)=width(ip+1)
 coord(4,1)=.5*(coord(3,1)+coord(5,1))
 coord(8,1)=.5*(coord(7,1)+coord(1,1))
 coord(1,2)=depth(iq+1); coord(7:8,2)=depth(iq+1); coord(3:5,2)=depth(iq)
 coord(2,2)=.5*(coord(1,2)+coord(3,2))
 coord(6,2)=.5*(coord(5,2)+coord(7,2))
 return
end subroutine geometry_8qyv

 subroutine geometry_9qx(iel,nxe,aa,bb,coord,num)
!     this subroutine forms the coordinates and steering vector
!     for equal 9-node Lagrangian quads counting in x-direction
 implicit none
 doubleprecision,intent(out)::coord(:,:); integer,intent(out)::num(:)
 integer,intent(in)::iel,nxe; doubleprecision,intent(in)::aa,bb
```

```
 integer:: ip,iq ;iq=(iel-1)/nxe+1;ip=iel-(iq-1)*nxe
  num(1)=iq*(4*nxe+2)+2*ip-1 ; num(2)=iq*(4*nxe+2)+2*ip-nxe-4
  num(3)= (iq-1)*(4*nxe+2)+2*ip-1 ;   num(4)=num(3)+1
  num(5)=num(4)+1; num(6)=num(2)+2 ;  num(7)=num(1)+2
  num(8)=num(1)+1     ;   num(9)=num(2)+1
  coord(1,1)=(ip-1)*aa  ;  coord(3,1)=(ip-1)*aa   ; coord(5,1)=ip*aa
  coord(7,1)=ip*aa ;   coord(1,2)=-iq*bb ;  coord(3,2)=-(iq-1)*bb
  coord(5,2)=-(iq-1)*bb    ;  coord(7,2)=-iq*bb
  coord(2,1)=.5*(coord(1,1)+coord(3,1)); coord(2,2)=.5*(coord(1,2)+coord(3,2))
  coord(4,1)=.5*(coord(3,1)+coord(5,1)); coord(4,2)=.5*(coord(3,2)+coord(5,2))
  coord(6,1)=.5*(coord(5,1)+coord(7,1)); coord(6,2)=.5*(coord(5,2)+coord(7,2))
  coord(8,1)=.5*(coord(1,1)+coord(7,1)); coord(8,2)=.5*(coord(1,2)+coord(7,2))
  coord(9,1)=.5*(coord(2,1)+coord(6,1)); coord(9,2)=.5*(coord(4,2)+coord(8,2))
 return
 end subroutine geometry_9qx
!-----------------------Hexahedra "Bricks" -----------------------------------
 subroutine geometry_8bxz(iel,nxe,nze,aa,bb,cc,coord,num)
!     this subroutine forms the coordinates and nodal vector
!     for boxes of 8-node brick elements counting x-z planes in y-direction
 implicit none
  integer,intent(in)::iel,nxe,nze;integer,intent(out)::num(:)
  doubleprecision,intent(in)::aa,bb,cc; doubleprecision,intent(out)::coord(:,:)
  integer::ip,iq,is,iplane
  iq=(iel-1)/(nxe*nze)+1 ; iplane = iel -(iq-1)*nxe*nze
  is=(iplane-1)/nxe+1; ip = iplane-(is-1)*nxe
  num(1)=(iq-1)*(nxe+1)*(nze+1)+is*(nxe+1)+ip ;  num(2)=num(1)-nxe-1
  num(3)=num(2)+1 ;   num(4)=num(1)+1 ; num(5)=num(1)+(nxe+1)*(nze+1)
  num(6)=num(5)-nxe-1 ;   num(7)=num(6)+1    ;   num(8)=num(5)+1
  coord(1,1)=(ip-1)*aa ; coord(2,1)=(ip-1)*aa ;  coord(5,1)=(ip-1)*aa
  coord(6,1)=(ip-1)*aa ; coord(3,1)=ip*aa ;   coord(4,1)=ip*aa
  coord(7,1)=ip*aa   ;   coord(8,1)=ip*aa
```

```
 coord(1,2)=(iq-1)*bb  ;   coord(2,2)=(iq-1)*bb ; coord(3,2)=(iq-1)*bb
 coord(4,2)=(iq-1)*bb  ;   coord(5,2)=iq*bb  ;    coord(6,2)=iq*bb
 coord(7,2)=iq*bb ;    coord(8,2)=iq*bb ;   coord(1,3)=-is*cc
 coord(4,3)=-is*cc  ;   coord(5,3)=-is*cc ;   coord(8,3)=-is*cc
 coord(2,3)=-(is-1)*cc ; coord(3,3)=-(is-1)*cc  ;  coord(6,3)=-(is-1)*cc
 coord(7,3)=-(is-1)*cc
 return
end subroutine geometry_8bxz

subroutine geometry_20bxz(iel,nxe,nze,aa,bb,cc,coord,num)
! nodal vector and nodal coordinates for boxes of 20-node
! bricks counting x-z planes in the y-direction
implicit none
 integer,intent(in)::iel,nxe,nze; doubleprecision,intent(in)::aa,bb,cc
 doubleprecision,intent(out)::coord(:,:); integer,intent(out)::num(:)
 integer::fac1,fac2,ip,iq,is,iplane
 iq = (iel-1)/(nxe*nze)+1; iplane = iel-(iq-1)*nxe*nze
 is = (iplane-1)/nxe+1 ; ip = iplane-(is-1)*nxe
 fac1=((2*nxe+1)*(nze+1)+(2*nze+1)*(nxe+1))*(iq-1)
 fac2=((2*nxe+1)*(nze+1)+(2*nze+1)*(nxe+1))*iq
 num(1)=fac1+(3*nxe+2)*is+2*ip-1
 num(2)=fac1+(3*nxe+2)*is-nxe+ip-1; num(3)=num(1)-3*nxe-2
 num(4)=num(3)+1; num(5)=num(4)+1; num(6)=num(2)+1
 num(7)=num(1)+2; num(8)=num(1)+1
 num(9)=fac2-(nxe+1)*(nze+1)+(nxe+1)*is+ip
 num(10)=num(9)-nxe-1; num(11)=num(10)+1; num(12)=num(9)+1
 num(13)=fac2+(3*nxe+2)*is+2*ip-1
 num(14)=fac2+(3*nxe+2)*is-nxe+ip-1
 num(15)=num(13)-3*nxe-2; num(16)=num(15)+1; num(17)=num(16)+1
 num(18)=num(14)+1; num(19)=num(13)+2; num(20)=num(13)+1
 coord(1:3,1)=(ip-1)*aa; coord(9:10,1)=(ip-1)*aa; coord(13:15,1)=(ip-1)*aa
```

```
 coord(5:7,1)=ip*aa; coord(11:12,1)=ip*aa; coord(17:19,1)=ip*aa
 coord(4,1)=.5*(coord(3,1)+coord(5,1));coord(8,1)=.5*(coord(1,1)+coord(7,1))
 coord(16,1)=.5*(coord(15,1)+coord(17,1))
 coord(20,1)=.5*(coord(13,1)+coord(19,1))
 coord(1:8,2)=(iq-1)*bb; coord(13:20,2)=iq*bb
 coord(9,2)=.5*(coord(1,2)+coord(13,2))
 coord(10,2)=.5*(coord(3,2)+coord(15,2))
 coord(11,2)=.5*(coord(5,2)+coord(17,2))
 coord(12,2)=.5*(coord(7,2)+coord(19,2))
 coord(1,3)=-is*cc; coord(7:9,3)=-is*cc; coord(12:13,3)=-is*cc
 coord(19:20,3)=-is*cc; coord(3:5,3)=-(is-1)*cc
 coord(10:11,3)=-(is-1)*cc; coord(15:17,3)=-(is-1)*cc
 coord(2,3)=.5*(coord(1,3)+coord(3,3))
 coord(6,3)=.5*(coord(5,3)+coord(7,3))
 coord(14,3)=.5*(coord(13,3)+coord(15,3))
 coord(18,3)=.5*(coord(17,3)+coord(19,3))
 return
end subroutine geometry_20bxz
end module geometry_lib
!---------------------------------------------  Declaration  ----------------------------------------------------
module new_library
contains
subroutine sparin_gauss(kv,kdiag)
! Gaussian factorisation of a skyline matrix
implicit none
doubleprecision,intent(out)::kv(:) ; integer,intent(in)::kdiag(:)
doubleprecision::num,den,fac; integer::n,ii,i,j,k,l,kk,l1,l2,l3; n = ubound(kdiag,1)
 do j = 1 , n-1
   den = kv(kdiag(j))
   ii = 0
   do i = j+1 , n
```

```
   ii = ii + 1 ;   l = kdiag(i) - ii
    if(l-kdiag(i-1)>.0) then
     num = kv(l)   ; fac = num/den ; kk = -1
     do k = i , n
       kk = kk + 1; l1=kdiag(i+kk)-kk;  l2=l1-ii; l3=kdiag(i+kk-1)
       if(l2-l3>.0) then
         kv(l1) = kv(l1) - fac*kv(l2)
       end if
     end do
    end if
   end do
 end do
 return
end subroutine sparin_gauss

subroutine spabac_gauss(kv,loads,kdiag)
! Gaussian back-substitution on a skyline matrix
implicit none
doubleprecision,intent(in)::kv(:);doubleprecision,intent(inout)::loads(0:)
integer,intent(in)::kdiag(:)
doubleprecision::num,den,fac,asum;integer::i,j,l,n,ii,jj,l1,l2; n=ubound(kdiag,1)
 do j = 1 , n-1
   den = kv(kdiag(j))  ;  ii = 0
   do i = j+1 , n
     ii = ii + 1     ; l = kdiag(i) - ii
     if(l-kdiag(i-1)>.0) then
       num = kv(l)   ; fac = num/den  ; loads(i)=loads(i)-fac*loads(j)
     end if
   end do
 end do
 loads(n) = loads(n)/kv(kdiag(n))
```

```
 do i = n-1 , 1 , -1
   jj = 0    ;  asum = .0
   do j = i+1 , n
     jj = jj + 1 ; l1 = kdiag(i+jj)-jj ;  l2 = kdiag(i+jj-1)
     if(l1-l2>.0) then
       asum = asum + kv(l1) * loads(j)
     end if
   end do
   loads(i) = (loads(i) - asum)/kv(kdiag(i))
 end do
 return
end subroutine spabac_gauss

subroutine bandred(a,d,e,e2)
!     this subroutine transforms a real symmetric band matrix a,
!     of order n and band width iw,to tridiagonal form by an appropriate
!     sequence of jacobi rotations. during the transformation the
!     property of the band matrix is maintained. the method yields
!     a tridiagonal matrix, the diagonal elements of which are in
!     d(n) and off-diagonal elements in e(n).
implicit none
   doubleprecision,intent(in out)::a(:,:)
   doubleprecision,intent(out)::d(0:),e(0:),e2(0:)
    integer::  iw, n2, n, k, maxr, irr, ir, kr, j, jm, iugl, j2,         &
           l, jl, maxl, i
    doubleprecision ::    g, b, s, c, c2, s2, cs, u, u1
     n=ubound(a,1) ; iw = ubound(a,2)-1
    n2 = n - 2
    if (n2>=1) then
    do 160 k=1,n2
      maxr = iw     ;     if (n-k<iw) maxr = n - k
```

```
 do 140 irr=2,maxr
   ir = 2 + maxr - irr  ;    kr = k + ir
   do 120 j=kr,n,iw
     if (j==kr) go to 20  ;  if (g==0.0) go to 140
     jm = j - iw   ; b = -a(jm-1,iw+1)/g   ; iugl = j - iw
     go to 40
20    if (a(k,ir+1)==0.0) go to 140
     b = -a(k,ir)/a(k,ir+1) ;  iugl = k
40    s = 1.0/sqrt(1.0+b*b);  c = b*s; c2 = c*c ;s2 = s*s  ; cs = c*s
     u = c2*a(j-1,1) - 2.0*cs*a(j-1,2) + s2*a(j,1)
     u1 = s2*a(j-1,1) + 2.0*cs*a(j-1,2) + c2*a(j,1)
     a(j-1,2) = cs*(a(j-1,1)-a(j,1)) + (c2-s2)*a(j-1,2)
     a(j-1,1) = u       ;   a(j,1) = u1 ;    j2 = j - 2
     do  l=iugl,j2
       jl = j - l     ;   u = c*a(l,jl) - s*a(l,jl+1)
       a(l,jl+1) = s*a(l,jl) + c*a(l,jl+1) ;   a(l,jl) = u
     end do
     jm = j - iw
     if (j/=kr) a(jm-1,iw+1) = c*a(jm-1,iw+1) - s*g
     maxl = iw - 1     ; if (n-j<iw-1) maxl = n - j
     if (maxl>0) then
      do  l=1,maxl
       u = c*a(j-1,l+2) - s*a(j,l+1)
       a(j,l+1) = s*a(j-1,l+2) + c*a(j,l+1) ;  a(j-1,l+2) = u
      end do
     end if
   if (j+iw>n) go to 120
     g = -s*a(j,iw+1)   ;   a(j,iw+1) = c*a(j,iw+1)
120   continue
140   continue
160 continue
```

```
 end if
 e(1) = 0.0
   d(1:n) = a(1:n,1) ;  if (2>n) go to 240
   do  i=2,n   ;   e(i) = a(i-1,2) ;  end do
 240  e2 = e*e
 return
end subroutine bandred

 subroutine formnf(nf)
 ! reform nf
 implicit none
 integer,intent(in out)::nf(:,:)
  integer:: i,j,m
  m=0
  do j=1,ubound(nf,2)
    do i=1,ubound(nf,1)
      if(nf(i,j)/=0) then
        m=m+1; nf(i,j)=m
      end if
    end do
  end do
  return
end subroutine formnf

subroutine invert(matrix)
! invert a small square matrix onto itself
implicit none
doubleprecision,intent(in out)::matrix(:,:)
integer::i,k,n; doubleprecision::con  ; n= ubound(matrix,1)
do k=1,n
  con=matrix(k,k); matrix(k,k)=1.
```

```
   matrix(k,:)=matrix(k,:)/con
   do i=1,n
     if(i/=k) then
       con=matrix(i,k); matrix(i,k)=0.0
       matrix(i,:)=matrix(i,:) - matrix(k,:)*con
     end if
   end do
 end do
 return
end subroutine invert

function determinant (jac) result(det)
! returns the determinant of a 1x1 2x2 3x3 jacobian matrix
implicit none    ; doubleprecision :: det
doubleprecision,intent(in)::jac(:,:); integer:: it ; it = ubound(jac,1)
select case (it)
  case (1)
   det=1.0
  case (2)
   det=jac(1,1)*jac(2,2) - jac(1,2) * jac(2,1)
  case (3)
   det= jac(1,1)*(jac(2,2) * jac(3,3) -jac(3,2) * jac(2,3))
   det= det-jac(1,2)*(jac(2,1)*jac(3,3)-jac(3,1)*jac(2,3))
   det= det+jac(1,3)*(jac(2,1)*jac(3,2)-jac(3,1)*jac(2,2))
  case default
   print*,' wrong dimension for jacobian matrix'
end select
return
end function determinant

subroutine deemat(dee,e,v)
```

```
! returns the elastic dee matrix for given ih
! ih=3,plane strain; =4,axisymmetry or plane strain elastoplasticity
! =6 , three dimensional
implicit none
 doubleprecision,intent(in)::e,v; doubleprecision,intent(out)::dee(:,:)
! local variables
 doubleprecision::v1,v2,c,vv; integer :: i,ih;  dee=0.0  ; ih = ubound(dee,1)
     v1 = 1. - v; c = e/((1.+v)*(1.-2.*v))
 select case (ih)
      case(3)
        dee(1,1)=v1*c; dee(2,2)=v1*c; dee(1,2)=v*c; dee(2,1)=v*c
        dee(3,3)=.5*c*(1.-2.*v)
      case(4)
        dee(1,1)=v1*c; dee(2,2)=v1*c; dee(4,4)=v1*c
        dee(3,3)=.5*c*(1.-2.*v) ; dee(1,2)=v*c; dee(2,1)=v*c
        dee(1,4)=v*c; dee(4,1)=v*c; dee(2,4)=v*c; dee(4,2)=v*c
      case(6)
        v2=v/(1.-v); vv=(1.-2.*v)/(1.-v)*.5
        do i=1,3; dee(i,i)=1.;end do; do i=4,6; dee(i,i)=vv; end do
        dee(1,2)=v2; dee(2,1)=v2; dee(1,3)=v2; dee(3,1)=v2
        dee(2,3)=v2; dee(3,2)=v2
        dee = dee*e/(2.*(1.+v)*vv)
      case default
        print*,'wrong size for dee matrix'
 end select
return
end subroutine deemat

subroutine beemat(bee,deriv)
! bee matrix for 2-d elasticity or elastoplasticity (ih=3 or 4 respectively)
! or for 3-d (ih = 6)
```

```
implicit none
doubleprecision,intent(in)::deriv(:,:);  doubleprecision,intent(out)::bee(:,:)
! local variables
integer::k,l,m,n , ih,nod; doubleprecision::x,y,z
bee=0. ; ih = ubound(bee,1); nod = ubound(deriv,2)
   select case (ih)
    case(3,4)
     do m=1,nod
       k=2*m; l=k-1; x=deriv(1,m); y=deriv(2,m)
       bee(1,l)=x; bee(3,k)=x; bee(2,k)=y; bee(3,l)=y
     end do
    case(6)
     do m=1,nod
       n=3*m;  k=n-1; l=k-1
       x=deriv(1,m); y=deriv(2,m); z=deriv(3,m)
       bee(1,l)=x; bee(4,k)=x; bee(6,n)=x
       bee(2,k)=y; bee(4,l)=y; bee(5,n)=y
       bee(3,n)=z; bee(5,k)=z; bee(6,l)=z
     end do
    case default
     print*,'wrong dimension for nst in bee matrix'
   end select
 return
end subroutine beemat

subroutine bmataxi(bee,radius,coord,deriv,fun)
! b matrix for axisymmetry
doubleprecision,intent(in)::deriv(:,:),fun(:),coord(:,:)
doubleprecision,intent(out)::bee(:,:),radius
 integer::nod ,k,l,m; doubleprecision :: x,y
  radius = sum(fun * coord(:,1))  ; nod = ubound(deriv , 2) ; bee = .0
```

```
do m = 1 , nod
 k=2*m; l = k-1 ; x = deriv(1,m); bee(1,l) = x; bee(3 , k) = x
 y = deriv(2,m); bee(2,k)=y; bee(3,l) = y; bee(4,l)=fun(m)/radius
end do
return
end subroutine bmataxi

subroutine sample(element,s,wt)
! returns the local coordinates of the integrating points
implicit none
 doubleprecision,intent(out)::s(:,:),wt(:)  ; character(*),intent(in):: element
 integer::nip ;  doubleprecision:: root3, r15 , w(3),v(9),b,c
 root3 = 1./sqrt(3.)   ;  r15 = .2*sqrt(15.)
 nip = ubound( s , 1 )
      w = (/5./9.,8./9.,5./9./); v=(/5./9.*w,8./9.*w,5./9.*w/)
   select case (element)
        case('line')
         select case(nip)
          case(1)
           s(1,1)=0.  ;  wt(1)=2.
          case(2)
           s(1,1)=root3  ; s(2,1)=-s(1,1)  ;  wt(1)=1.  ; wt(2)=1.
          case(3)
           s(1,1)=r15 ; s(2,1)=.0     ; s(3,1)=-s(1,1)
           wt = w
          case(4)
           s(1,1)=.861136311594053  ; s(2,1)=.339981043584856
           s(3,1)=-s(2,1)  ; s(4,1)=-s(1,1)
           wt(1)=.347854845137454 ; wt(2)=.652145154862546
           wt(3)=wt(2) ; wt(4)=wt(1)
          case(5)
```

```
        s(1,1)=.906179845938664 ; s(2,1)=.538469310105683
        s(3,1)=.0 ; s(4,1)=-s(2,1) ; s(5,1)=-s(1,1)
        wt(1)=.236926885056189 ; wt(2)=.478628670499366
        wt(3)=.568888888888889 ; wt(4)=wt(2) ; wt(5)=wt(1)
       case(6)
        s(1,1)=.932469514203152 ; s(2,1)=.661209386466265
        s(3,1)=.238619186083197
        s(4,1)=-s(3,1) ; s(5,1)=-s(2,1) ; s(6,1)=-s(1,1)
        wt(1)=.171324492379170 ; wt(2)=.360761573048139
        wt(3)=.467913934572691
        wt(4)=wt(3); wt(5)=wt(2) ; wt(6)=wt(1)
          case default
           print*,"wrong number of integrating points for a line"
      end select
     case('triangle')
      select case(nip)
       case(1)   ! for triangles weights multiplied by .5
        s(1,1)=1./3. ; s(1,2)=1./3. ;  wt(1)= .5
       case(3)
        s(1,1)=.5 ;  s(1,2)=.5 ;  s(2,1)=.5
        s(2,2)=0.;  s(3,1)=0. ;  s(3,2)=.5
        wt(1)=1./3. ;  wt(2)=wt(1) ; wt(3)=wt(1)   ; wt = .5*wt
       case(6)
s(1,1)=.816847572980459  ; s(1,2)=.091576213509771
s(2,1)=s(1,2);  s(2,2)=s(1,1) ;  s(3,1)=s(1,2); s(3,2)=s(1,2)
s(4,1)=.108103018168070 ;  s(4,2)=.445948490915965
s(5,1)=s(4,2) ;   s(5,2)=s(4,1) ;  s(6,1)=s(4,2)  ; s(6,2)=s(4,2)
wt(1)=.109951743655322 ;   wt(2)=wt(1)  ;   wt(3)=wt(1)
wt(4)=.223381589678011 ;   wt(5)=wt(4)  ;   wt(6)=wt(4)    ; wt = .5*wt
       case(7)
s(1,1)=1./3. ; s(1,2)=1./3.;s(2,1)=.797426985353087 ;s(2,2)=.101286507323456
```

s(3,1)=s(2,2) ; s(3,2)=s(2,1) ; s(4,1)=s(2,2) ; s(4,2)=s(2,2)

s(5,1)=.470142064105115 ; s(5,2)=.059715871789770

s(6,1)=s(5,2) ; s(6,2)=s(5,1); s(7,1)=s(5,1); s(7,2)=s(5,1)

wt(1)=.225 ; wt(2)=.125939180544827 ; wt(3)=wt(2); wt(4)=wt(2)

*wt(5)=.132394152788506; wt(6)=wt(5) ; wt(7)=wt(5) ;wt = .5*wt*

case(12)

s(1,1)=.873821971016996 ; s(1,2)=.063089014491502

s(2,1)=s(1,2) ; s(2,2)=s(1,1); s(3,1)=s(1,2) ; s(3,2)=s(1,2)

s(4,1)=.501426509658179 ; s(4,2)=.249286745170910

s(5,1)=s(4,2); s(5,2)=s(4,1) ; s(6,1)=s(4,2) ; s(6,2)=s(4,2)

s(7,1)=.636502499121399 ; s(7,2)=.310352451033785

s(8,1)=s(7,1) ; s(8,2)=.053145049844816 ; s(9,1)=s(7,2) ; s(9,2)=s(7,1)

s(10,1)=s(7,2) ; s(10,2)=s(8,2) ; s(11,1)=s(8,2); s(11,2)=s(7,1)

s(12,1)=s(8,2) ; s(12,2)=s(7,2)

wt(1)=.050844906370207 ; wt(2)=wt(1); wt(3)=wt(1)

wt(4)=.116786275726379 ; wt(5)=wt(4); wt(6)=wt(4)

*wt(7)=.082851075618374 ; wt(8:12)=wt(7) ; wt = .5*wt*

case(16)

s(1,1)=1./3. ; s(1,2)=1./3. ; s(2,1)=.658861384496478

s(2,2)=.170569307751761 ; s(3,1)=s(2,2) ; s(3,2)=s(2,1)

s(4,1)=s(2,2) ; s(4,2)=s(2,2)

s(5,1)=.898905543365938 ; s(5,2)=.050547228317031

s(6,1)=s(5,2); s(6,2)=s(5,1) ; s(7,1)=s(5,2) ; s(7,2)=s(5,2)

s(8,1)=.081414823414554; s(8,2)=.459292588292723

s(9,1)=s(8,2) ; s(9,2)=s(8,1); s(10,1)=s(8,2) ; s(10,2)=s(8,2)

s(11,1)=.008394777409958; s(11,2)=.263112829634638

s(12,1)=s(11,1) ; s(12,2)=.728492392955404

s(13,1)=s(11,2) ; s(13,2)=s(11,1) ; s(14,1)=s(11,2); s(14,2)=s(12,2)

s(15,1)=s(12,2) ; s(15,2)=s(11,1) ; s(16,1)=s(12,2) ; s(16,2)=s(11,2)

wt(1)=.144315607677787 ; wt(2)=.103217370534718 ; wt(3)=wt(2); wt(4)=wt(2)

wt(5)=.032458497623198 ; wt(6)=wt(5) ; wt(7)=wt(5)

```
wt(8)=.095091634267284 ; wt(9)=wt(8)  ;  wt(10)=wt(8)
wt(11)=.027230314174435 ; wt(12:16) = wt(11)  ;    wt = .5*wt
        case default
           print*,"wrong number of integrating points for a triangle"
        end select
       case ('quadrilateral')
        select case (nip)
        case(1)
          s(1,1) = .0 ; wt(1) = 4.
        case(4)
          s(1,1)=-root3; s(1,2)= root3
          s(2,1)= root3; s(2,2)= root3
          s(3,1)=-root3; s(3,2)=-root3
          s(4,1)= root3; s(4,2)=-root3
          wt = 1.0
        case(9)
          s(1:7:3,1) = -r15; s(2:8:3,1) = .0
          s(3:9:3,1) =  r15; s(1:3,2)  = r15
          s(4:6,2)  =  .0 ; s(7:9,2)  =-r15
             wt= v
        case default
          print*,"wrong number of integrating points for a quadrilateral"
       end select
       case('tetrahedron')
        select case(nip)
        case(1)        ! for tetrahedra weights multiplied by 1/6
          s(1,1)=.25   ; s(1,2)=.25  ;  s(1,3)=.25   ; wt(1)=1./6.
        case(4)
          s(1,1)=.58541020 ; s(1,2)=.13819660  ;  s(1,3)=s(1,2)
          s(2,2)=s(1,1) ; s(2,3)=s(1,2)  ;  s(2,1)=s(1,2)
          s(3,3)=s(1,1) ; s(3,1)=s(1,2)  ;  s(3,2)=s(1,2)
```

```
     s(4,1)=s(1,2) ; s(4,2)=s(1,2)  ;  s(4,3)=s(1,2) ; wt(1:4)=.25/6.
    case(5)
     s(1,1)=.25  ;  s(1,2)=.25   ; s(1,3)=.25 ;  s(2,1)=.5
     s(2,2)=1./6. ;  s(2,3)=s(2,2);  s(3,2)=.5
     s(3,3)=1./6.  ;   s(3,1)=s(3,3)   ;   s(4,3)=.5
     s(4,1)=1./6. ;    s(4,2)=s(4,1);    s(5,1)=1./6.
     s(5,2)=s(5,1) ;  s(5,3)=s(5,1)
     wt(1)=-.8  ;  wt(2)=9./20. ;   wt(3:5)=wt(2)   ; wt =wt/6.
    case(6)
wt = 4./3.       ;  s(6,3) = 1.
s(1,1)=-1. ;s(2,1)=1. ; s(3,2)=-1. ; s(4,2)=1. ;  s(5,3)=-1.
    case default
     print*,"wrong number of integrating points for a tetrahedron"
  end select
  case('hexahedron')
   select case ( nip )
    case(1)
        s(1,1) = .0 ; wt(1) = 8.
    case(8)
        s(1,1)= root3;s(1,2)= root3;s(1,3)= root3
        s(2,1)= root3;s(2,2)= root3;s(2,3)=-root3
        s(3,1)= root3;s(3,2)=-root3;s(3,3)= root3
        s(4,1)= root3;s(4,2)=-root3;s(4,3)=-root3
        s(5,1)=-root3;s(5,2)= root3;s(5,3)= root3
        s(6,1)=-root3;s(6,2)=-root3;s(6,3)= root3
        s(7,1)=-root3;s(7,2)= root3;s(7,3)=-root3
        s(8,1)=-root3;s(8,2)=-root3;s(8,3)=-root3
        wt = 1.0
    case(14)
 b=0.795822426    ;     c=0.758786911
 wt(1:6)=0.886426593   ; wt(7:) =  0.335180055
```

```
s(1,1)=-b ; s(2,1)=b  ;  s(3,2)=-b ;   s(4,2)=b
s(5,3)=-b   ;    s(6,3)=b
s(7:,:) = c
s(7,1)=-c  ;  s(7,2)=-c  ; s(7,3)=-c ; s(8,2)=-c ;   s(8,3)=-c
s(9,1)=-c  ;  s(9,3)=-c  ; s(10,3)=-c; s(11,1)=-c
s(11,2)=-c ;  s(12,2)=-c ; s(13,1)=-c
   case(15)
b=1.    ;     c=0.674199862
wt(1)=1.564444444 ;  wt(2:7)=0.355555556  ; wt(8:15)=0.537777778
s(2,1)=-b  ;   s(3,1)=b  ;   s(4,2)=-b  ;   s(5,2)=b
s(6,3)=-b  ;   s(7,3)=b  ;   s(8:,:)=c  ;   s(8,1)=-c
s(8,2)=-c  ;   s(8,3)=-c ;   s(9,2)=-c  ;   s(9,3)=-c
s(10,1)=-c ;   s(10,3)=-c  ;  s(11,3)=-c ;   s(12,1)=-c
s(12,2)=-c ;   s(13,2)=-c  ;  s(14,1)=-c
   case(27)
        wt = (/5./9.*v,8./9.*v,5./9.*v/)
        s(1:7:3,1) = -r15; s(2:8:3,1) = .0
        s(3:9:3,1) =  r15; s(1:3,3)   = r15
        s(4:6,3)   =  .0 ; s(7:9,3)   =-r15
        s(1:9,2)   = -r15
        s(10:16:3,1) = -r15; s(11:17:3,1) = .0
        s(12:18:3,1) =  r15; s(10:12,3)   = r15
        s(13:15,3)   =  .0 ; s(16:18,3)   =-r15
        s(10:18,2)   = .0
        s(19:25:3,1) = -r15; s(20:26:3,1) = .0
        s(21:27:3,1) =  r15; s(19:21,3)   = r15
        s(22:24,3)   =  .0 ; s(25:27,3)   =-r15
        s(19:27,2)   =  r15
    case default
      print*,"wrong number of integrating points for a hexahedron"
  end select
```

```
        case default
         print*,"not a valid element type"
   end select
  return
end subroutine sample

subroutine shape_der(der,points,i)
implicit none
integer,intent(in):: i; doubleprecision,intent(in)::points(:,:)
doubleprecision,intent(out)::der(:,:)
doubleprecision::eta,xi,zeta,xi0,eta0,zeta0,etam,etap,xim,xip,c1,c2,c3 ! local variables
 doubleprecision:: t1,t2,t3,t4,t5,t6,t7,t8,t9 ,x2p1,x2m1,e2p1,e2m1,zetam,zetap,x,y,z
 integer :: xii(20), etai(20), zetai(20) ,l,ndim , nod   ! local variables
 ndim = ubound(der , 1); nod = ubound(der , 2)
 select case (ndim)
  case(1) ! one dimensional case
      xi=points(i,1)
   select case (nod)
      case(2)
        der(1,1)=-0.5 ; der(1,2)=0.5
      case(3)
        t1=-1.-xi ; t2=-xi  ; t3=1.-xi
        der(1,1)=-(t3+t2)/2.  ; der(1,2)=(t3+t1)
        der(1,3)=-(t2+t1)/2.
      case(4)
        t1=-1.-xi ; t2=-1./3.-xi ; t3=1./3.-xi ; t4=1.-xi
        der(1,1)=-(t3*t4+t2*t4+t2*t3)*9./16.
        der(1,2)=(t3*t4+t1*t4+t1*t3)*27./16.
        der(1,3)=-(t2*t4+t1*t4+t1*t2)*27./16.
        der(1,4)=(t2*t3+t1*t3+t1*t2)*9./16.
      case(5)
```

```
      t1=-1.-xi ; t2=-0.5-xi ; t3=-xi ; t4=0.5-xi ; t5=1.-xi
      der(1,1)=-(t3*t4*t5+t2*t4*t5+t2*t3*t5+t2*t3*t4)*2./3.
      der(1,2)=(t3*t4*t5+t1*t4*t5+t1*t3*t5+t1*t3*t4)*8./3.
      der(1,3)=-(t2*t4*t5+t1*t4*t5+t1*t2*t5+t1*t2*t4)*4.
      der(1,4)=(t2*t3*t5+t1*t3*t5+t1*t2*t5+t1*t2*t3)*8./3.
      der(1,5)=-(t2*t3*t4+t1*t3*t4+t1*t2*t4+t1*t2*t3)*2./3.
  case default
   print*,"wrong number of nodes in shape_der"
  end select
 case(2)     ! two dimensional elements
   xi=points(i,1); eta=points(i,2) ; c1=xi ; c2=eta ; c3=1.-c1-c2
   etam=.25*(1.-eta); etap=.25*(1.+eta); xim=.25*(1.-xi); xip=.25*(1.+xi)
   x2p1=2.*xi+1. ;   x2m1=2.*xi-1. ;  e2p1=2.*eta+1. ;   e2m1=2.*eta-1.
  select case (nod)
   case(3)
   der(1,1)=1.;der(1,3)=0.;der(1,2)=-1.
   der(2,1)=0.;der(2,3)=1.;der(2,2)=-1.
   case(6)
   der(1,1)=4.*c1-1. ;  der(1,6)=4.*c2;  der(1,5)=0.  ; der(1,4)=-4.*c2
   der(1,3)=-(4.*c3-1.);  der(1,2)=4.*(c3-c1);   der(2,1)=0.
   der(2,6)=4.*c1 ; der(2,5)=4.*c2-1.; der(2,4)=4.*(c3-c2)
   der(2,3)=-(4.*c3-1.)  ; der(2,2)=-4.*c1
   case(15)
   t1=c1-.25  ;  t2=c1-.5 ;  t3=c1-.75   ;   t4=c2-.25
   t5=c2-.5   ;  t6=c2-.75 ;  t7=c3-.25  ;   t8=c3-.5 ;  t9=c3-.75
   der(1,1)=32./3.*(t2*t3*(t1+c1)+c1*t1*(t3+t2))
   der(1,12)=128./3.*c2*(t2*(t1+c1)+c1*t1) ;  der(1,11)=64.*c2*t4*(t1+c1)
   der(1,10)=128./3.*c2*t4*t5  ; der(1,9)=0. ; der(1,8)=-128./3.*c2*t4*t5
   der(1,7)=-64.*c2*t4*(t7+c3) ; der(1,6)=-128./3.*c2*(t8*(t7+c3)+c3*t7)
   der(1,5)=-32./3.*(t8*t9*(t7+c3)+c3*t7*(t8+t9))
   der(1,4)=128./3.*(c3*t7*t8-c1*(t8*(t7+c3)+c3*t7))
```

```
 der(1,3)=64.*(c3*t7*(t1+c1)-c1*t1*(t7+c3))
 der(1,2)=128./3.*(c3*(t2*(t1+c1)+c1*t1)-c1*t1*t2)
 der(1,13)=128.*c2*(c3*(t1+c1)-c1*t1) ;  der(1,15)=128.*c2*t4*(c3-c1)
 der(1,14)=128.*c2*(c3*t7-c1*(t7+c3))
 der(2,1)=0.0 ;  der(2,12)=128./3.*c1*t1*t2;  der(2,11)=64.*c1*t1*(t4+c2)
 der(2,10)=128./3.*c1*(t5*(t4+c2)+c2*t4)
 der(2,9)=32./3.*(t5*t6*(t4+c2)+c2*t4*(t6+t5))
 der(2,8)=128./3.*((c3*(t5*(t4+c2)+c2*t4))-c2*t4*t5)
 der(2,7)=64.*(c3*t7*(t4+c2)-c2*t4*(t7+c3))
 der(2,6)=128./3.*(c3*t7*t8-c2*(t8*(t7+c3)+c3*t7))
 der(2,5)=-32./3.*(t8*t9*(t7+c3)+c3*t7*(t8+t9))
 der(2,4)=-128./3.*c1*(t8*(t7+c3)+c3*t7)
 der(2,3)=-64.*c1*t1*(t7+c3)  ;  der(2,2)=-128./3.*c1*t1*t2
 der(2,13)=128.*c1*t1*(c3-c2)
 der(2,15)=128.*c1*(c3*(t4+c2)-c2*t4)
 der(2,14)=128.*c1*(c3*t7-c2*(c3+t7))
case (4)
 der(1,1)=-etam; der(1,2)=-etap; der(1,3)=etap; der(1,4)=etam
 der(2,1)=-xim; der(2,2)=xim; der(2,3)=xip; der(2,4)=-xip
case(8)
 der(1,1)=etam*(2.*xi+eta); der(1,2)=-8.*etam*etap
 der(1,3)=etap*(2.*xi-eta); der(1,4)=-4.*etap*xi
 der(1,5)=etap*(2.*xi+eta); der(1,6)=8.*etap*etam
 der(1,7)=etam*(2.*xi-eta); der(1,8)=-4.*etam*xi
 der(2,1)=xim*(xi+2.*eta); der(2,2)=-4.*xim*eta
 der(2,3)=xim*(2.*eta-xi); der(2,4)=8.*xim*xip
 der(2,5)=xip*(xi+2.*eta); der(2,6)=-4.*xip*eta
 der(2,7)=xip*(2.*eta-xi); der(2,8)=-8.*xim*xip
case(9)
 etam = eta - 1.; etap = eta + 1.; xim = xi - 1.; xip = xi + 1.
 der(1,1)=.25*x2m1*eta*etam  ;   der(1,2)=-.5*x2m1*etap*etam
```

```
   der(1,3)=.25*x2m1*eta*etap  ;   der(1,4)=-xi*eta*etap
   der(1,5)=.25*x2p1*eta*etap  ;   der(1,6)=-.5*x2p1*etap*etam
   der(1,7)=.25*x2p1*eta*etam  ;   der(1,8)=-xi*eta*etam
   der(1,9)=2.*xi*etap*etam    ;   der(2,1)=.25*xi*xim*e2m1
   der(2,2)=-xi*xim*eta        ;   der(2,3)=.25*xi*xim*e2p1
   der(2,4)=-.5*xip*xim*e2p1   ;   der(2,5)=.25*xi*xip*e2p1
   der(2,6)=-xi*xip*eta        ;   der(2,7)=.25*xi*xip*e2m1
   der(2,8)=-.5*xip*xim*e2m1   ;   der(2,9)=2.*xip*xim*eta
  case default
   print*,"wrong number of nodes in shape_der"
  end select
 case(3)  ! three dimensional elements
   xi=points(i,1); eta=points(i,2); zeta=points(i,3)
   etam=1.-eta ; xim=1.-xi;  zetam=1.-zeta
   etap=eta+1. ; xip=xi+1. ;  zetap=zeta+1.
 select case (nod)
  case(4)
  der(1:3,1:4) = .0
  der(1,1)=1.;  der(2,2)=1.  ;  der(3,3)=1.
  der(1,4)=-1. ;  der(2,4)=-1. ;  der(3,4)=-1.
  case(8)
  der(1,1)=-.125*etam*zetam    ;   der(1,2)=-.125*etam*zetap
  der(1,3)=.125*etam*zetap     ;   der(1,4)=.125*etam*zetam
  der(1,5)=-.125*etap*zetam    ;   der(1,6)=-.125*etap*zetap
  der(1,7)=.125*etap*zetap     ;   der(1,8)=.125*etap*zetam
  der(2,1)=-.125*xim*zetam     ;   der(2,2)=-.125*xim*zetap
  der(2,3)=-.125*xip*zetap     ;   der(2,4)=-.125*xip*zetam
  der(2,5)=.125*xim*zetam      ;   der(2,6)=.125*xim*zetap
  der(2,7)=.125*xip*zetap      ;   der(2,8)=.125*xip*zetam
  der(3,1)=-.125*xim*etam      ;   der(3,2)=.125*xim*etam
  der(3,3)=.125*xip*etam       ;   der(3,4)=-.125*xip*etam
```

```
    der(3,5)=-.125*xim*etap     ;   der(3,6)=.125*xim*etap
    der(3,7)=.125*xip*etap      ;   der(3,8)=-.125*xip*etap
  case(14) ! type 6 element
   x= points(i,1)   ;   y= points(i,2)  ;    z= points(i,3)
   der(1,1)=((2.*x*y+2.*x*z+4.*x+y*z+y+z)*(y-1.)*(z-1.))/8.
   der(1,2)=((2.*x*y-2.*x*z-4.*x+y*z+y-z)*(y+1.)*(z-1.))/8.
   der(1,3)=((2.*x*y+2.*x*z+4.*x-y*z-y-z)*(y-1.)*(z-1.))/8.
   der(1,4)=((2.*x*y-2.*x*z-4.*x-y*z-y+z)*(y+1.)*(z-1.))/8.
   der(1,5)=-((2.*x*y-2.*x*z+4.*x-y*z+y-z)*(y-1.)*(z+1.))/8.
   der(1,6)=-((2.*x*y+2.*x*z-4.*x-y*z+y+z)*(y+1.)*(z+1.))/8.
   der(1,7)=-((2.*x*y-2.*x*z+4.*x+y*z-y+z)*(y-1.)*(z+1.))/8.
   der(1,8)=-((2.*x*y+2.*x*z-4.*x+y*z-y-z)*(y+1.)*(z+1.))/8.
   der(1,9)=-(y+1.)*(y-1.)*(z-1.)*x  ;   der(1,10)=(y+1.)*(y-1.)*(z+1.)*x
   der(1,11)=-(y-1.)*(z+1.)*(z-1.)*x ;   der(1,12)=(y+1.)*(z+1.)*(z-1.)*x
   der(1,13)=-((y+1.)*(y-1.)*(z+1.)*(z-1.))/2.
   der(1,14)=((y+1.)*(y-1.)*(z+1.)*(z-1.))/2.
   der(2,1)=((2.*x*y+x*z+x+2.*y*z+4.*y+z)*(x-1.)*(z-1.))/8.
   der(2,2)=((2.*x*y-x*z-x+2.*y*z+4.*y-z)*(x-1.)*(z-1.))/8.
   der(2,3)=((2.*x*y+x*z+x-2.*y*z-4.*y-z)*(x+1.)*(z-1.))/8.
   der(2,4)=((2.*x*y-x*z-x-2.*y*z-4.*y+z)*(x+1.)*(z-1.))/8.
   der(2,5)=-((2.*x*y-x*z+x-2.*y*z+4.*y-z)*(x-1.)*(z+1.))/8.
   der(2,6)=-((2.*x*y+x*z-x-2.*y*z+4.*y+z)*(x-1.)*(z+1.))/8.
   der(2,7)=-((2.*x*y-x*z+x+2.*y*z-4.*y+z)*(x+1.)*(z+1.))/8.
   der(2,8)=-((2.*x*y+x*z-x+2.*y*z-4.*y-z)*(x+1.)*(z+1.))/8.
   der(2,9)=-(x+1.)*(x-1.)*(z-1.)*y
   der(2,10)=(x+1.)*(x-1.)*(z+1.)*y
   der(2,11)=-((x+1.)*(x-1.)*(z+1.)*(z-1.))/2.
   der(2,12)=((x+1.)*(x-1.)*(z+1.)*(z-1.))/2.
   der(2,13)=-(x-1.)*(z+1.)*(z-1.)*y
   der(2,14)=(x+1.)*(z+1.)*(z-1.)*y
   der(3,1)=((x*y+2.*x*z+x+2.*y*z+y+4.*z)*(x-1.)*(y-1.))/8.
```

```
der(3,2)=((x*y-2.*x*z-x+2.*y*z+y-4.*z)*(x-1.)*(y+1.))/8.
der(3,3)=((x*y+2.*x*z+x-2.*y*z-y-4.*z)*(x+1.)*(y-1.))/8.
der(3,4)=((x*y-2.*x*z-x-2.*y*z-y+4.*z)*(x+1.)*(y+1.))/8.
der(3,5)=-((x*y-2.*x*z+x-2.*y*z+y-4.*z)*(x-1.)*(y-1.))/8.
der(3,6)=-((x*y+2.*x*z-x-2.*y*z+y+4.*z)*(x-1.)*(y+1.))/8.
der(3,7)=-((x*y-2.*x*z+x+2.*y*z-y+4.*z)*(x+1.)*(y-1.))/8.
der(3,8)=-((x*y+2.*x*z-x+2.*y*z-y-4.*z)*(x+1.)*(y+1.))/8.
der(3,9)=-((x+1.)*(x-1.)*(y+1.)*(y-1.))/2.
der(3,10)=((x+1.)*(x-1.)*(y+1.)*(y-1.))/2.
der(3,11)=-(x+1.)*(x-1.)*(y-1.)*z  ; der(3,12)=(x+1.)*(x-1.)*(y+1.)*z
der(3,13)=-(x-1.)*(y+1.)*(y-1.)*z  ; der(3,14)=(x+1.)*(y+1.)*(y-1.)*z
 case(20)
  xii=(/-1,-1,-1,0,1,1,1,0,-1,-1,1,1,-1,-1,-1,0,1,1,1,0/)
  etai=(/-1,-1,-1,-1,-1,-1,-1,-1,0,0,0,0,1,1,1,1,1,1,1,1/)
  zetai=(/-1,0,1,1,1,0,-1,-1,-1,1,1,-1,-1,0,1,1,1,0,-1,-1/)
  do l=1,20
    xi0=xi*xii(l); eta0=eta*etai(l); zeta0=zeta*zetai(l)
    if(l==4.or.l==8.or.l==16.or.l==20) then
     der(1,l)=-.5*xi*(1.+eta0)*(1.+zeta0)
     der(2,l)=.25*etai(l)*(1.-xi*xi)*(1.+zeta0)
     der(3,l)=.25*zetai(l)*(1.-xi*xi)*(1.+eta0)
    else if(l>=9.and.l<=12)then
     der(1,l)=.25*xii(l)*(1.-eta*eta)*(1.+zeta0)
     der(2,l)=-.5*eta*(1.+xi0)*(1.+zeta0)
     der(3,l)=.25*zetai(l)*(1.+xi0)*(1.-eta*eta)
    else if(l==2.or.l==6.or.l==14.or.l==18) then
     der(1,l)=.25*xii(l)*(1.+eta0)*(1.-zeta*zeta)
     der(2,l)=.25*etai(l)*(1.+xi0)*(1.-zeta*zeta)
     der(3,l)=-.5*zeta*(1.+xi0)*(1.+eta0)
    else
     der(1,l)=.125*xii(l)*(1.+eta0)*(1.+zeta0)*(2.*xi0+eta0+zeta0-1.)
```

```
      der(2,l)=.125*etai(l)*(1.+xi0)*(1.+zeta0)*(xi0+2.*eta0+zeta0-1.)
      der(3,l)=.125*zetai(l)*(1.+xi0)*(1.+eta0)*(xi0+eta0+2.*zeta0-1.)
     end if
   end do
  case default
   print*,"wrong number of nodes in shape_der"
 end select
case default
 print*,"wrong number of dimensions in shape_der"
end select
return
end subroutine shape_der

subroutine shape_fun(fun,points,i)
 implicit none
 integer,intent(in):: i; doubleprecision,intent(in)::points(:,:)
 doubleprecision,intent(out)::fun(:)
 doubleprecision :: eta,xi,etam,etap,xim,xip,zetam,zetap,c1,c2,c3     !local variables
 doubleprecision :: t1,t2,t3,t4,t5,t6,t7,t8,t9,x,y,z
 doubleprecision :: zeta,xi0,eta0,zeta0; integer::xii(20),etai(20),zetai(20),l,ndim,nod
     ndim = ubound(points , 2 ); nod = ubound(fun , 1 )
  select case (ndim)
   case(1) ! one dimensional cases
      xi=points(i,1)
    select case(nod)
     case(2)
      t1=-1.-xi ; t2=1.-xi
      fun(1)=t2/2. ; fun(2)=-t1/2.
     case(3)
      t1=-1.-xi ; t2=-xi ; t3=1.-xi
      fun(1)=t2*t3/2. ; fun(2)=-t1*t3 ; fun(3)=t1*t2/2.
```

```
 case(4)
  t1=-1.-xi ; t2=-1./3.-xi ; t3=1./3.-xi ; t4=1.-xi
  fun(1)=t2*t3*t4*9./16.  ; fun(2)=-t1*t3*t4*27./16.
  fun(3)=t1*t2*t4*27./16. ; fun(4)=-t1*t2*t3*9./16.
 case(5)
  t1=-1.-xi ; t2=-0.5-xi ; t3=-xi ; t4=0.5-xi ; t5=1.-xi
  fun(1)=t2*t3*t4*t5*2./3. ; fun(2)=-t1*t3*t4*t5*8./3.
  fun(3)=t1*t2*t4*t5*4. ; fun(4)=-t1*t2*t3*t5*8./3.
  fun(5)=t1*t2*t3*t4*2./3.
 case default
   print*,"wrong number of nodes in shape_fun"
end select
case(2) ! two dimensional cases
  c1=points(i,1); c2=points(i,2); c3=1.-c1-c2
  xi=points(i,1);  eta=points(i,2)
  etam=.25*(1.-eta); etap=.25*(1.+eta)
  xim=.25*(1.-xi); xip=.25*(1.+xi)
 select case(nod)
  case(3)
   fun = (/c1,c3,c2/)
  case(6)
   fun(1)=(2.*c1-1.)*c1 ;  fun(6)=4.*c1*c2 ;  fun(5)=(2.*c2-1.)*c2
   fun(4)=4.*c2*c3     ;  fun(3)=(2.*c3-1.)*c3 ; fun(2)=4.*c3*c1
  case(15)
   t1=c1-.25  ;  t2=c1-.5 ;  t3=c1-.75  ;  t4=c2-.25
   t5=c2-.5  ;  t6=c2-.75 ;  t7=c3-.25  ;  t8=c3-.5 ;  t9=c3-.75
   fun(1)=32./3.*c1*t1*t2*t3  ;  fun(12)=128./3.*c1*c2*t1*t2
   fun(11)=64.*c1*c2*t1*t4    ;  fun(10)=128./3.*c1*c2*t4*t5
   fun(9)=32./3.*c2*t4*t5*t6  ;  fun(8)=128./3.*c2*c3*t4*t5
   fun(7)=64.*c2*c3*t4*t7     ;  fun(6)=128./3.*c2*c3*t7*t8
   fun(5)=32./3.*c3*t7*t8*t9  ;  fun(4)=128./3.*c3*c1*t7*t8
```

```
   fun(3)=64.*c3*c1*t1*t7     ;  fun(2)=128./3.*c3*c1*t1*t2
   fun(13)=128.*c1*c2*t1*c3   ;  fun(15)=128.*c1*c2*c3*t4
   fun(14)=128.*c1*c2*c3*t7
  case(4)
   fun=(/4.*xim*etam,4.*xim*etap,4.*xip*etap,4.*xip*etam/)
  case(8)
   fun=(/4.*etam*xim*(-xi-eta-1.),32.*etam*xim*etap,&
        4.*etap*xim*(-xi+eta-1.),32.*xim*xip*etap, &
        4.*etap*xip*(xi+eta-1.), 32.*etap*xip*etam,&
        4.*xip*etam*(xi-eta-1.), 32.*xim*xip*etam/)
  case(9)
   etam = eta - 1.; etap= eta + 1.; xim = xi - 1.; xip = xi + 1.
   fun=(/.25*xi*xim*eta*etam,-.5*xi*xim*etap*etam,&
        .25*xi*xim*eta*etap,-.5*xip*xim*eta*etap,&
        .25*xi*xip*eta*etap,-.5*xi*xip*etap*etam,&
        .25*xi*xip*eta*etam,-.5*xip*xim*eta*etam,xip*xim*etap*etam/)
  case default
    print*,"wrong number of nodes in shape_fun"
 end select
case(3) ! three dimensional cases
 xi=points(i,1); eta=points(i,2); zeta=points(i,3)
 etam=1.-eta ;  xim=1.-xi  ;  zetam=1.-zeta
 etap=eta+1. ;  xip=xi+1.   ;  zetap=zeta+1.
 select case(nod)
  case(4)
  fun(1)=xi   ;   fun(2)= eta ;  fun(3)=zeta
  fun(4)=1.-fun(1)-fun(2)-fun(3)
  case(8)
  fun=(/.125*xim*etam*zetam,.125*xim*etam*zetap,.125*xip*etam*zetap,&
       .125*xip*etam*zetam,.125*xim*etap*zetam,.125*xim*etap*zetap,&
       .125*xip*etap*zetap,.125*xip*etap*zetam/)
```

```
      case(14) !type 6 element
    x = points(i,1);  y = points(i,2);  z = points(i,3)
  fun(1)=((x*y+x*z+2.*x+y*z+2.*y+2.*z+2.)*(x-1.)*(y-1.)*(z-1.))/8.
  fun(2)=((x*y-x*z-2.*x+y*z+2.*y-2.*z-2.)*(x-1.)*(y+1.)*(z-1.))/8.
  fun(3)=((x*y+x*z+2.*x-y*z-2.*y-2.*z-2.)*(x+1.)*(y-1.)*(z-1.))/8.
  fun(4)=((x*y-x*z-2.*x-y*z-2.*y+2.*z+2.)*(x+1.)*(y+1.)*(z-1.))/8.
  fun(5)=-((x*y-x*z+2.*x-y*z+2.*y-2.*z+2.)*(x-1.)*(y-1.)*(z+1.))/8.
  fun(6)=-((x*y+x*z-2.*x-y*z+2.*y+2.*z-2.)*(x-1.)*(y+1.)*(z+1.))/8.
  fun(7)=-((x*y-x*z+2.*x+y*z-2.*y+2.*z-2.)*(x+1.)*(y-1.)*(z+1.))/8.
  fun(8)=-((x*y+x*z-2.*x+y*z-2.*y-2.*z+2.)*(x+1.)*(y+1.)*(z+1.))/8.
  fun(9)=-((x+1.)*(x-1.)*(y+1.)*(y-1.)*(z-1.))/2.
  fun(10)=((x+1.)*(x-1.)*(y+1.)*(y-1.)*(z+1.))/2.
  fun(11)=-((x+1.)*(x-1.)*(y-1.)*(z+1.)*(z-1.))/2.
  fun(12)=((x+1.)*(x-1.)*(y+1.)*(z+1.)*(z-1.))/2.
  fun(13)=-((x-1.)*(y+1.)*(y-1.)*(z+1.)*(z-1.))/2.
  fun(14)=((x+1.)*(y+1.)*(y-1.)*(z+1.)*(z-1.))/2.
      case(20)
        xii=(/-1,-1,-1,0,1,1,1,0,-1,-1,1,1,-1,-1,-1,0,1,1,1,0/)
        etai=(/-1,-1,-1,-1,-1,-1,-1,-1,0,0,0,0,1,1,1,1,1,1,1,1/)
        zetai=(/-1,0,1,1,1,0,-1,-1,-1,1,1,-1,-1,0,1,1,1,0,-1,-1/)
        do l=1,20
         xi0=xi*xii(l); eta0=eta*etai(l); zeta0=zeta*zetai(l)
         if(l==4.or.l==8.or.l==16.or.l==20) then
          fun(l)=.25*(1.-xi*xi)*(1.+eta0)*(1.+zeta0)
         else if(l>=9.and.l<=12)then
          fun(l)=.25*(1.+xi0)*(1.-eta*eta)*(1.+zeta0)
         else if(l==2.or.l==6.or.l==14.or.l==18) then
          fun(l)=.25*(1.+xi0)*(1.+eta0)*(1.-zeta*zeta)
         else
          fun(l)=.125*(1.+xi0)*(1.+eta0)*(1.+zeta0)*(xi0+eta0+zeta0-2)
         end if
```

```
        end do
       case default
        print*,"wrong number of nodes in shape_fun"
      end select
    case default
     print*,"wrong number of dimensions in shape_fun"
  end select
 return
end subroutine shape_fun

subroutine formkv(bk,km,g,n)
!global stiffness matrix stored as a vector (upper triangle)
implicit none
doubleprecision,intent(in)::km(:,:);doubleprecision,intent(out)::bk(:)
integer,intent(in)::g(:),n
integer::idof,i,j,icd,ival
idof=size(km,1)
   do i=1,idof
     if(g(i)/=0) then
       do j=1,idof
         if(g(j)/=0) then
           icd=g(j)-g(i)+1
           if(icd-1>=0) then
             ival=n*(icd-1)+g(i)
             bk(ival)=bk(ival)+km(i,j)
           end if
         end if
       end do
     end if
   end do
return
```

```
end subroutine formkv

subroutine fsparv(bk,km,g,kdiag)
! assembly of element matrices into skyline global matrix
implicit none
doubleprecision,intent(in)::km(:,:); integer,intent(in)::g(:),kdiag(:)
doubleprecision,intent(out)::bk(:) ;  integer::i,idof,k,j,iw,ival
 idof=ubound(g,1)
  do i=1,idof
    k=g(i)
    if(k/=0) then
      do j=1,idof
        if(g(j)/=0) then
          iw=k-g(j)
          if(iw>=0) then
            ival=kdiag(k)-iw
            bk(ival)=bk(ival)+km(i,j)
          end if
        end if
      end do
    end if
  end do
 return
end subroutine fsparv

subroutine banred(bk,n)
! gaussian reduction on a vector stored as an upper triangle
implicit none
doubleprecision,intent(in out)::bk(:);integer,intent(in)::n
integer::i,il1,kbl,j,ij,nkb,m,ni,nj,iw ; doubleprecision::sum
 iw = ubound(bk,1)/n-1
```

```
    do i=2,n
      il1=i-1;kbl=il1+iw+1
      if(kbl-n>0)kbl=n
      do j=i,kbl
        ij=(j-i)*n+i;sum=bk(ij);nkb=j-iw
        if(nkb<=0)nkb=1
        if(nkb-il1<=0)then
          do m=nkb,il1
            ni=(i-m)*n+m ; nj=(j-m)*n+m
            sum=sum-bk(ni)*bk(nj)/bk(m)
          end do
        end if
        bk(ij)=sum
      end do
    end do
return
end subroutine banred

subroutine bacsub(bk,loads)
! performs the complete gaussian backsubstitution
implicit none
doubleprecision,intent(in)::bk(:);doubleprecision,intent(in out)::loads(0:)
integer::nkb,k,i,jn,jj,i1,n,iw;doubleprecision::sum
n = ubound(loads,1); iw = ubound(bk,1)/n - 1
loads(1)=loads(1)/bk(1)
  do i=2,n
    sum=loads(i);i1=i-1 ; nkb=i-iw
    if(nkb<=0)nkb=1
    do k=nkb,i1
      jn=(i-k)*n+k;sum=sum-bk(jn)*loads(k)
    end do
```

```
    loads(i)=sum/bk(i)
  end do
  do jj=2,n
    i=n-jj+1;sum=.0;i1=i+1;nkb=i+iw
    if(nkb-n>0)nkb=n
    do k=i1,nkb
       jn=(k-i)*n+i  ; sum=sum+bk(jn)*loads(k)
    end do
    loads(i)=loads(i)-sum/bk(i)
  end do
return
end subroutine bacsub

subroutine sparin(a,kdiag)
! Choleski factorisation of variable bandwidth matrix a
! stored as a vector and overwritten
implicit none
doubleprecision,intent(in out)::a(:);integer,intent(in)::kdiag(:)
integer::n,i,ki,l,kj,j,ll,m,k; doubleprecision::x
 n=ubound(kdiag,1)  ; a(1)=sqrt(a(1))
 do i=2,n
   ki=kdiag(i)-i;  l=kdiag(i-1)-ki+1
   do j=l,i
     x=a(ki+j);  kj=kdiag(j)-j
     if(j/=1) then
       ll=kdiag(j-1)-kj+1; ll=max0(l,ll)
       if(ll/=j) then
          m=j-1
          do k=ll,m ; x=x-a(ki+k)*a(kj+k) ; end do
       end if
     end if
```

```
    a(ki+j)=x/a(kj+j)
   end do
   a(ki+i)=sqrt(x)
 end do
 return
end subroutine sparin

subroutine spabac(a,b,kdiag)
! Choleski forward and backward substitution combined
! variable bandwidth factorised matrix a stored as a vector
implicit none
doubleprecision,intent(in)::a(:);doubleprecision,intent(in
out)::b(0:);integer,intent(in)::kdiag(:)
integer::n,i,ki,l,m,j,it,k; doubleprecision::x
n=ubound(kdiag,1)
 b(1)=b(1)/a(1)
 do i=2,n
   ki=kdiag(i)-i;  l=kdiag(i-1)-ki+1 ; x=b(i)
   if(l/=i) then
     m=i-1
     do j=l,m ; x=x-a(ki+j)*b(j); end do
   end if
   b(i)=x/a(ki+i)
 end do
 do it=2,n
   i=n+2-it; ki=kdiag(i)-i; x=b(i)/a(ki+i); b(i)=x; l=kdiag(i-1)-ki+1
   if(l/=i) then
     m=i-1
     do k=l,m; b(k)=b(k)-x*a(ki+k); end do
   end if
 end do
```

```
 b(1)=b(1)/a(1)
 return
end subroutine spabac

subroutine formkb(kb,km,g)
! lower triangular global stiffness kb stored as kb(n,iw+1)
implicit none
doubleprecision,intent(in)::km(:,:);doubleprecision,intent(out)::kb(:,:)
integer,intent(in)::g(:);integer::iw,idof,i,j,icd
idof=size(km,1);  iw=size(kb,2)-1
  do i=1,idof
    if(g(i)>0) then
      do j=1,idof
        if(g(j)>0) then
          icd=g(j)-g(i)+iw+1
          if(icd-iw-1<=0) kb(g(i),icd)= kb(g(i),icd) +km(i,j)
        end if
      end do
    end if
  end do
 return
end subroutine formkb

subroutine fkdiag(kdiag,g)
! finds the maximum bandwidth for each freedom
implicit none
 integer,intent(in)::g(:); integer,intent(out)::kdiag(:)
 integer::idof,i,iwp1,j,im,k
  idof=size(g)
  do i = 1,idof
    iwp1=1
```

```
   if(g(i)/=0) then
     do j=1,idof
       if(g(j)/=0) then
         im=g(i)-g(j)+1
         if(im>iwp1) iwp1=im
       end if
     end do
     k=g(i);   if(iwp1>kdiag(k))kdiag(k)=iwp1
   end if
 end do
 return
end subroutine fkdiag

subroutine invar(stress,sigm,dsbar,theta)
! forms the stress invariants in 2-d or 3-d
implicit none
 doubleprecision,intent(in)::stress(:)
 doubleprecision,intent(out)::sigm,dsbar,theta
 doubleprecision::sx,sy,sz,txy,dx,dy,dz,xj3,sine,s1,s2,s3,s4,s5,s6,ds1,ds2,ds3,d2,d3,sq3
 integer :: nst ; nst = ubound(stress,1)
 select case (nst)
 case(4)
  sx=stress(1); sy=stress(2); txy=stress(3); sz=stress(4)
  sigm=(sx+sy+sz)/3.
  dsbar=sqrt((sx-sy)**2+(sy-sz)**2+(sz-sx)**2+6.*txy**2)/sqrt(2.)
  if(dsbar<1.e-10) then
    theta=.0
  else
    dx=(2.*sx-sy-sz)/3.; dy=(2.*sy-sz-sx)/3.; dz=(2.*sz-sx-sy)/3.
    xj3=dx*dy*dz-dz*txy**2
    sine=-13.5*xj3/dsbar**3
```

```
   if(sine>1.) sine=1.
   if(sine<-1.) sine=-1.
   theta=asin(sine)/3.
 end if
case(6)
 sq3=sqrt(3.);  s1=stress(1)  ;  s2=stress(2)
 s3=stress(3) ;  s4=stress(4);  s5=stress(5);  s6=stress(6)
 sigm=(s1+s2+s3)/3.
 d2=((s1-s2)**2+(s2-s3)**2+(s3-s1)**2)/6.+s4*s4+s5*s5+s6*s6
 ds1=s1-sigm ;  ds2=s2-sigm  ;  ds3=s3-sigm
 d3=ds1*ds2*ds3-ds1*s5*s5-ds2*s6*s6-ds3*s4*s4+2.*s4*s5*s6
 dsbar=sq3*sqrt(d2)
 if(dsbar==0.)then
    theta=0.
  else
    sine=-3.*sq3*d3/(2.*sqrt(d2)**3)
    if(sine>1.)sine=1. ;  if(sine<-1.)sine=-1. ; theta=asin(sine)/3.
 end if
 case default
 print*,"wrong size for nst in invar"
 end select
 return
end subroutine invar

subroutine formm(stress,m1,m2,m3)
! forms the derivatives of the invariants with respect to stress 2- or 3-d
 implicit none
 doubleprecision,intent(in)::stress(:)
 doubleprecision,intent(out)::m1(:,:),m2(:,:),m3(:,:)
 doubleprecision::sx,sy,txy,tyz,tzx,sz,dx,dy,dz,sigm ; integer::nst , i , j
 nst=ubound(stress,1)
```

```
select case (nst)
case(4)
sx=stress(1); sy=stress(2); txy=stress(3); sz=stress(4)
dx=(2.*sx-sy-sz)/3.; dy=(2.*sy-sz-sx)/3.; dz=(2.*sz-sx-sy)/3.
sigm=(sx+sy+sz)/3.
m1=.0; m2=.0; m3=.0
m1(1,1:2)=1.; m1(2,1:2)=1.; m1(4,1:2)=1.
m1(1,4)=1.; m1(4,4)=1.; m1(2,4)=1.
m1=m1/9./sigm
m2(1,1)=.666666666666666; m2(2,2)=.666666666666666; m2(4,4)=.666666666666666
m2(2,4)=-.333333333333333;m2(4,2)=-.333333333333333;m2(1,2)=-.333333333333333
m2(2,1)=-.333333333333333;m2(1,4)=-.333333333333333;m2(4,1)=-.333333333333333
m2(3,3)=2.; m3(3,3)=-dz
m3(1:2,3)=txy/3.; m3(3,1:2)=txy/3.; m3(3,4)=-2.*txy/3.; m3(4,3)=-2.*txy/3.
m3(1,1)=dx/3.; m3(2,4)=dx/3.; m3(4,2)=dx/3.
m3(2,2)=dy/3.; m3(1,4)=dy/3.; m3(4,1)=dy/3.
m3(4,4)=dz/3.; m3(1,2)=dz/3.; m3(2,1)=dz/3.
case(6)
sx=stress(1); sy=stress(2)  ;  sz=stress(3)
txy=stress(4)  ;  tyz=stress(5) ;  tzx=stress(6)
sigm=(sx+sy+sz)/3.
dx=sx-sigm  ;  dy=sy-sigm ;  dz=sz-sigm
m1 = .0; m2 = .0; m1(1:3,1:3) = 1./(3.*sigm)
do  i=1,3 ; m2(i,i)=2. ;  m2(i+3,i+3)=6. ; end do
m2(1,2)=-1.; m2(1,3)=-1. ; m2(2,3)=-1.; m3(1,1)=dx
m3(1,2)=dz ; m3(1,3)=dy ; m3(1,4)=txy  ;  m3(1,5)=-2.*tyz
m3(1,6)=tzx ; m3(2,2)=dy ; m3(2,3)=dx ; m3(2,4)=txy
m3(2,5)=tyz ; m3(2,6)=-2.*tzx ;  m3(3,3)=dz
m3(3,4)=-2.*txy; m3(3,5)=tyz ;  m3(3,6)=tzx
m3(4,4)=-3.*dz ;  m3(4,5)=3.*tzx;  m3(4,6)=3.*tyz
m3(5,5)=-3.*dx;  m3(5,6)=3.*txy ;  m3(6,6)=-3.*dy
```

```
 do  i=1,6 ;  do  j=i,6
    m1(i,j)=m1(i,j)/3.;  m1(j,i)=m1(i,j) ;  m2(i,j)=m2(i,j)/3.
    m2(j,i)=m2(i,j)   ;  m3(i,j)=m3(i,j)/3. ; m3(j,i)=m3(i,j)
 end do; end do
case default
 print*,"wrong size for nst in formm"
end select
return
end subroutine formm
end module new_library
```

Appendix C: Source Codes of Feng Tan's Subroutine Library for HITSI

```
!------------------------------------------ Declaration ----------------------------------------------------
!---------------------- Source codes of Feng Tan's subroutine library for HITSI ----------------------
module lib_add
    !> \brief
    ! The lib-add is developing for extend the creep damage analysis capability
    ! of an in-house finite element analysis (FEA) software HITSI
    !  1. transformation of stress state (rdmpes)
    !  2. constitutive equations (KRH,KR)
    !  3. numerical methods (EULER,RK4)

  contains
  subroutine rdmpes (sigma,mpris,equs)
    !> \brief
    ! The RDMPES is used to return the value of stress deviator,
    ! maximum principal stress, and the equivalent stress.
    !!
    implicit none
    doubleprecision, intent(inout) :: sigma(:)!< stress component & deviator
    doubleprecision, intent(out) :: mpris,equs!< maximum principal stress
 & equivalent stress
    doubleprecision :: sx,sy,sz,txy,tyz,tzx,pi,j2,j3,sig0,loang
    doubleprecision, dimension(3) :: mps
    integer :: nst
    nst=ubound(sigma,1)
    pi=3.1415926
    select case (nst)
    !> \brief
```

```
! case 4 is 2D problem, and case 6 is 3D problem
!!
case (4)
   sig0=(sigma(1)+sigma(2)+sigma(4))/3
   sx=sigma(1); sy=sigma(2); txy=sigma(3); sz=sigma(4)
   j2=((sx-sy)**2+(sy-sz)**2+(sz-sx)**2)/6.+txy**2
   j3=(sx-sig0)*(sy-sig0)*(sz-sig0)-(sz-sig0)*txy**2
   loang=asin((-sqrt(27.)*j3)/(2*sqrt(j2**3)))/3
   mps(1)=2*sqrt(j2)/sqrt(3.)*sin(loang+2*pi/3)+sig0
   mps(2)=2*sqrt(j2)/sqrt(3.)*sin(loang)+sig0
   mps(3)=2*sqrt(j2)/sqrt(3.)*sin(loang-2*pi/3)+sig0
   mpris=maxval(mps)
   equs=1/sqrt(2.)*
&  sqrt((mps(1)-mps(2))**2+(mps(2)-mps(3))**2+(mps(3)-mps(1))**2)
   sigma(1)=sigma(1)-sig0; sigma(2)=sigma(2)-sig0
   sigma(4)=sigma(4)-sig0
   case (6)
      sig0=(sigma(1)+sigma(2)+sigma(3))/3
      sx=sigma(1); sy=sigma(2); sz=sigma(3)
      txy=sigma(4); tyz=sigma(5); tzx=sigma(6)
      j2=((sx-sy)**2+(sy-sz)**2+(sz-sx)**2)/6.+
&  txy**2+tyz**2+tzx**2
      j3=(sx-sig0)*(sy-sig0)*(sz-sig0)+2*txy*tyz*tzx-
&  (sx-sig0)*tyz**2-(sz-sig0)*txy**2-(sy-sig0)*tzx**2
      loang=asin((-sqrt(27.)*j3)/(2*sqrt(j2**3)))/3
      mps(1)=2*sqrt(j2)/sqrt(3.)*sin(loang+2*pi/3)+sig0
      mps(2)=2*sqrt(j2)/sqrt(3.)*sin(loang)+sig0
      mps(3)=2*sqrt(j2)/sqrt(3.)*sin(loang-2*pi/3)+sig0
      mpris=maxval(mps)
      equs=1/sqrt(2.)*
&  sqrt((mps(1)-mps(2))**2+(mps(2)-mps(3))**2+(mps(3)-mps(1))**2)
```

```
        sigma(1)=sigma(1)-sig0; sigma(2)=sigma(2)-sig0
        sigma(3)=sigma(3)-sig0
        case default
           print*, "wrong size for nst in rdmpes"
           end select
        return
        end subroutine rdmpes

subroutine KRH (f,x,sigma,equs,mpris,matpro)
 !> \brief
 ! KRH is used to return the creep strain and its damage value,
 ! this subroutine include the uni-axial and multi-axial format
 ! of KRH constitutive equation.
 !!
 implicit none
 doubleprecision, intent(inout) :: x(:),f(:)!< creep rates and its absolute values
 doubleprecision, intent(in) :: sigma(:), matpro(:)!< material property
 doubleprecision, intent(in) :: equs, mpris
 doubleprecision :: A, B, C, h, Hstar, Kc, v
 integer :: N, i, nst
 A=matpro(1); B=matpro(2); C=matpro(3); h=matpro(4)
 Hstar=matpro(5); Kc=matpro(6); v=matpro(7)
 if (mpris>0) then!< recognition of the value of N
    N=1
    else if (mpris<=0) then
       N=0
       end if
 nst=ubound(sigma,1)
 select case (nst)
       !> \brief
       ! case 1 is uni-axial form of constitutive equation
```

```
      ! case 4 is multi-axial form for 2D problem
      ! case 6 is multi-axial form for 3D problem
      !!
      case (1)
      f(1)=A*sinh((B*sigma(1)*(1-x(2)))/((1-x(3))*(1-x(4))))
      f(2)=h*f(1)/sigma(1)*(1.-(x(2)/Hstar))
      f(3)=Kc/3.*(1-x(3))**4
      f(4)=C*f(1)
  case (4)
     do i=1, 4
      f(i)=(3./2.)*(sigma(i)/equs)*A*
&  sinh((B*equs*(1-x(6)))/((1-x(7))*(1-x(8))))
      end do
      f(5)=sqrt((2./3.)*(f(1)**2+f(2)**2+2*f(3)**2+f(4)**2))
      f(6)=h*f(5)/equs*(1.-(x(6)/Hstar))
      f(7)=Kc/3.*(1.-x(7))**4
      f(8)=C*N*f(5)*(mpris/equs)**v
      case (6)
           do i=1, 6
             f(i)=(3./2.)*(sigma(i)/equs)*A*
&  sinh((B*equs*(1-x(8)))/((1-x(9))*(1-x(10))))
              end do
              f(7)=sqrt((2./3.)*(f(1)**2+f(2)**2+
&  f(3)**2+2*f(4)**2+2.*f(5)**2+2*f(6)**2))
              f(8)=h*f(7)/equs*(1.-(x(8)/Hstar))
              f(9)=Kc/3.*(1-x(9))**4
              f(10)=C*N*f(7)*(mpris/equs)**v
              case default
                print*, "wrong size for nst in KRH"
                end select
                return
```

```
                end subroutine KRH

subroutine KR (f,y,time,sigma,equs,mpris,matpro)
 !> \brief
 ! KR is used to return the creep strain and its damage value,
 ! this subroutine include the uni-axial and multi-axial format
 ! of KR constitutive equation.
 !!
 implicit none
 doubleprecision, intent(inout) :: f(:), y(:)!< creep rates and its absolute values
 doubleprecision, intent(in) :: sigma(:), matpro(:)!< material property
 doubleprecision, intent(in) :: mpris, equs, time
 doubleprecision :: A, n, m, B, phi, X, aerfa, rups
 integer :: i, nst
 A=matpro(1); n=matpro(2); m=matpro(3); B=matpro(4)
 phi=matpro(5); X=matpro(6); aerfa=matpro(7)
 rups=aerfa*mpris+(1.-aerfa)*equs!< return the value of rupture stress
 nst=ubound(sigma,1)
 select case (nst)
 !> \brief
 ! case 1 is uni-axial form of constitutive equation
 ! case 4 is multi-axial form for 2D problem
 ! case 6 is multi-axial form for 3D problem
 !!
 case (1)
      f(1)=A*((sigma(1)/(1-y(2)))**n)*(time**m)
      f(2)=B*(sigma(1)**X)/((1-y(2))**phi)*(time**m)
 case (4)
   do i=1,4
      f(i)=(3./2.)*(sigma(i)/equs)*A*
&       ((equs/(1-y(5)))**n)*(time**m)
```

```
        end do
        f(5)=B*(rups**X)/((1-y(5))**phi)*(time**m)
        case (6)
          do i=1,6
            f(i)=(3./2.)*(sigma(i)/equs)*A*
&             ((equs/(1-y(7)))**n)*(time**m)/2
            end do
            f(7)=B*(rups**X)/((1-y(7))**phi)*(time**m)
            case default
              print*, "wrong size of nst in KR"
              end select
              return
              end subroutine KR

 subroutine EULER_KRH (x,incx,dt,sigma,equs,mpris,matpro)
  !> \brief
  ! The EULER_KRH is used to integrate KRH constitutive equation
  ! with euler's method.
  !!
  implicit none
  doubleprecision, intent(inout) :: x(:), incx(:)
  doubleprecision, intent(in) :: sigma(:), matpro(:)
  doubleprecision, intent(in) :: equs, mpris, dt
  doubleprecision, allocatable, dimension(:) :: k
  integer :: nst, n
  nst=ubound(sigma,1)
  if (nst==1) then
    n=4
    elseif (nst==4) then
      n=8
      elseif (nst==6) then
```

```
        n=10
        end if
        allocate(k(n))
        call KRH(k,x,sigma,equs,mpris,matpro)
        incx=k*dt
        x=x+k*dt
        return
        end subroutine EULER_KRH

subroutine EULER_KR (x,incx,dt,time,sigma,equs,mpris,matpro)
 !> \brief
 ! The EULER_KR is used to integrate KR constitutive equation
 ! with euler's method.
 !!
 implicit none
 doubleprecision, intent(inout) :: x(:), incx(:)
 doubleprecision, intent(in) :: sigma(:), matpro(:)
 doubleprecision, intent(in) :: equs, mpris, dt, time
 doubleprecision, allocatable, dimension(:) :: k
 integer :: nst, n
 nst=ubound(sigma,1)
 if (nst==1) then
   n=4
   elseif (nst==4) then
     n=8
     elseif (nst==6) then
        n=10
        end if
        allocate(k(n))
        call KR(k,x,time,sigma,equs,mpris,matpro)
        incx=k*dt
```

```
            x=x+k*dt
            return
            end subroutine EULER_KR

 subroutine RK4_KRH (x,incx,dt,sigma,equs,mpris,matpro)
  !> \brief
  ! The EULER_KRH is used to integrate KRH constitutive equation
  ! with classical 4th order Runge-Kutta method.
  !!
  implicit none
  doubleprecision, intent(inout) :: x(:), incx(:)
  doubleprecision, intent(in) :: sigma(:), matpro(:)
  doubleprecision, intent(in) :: equs, mpris, dt
  doubleprecision, allocatable, dimension(:) :: k1, k2, k3, k4,
&  x1, x2, x3, x4
  integer :: nst, n
  nst=ubound(sigma,1)
  if (nst==1) then
     n=4
     elseif (nst==4) then
        n=8
        elseif (nst==6) then
           n=10
           end if
           allocate(k1(n),k2(n),k3(n),k4(n))
           x1=x
           call KRH(k1,x,sigma,equs,mpris,matpro)
           x2=x+dt/2*k1
           call KRH(k2,x2,sigma,equs,mpris,matpro)
           x3=x+dt/2*k2
           call KRH(k3,x3,sigma,equs,mpris,matpro)
```

```
        x4=x+dt*k3
        call KRH(k4,x4,sigma,equs,mpris,matpro)
        incx=(k1+2*k2+2*k3+k4)*dt/6
        x=x+(k1+2*k2+2*k3+k4)*dt/6
        return
        end subroutine RK4_KRH

 subroutine RK4_KR (x,incx,dt,time,sigma,equs,mpris,matpro)
  !> \brief
  ! The EULER_KR is used to integrate KR constitutive equation
  ! with classical 4th order Runge-Kutta method.
  !!
  implicit none
  doubleprecision, intent(inout) :: x(:), incx(:)
  doubleprecision, intent(in) :: sigma(:), matpro(:)
  doubleprecision, intent(in) :: equs, mpris, dt, time
  doubleprecision, allocatable, dimension(:) :: k1, k2, k3, k4,
&  x1, x2, x3, x4
  integer :: nst, n
  nst=ubound(sigma,1)
  if (nst==1) then
     n=2
     elseif (nst==4) then
        n=5
        elseif (nst==6) then
           n=7
           end if
           allocate(k1(n),k2(n),k3(n),k4(n))
           x1=x
           call KR(k1,x1,time,sigma,equs,mpris,matpro)
           x2=x+dt/2*k1
```

```
        call KR(k2,x2,time,sigma,equs,mpris,matpro)
        x3=x+dt/2*k2
        call KR(k3,x3,time,sigma,equs,mpris,matpro)
        x4=x+dt*k3
        call KR(k4,x4,time,sigma,equs,mpris,matpro)
        incx=(k1+2*k2+2*k3+k4)*dt/6
        x=x+(k1+2*k2+2*k3+k4)*dt/6
        return
        end subroutine RK4_KR

subroutine stress_deviator_2D (x,y)
implicit none
doubleprecision, dimension(3) :: x
doubleprecision, dimension(4) :: y
doubleprecision :: z
z=(x(1)+x(2))/3.
y(1)=x(1)-z
y(2)=x(3)
y(3)=x(2)-z
y(4)=-z
return
end subroutine stress_deviator_2D

subroutine equivalent_stress_2D (x,y)
implicit none
doubleprecision, dimension(4) :: x
doubleprecision :: y
y=sqrt(3.*(x(1)**2+2.*x(2)**2+x(3)**2+x(4)**2)/2.)
return
end subroutine equivalent_stress_2D
```

```
  subroutine max_principal_stress_2D (x,y)
  implicit none
  doubleprecision, dimension(3) :: x
  doubleprecision, dimension(2) :: z
  doubleprecision :: y
  z(1) = (x(1)+x(2))/2.+sqrt(((x(1)-x(2))/2)**2+x(3)**2)
  z(2) = (x(1)+x(2))/2.-sqrt(((x(1)-x(2))/2)**2+x(3)**2)
     if (z(1)-z(2)>0) then
     y=z(1)
     else
     y=z(2)
     end if
     return
  end subroutine max_principal_stress_2D


 subroutine KRHX (f,x,t,stress,material,nost,nocmp,noce)
  !>--------------------------introduction-------------------------
  !< KRH returns the creep strain and damage rate according to
  !< Kachanov-Rabotnov-Hayhurst-Xu constitutive equation. noce == 9,
  !< 2D problem; noce == 11, 3D problem.
  !!
    implicit none
    integer, intent(in) :: nost, nocmp, noce
    doubleprecision, intent(in) :: stress(nost), material(nocmp),
&                                  x(noce)
    doubleprecision, intent(in) :: t
    doubleprecision, intent(out) :: f(noce)
    doubleprecision :: sx, sy, sz, txy, tyz, tzx, ps1, ps2, ps3,
&                      es, Ss, sm, S1
    doubleprecision :: A, B, C, h, Hstar, Kc, v, a1, b1, p, q
    integer :: N
```

```
      select case (nost)
      case (8)
        sx = stress(1); sy = stress(2); txy = stress(3)
        sz = stress(4); ps1 = stress(5); ps2 = stress(6)
        ps3 = stress(7); es = stress(8); tyz = 0.0; tzx = 0.0
        case (10)
          sx = stress(1); sy = stress(2); sz = stress(3)
          txy = stress(4); tyz = stress(5); tzx = stress(6)
          ps1 = stress(7); ps2 = stress(8); ps3 = stress(9)
          es = stress(10)
          case default
            print*, "wrong size for nost in KRHX"
            end select
      A = material(1); B = material(2); C = material(3)
      h = material(4); Hstar = material(5); Kc = material(6)
      v = material(7); a1 = material(8); b1 = material(9)
      p = material(10); q = material(11)
      sm = (ps1+ps2+ps3)/3; Ss = sqrt(ps1**2+ps2**2+ps3**2)
      S1 = ps1-sm
      if (ps1>0) then
        N=1
        else if (ps1<=0) then
          N=0
          end if
      select case (noce)
      case (9)
        f(1) = (3./2.)*(sx/es)*A*
     &       sinh((B*es*(1-x(6)))/((1-x(7))*(1-x(8))))
        f(2) = (3./2.)*(sy/es)*A*
     &       sinh((B*es*(1-x(6)))/((1-x(7))*(1-x(8))))
        f(3) = (3./2.)*(txy/es)*A*
```

```
&          sinh((B*es*(1-x(6)))/((1-x(7))*(1-x(8))))
      f(4) = (3./2.)*(sz/es)*A*
&          sinh((B*es*(1-x(6)))/((1-x(7))*(1-x(8))))
      f(5) = sqrt((2./3.)*(f(1)**2+f(2)**2+2*f(3)**2+f(4)**2))
      f(6) = h*f(5)/es*(1.-(x(6)/Hstar))
      f(7) = Kc/3.*(1.-x(7))**4
      f(8) = C*N*f(5)*
&          (exp(p*(1-(ps1/es))+q*(0.5-1.5*(sm/es))))**(-1)
      f(9) = f(8)*((2/3)*(es/S1))**a1*exp(b1*(3*sm/Ss-1))
      case (11)
        f(1) = (3./2.)*(sx/es)*A*
&            sinh((B*es*(1-x(8)))/((1-x(9))*(1-x(10))))
        f(2) = (3./2.)*(sy/es)*A*
&            sinh((B*es*(1-x(8)))/((1-x(9))*(1-x(10))))
        f(3) = (3./2.)*(sz/es)*A*
&            sinh((B*es*(1-x(8)))/((1-x(9))*(1-x(10))))
        f(4) = (3./2.)*(txy/es)*A*
&            sinh((B*es*(1-x(8)))/((1-x(9))*(1-x(10))))
        f(5) = (3./2.)*(tyz/es)*A*
&            sinh((B*es*(1-x(8)))/((1-x(9))*(1-x(10))))
        f(6) = (3./2.)*(tzx/es)*A*
&            sinh((B*es*(1-x(8)))/((1-x(9))*(1-x(10))))
        f(7) = sqrt((2./3.)*(f(1)**2+f(2)**2+f(3)**2+2*f(4)**2
&            +2.*f(5)**2+2*f(6)**2))
        f(8)=h*f(7)/es*(1.-(x(8)/Hstar))
        f(9)=Kc/3.*(1-x(9))**4
        f(10) = C*N*f(5)*
&            (exp(p*(1-(ps1/es))+q*(0.5-1.5*(sm/es))))**(-1)
        f(11) = f(10)*((2/3)*(es/S1))**a1*exp(b1*(3*sm/Ss-1))
        case default
          print*, "wrong size for noce in KRHX"
```

```
        end select
        return
        end subroutine KRHX

subroutine NOR_KR (f,x,t,stress,material,nost,nocmp,noce)
!>---------------------------introduction--------------------------
!< NORKR returns the creep strain and damage rate according to
!< normalized Kachanov-Rabotnov constitutive equation. noce == 9,
!< 2D problem; noce == 11, 3D problem.
!!
 implicit none
 !>----------------------variables list-------------------------
 !< Scalar integers:
 !< i          simple counter
 !< noc        number of components
 !<
 !< Scalar reals:
 !< mps        maximum principal stress
 !< es         equivalent stress
 !< rs         rupture stress
 !< ip         internal pressure
 !< cfs        creep failure strain
 !< alpha      creep material property
 !< beta       creep material property
 !< m          creep material property
 !< n          creep material property
 !< phi        creep material property
 !< chi        creep material property
 !< e          young¡¯s modulus
 !< a          stress state coefficient
 !< t          current total time
```

```
  !<
  !< Dynamic real arrays:
  !< f          creep strain rate components and damage rate
  !< y          creep strain value components and damage value
  !< cmp        creep material properties
  !< edstc      elastic deviatoric stress tensor components
  !!
  integer, intent(in) :: nost, nocmp, noce
  doubleprecision, intent(in) :: stress(nost), material(nocmp),
&                        x(noce)
  doubleprecision, intent(in) :: t
  doubleprecision, intent(out) ::  f(noce)
  doubleprecision :: sx, sy, sz, txy, tyz, tzx, mps, es, cfs
  doubleprecision :: A, n, m, B, phi, chi, alpha, rs, e, ip
  select case (nost)
  case (8)
    sx = stress(1); sy = stress(2); txy = stress(3)
    sz = stress(4); mps = stress(5); es = stress(8)
    tyz = 0.0; tzx = 0.0; ip = stress(9)
    case (10)
      sx = stress(1); sy = stress(2); sz = stress(3)
      txy = stress(4); tyz = stress(5); tzx = stress(6)
      mps = stress(7); es = stress(10); ip = stress(11)
      case default
        print*, "wrong size for nost in NOR_KR"
          end select
  A = material(1); n = material(2); m = material(3)
  B = material(4); phi = material(5); chi = material(6)
  alpha = material(7); e =  material(8)
  rs = alpha*mps+(1.-alpha)*es
  cfs = (A*e/B)*(ip**(n-chi-1.))
```

```
select case (noce)
case (5)
   f(1)=(3./2.)*(sx/es)*((es/(1-x(5)))**n)
   f(2)=(3./2.)*(sy/es)*((es/(1-x(5)))**n)
   f(3)=(3./2.)*(txy/es)*((es/(1-x(5)))**n)
   f(4)=(3./2.)*(sz/es)*((es/(1-x(5)))**n)
   f(5) = (rs**chi)/(cfs*(1.+phi)*((1.-x(5))**phi))
   case (7)
      f(1)=(3./2.)*(sx/es)*((es/(1-x(7)))**n)
      f(2)=(3./2.)*(sy/es)*((es/(1-x(7)))**n)
      f(3)=(3./2.)*(sz/es)*((es/(1-x(7)))**n)
      f(4)=(3./2.)*(txy/es)*((es/(1-x(7)))**n)
      f(5)=(3./2.)*(tyz/es)*((es/(1-x(7)))**n)
      f(6)=(3./2.)*(tzx/es)*((es/(1-x(7)))**n)
      f(7) = (rs**chi)/(cfs*(1.+phi)*((1.-x(7))**phi))
      case default
         print*, "wrong size of noce in NOR_KR"
         end select
         return
         end subroutine NOR_KR

subroutine RKM (func,y,t,dt,stress,material,nost,nocmp,noce,rcv)
!>-------------------------introduction-------------------------
!< RKM returns the solution of creep strains and creep damage
!< variables according to Runge-Kutta-Merson method.
!!
 implicit none
 !>-----------------------variables list----------------------
 !< external function:
 !< func            creep constitutive equation
 !<
```

```
!< Scalar integers:
!< rcv          re-do control value
!< nost         number of stress terms
!< nocmp        number of creep material properties
!< noce         number of constitutive equations
!<
!< Scalar reals:
!< dt           time increment
!< t            current total time
!<
!< Dynamic real arrays:
!< k            creep strain and damage rate
!< y            creep strain and damage value
!< mfs           mean function slope
!< material       creep material properties
!< stress         stress terms
!< loer           local error
!< aoi            acceptance of integration
!!
external :: func
integer, intent(in) :: nost, nocmp, noce
doubleprecision, intent(inout) :: y(noce)
doubleprecision, intent(in) :: stress(nost), material(nocmp)
doubleprecision, intent(in) :: t, dt
integer, intent(out) :: rcv
doubleprecision :: maoi
doubleprecision :: k1(noce), k2(noce), k3(noce), k4(noce),
&                  k5(noce), mfs(noce), loer(noce), aoi(noce)
call func(k1,y,t,stress,material,nost,nocmp,noce)
call func(k2,y+dt/3*k1,t+dt/3,stress,material,nost,nocmp,noce)
call func(k3,y+dt/6*(k1+k2),t+dt/3,stress,material,nost,nocmp,
```

```
&           noce)
    call func(k4,y+dt/8*(k1+3*k3),t+dt/2,stress,material,nost,
&           nocmp,noce)
    call func(k5,y+dt/2*(k1-3*k3+4*k4),t+dt,stress,material,nost,
&           nocmp,noce)
    mfs = (k1+4*k4+k5)/6
    y = y+mfs*dt
    loer = (2*k1-9*k3+8*k4-k5)/30
    aoi = loer/mfs
    maoi = maxval(aoi)
    if (maoi<0.001) then
     rcv = 0
     else
        rcv = 1
        end if
        return
        end subroutine RKM

  subroutine RKF (func,y,t,dt,stress,material,nost,nocmp,noce,rcv)
  !>------------------------introduction-------------------------
  !< RKF returns the solution of creep strains and creep damage
  !< variables according to Runge-Kutta-Fehlberg method.
  !!
    implicit none
    !>-----------------------variables list----------------------
    !< external function:
    !< func            creep constitutive equation
    !<
    !< Scalar integers:
    !< rcv           re-do control value
    !< nost          number of stress terms
```

```
!< nocmp          number of creep material properties
!< noce           number of constitutive equations
!<
!< Scalar reals:
!< dt             time increment
!< t              current total time
!<
!< Dynamic real arrays:
!< k              creep strain and damage rate
!< y              creep strain and damage value
!< mfs            mean function slope
!< material       creep material properties
!< stress         stress terms
!< aoi            acceptance of integration
!!
external :: func
integer, intent(in) :: nost, nocmp, noce
doubleprecision, intent(inout) :: y(noce)
doubleprecision, intent(in) :: stress(nost), material(nocmp)
doubleprecision, intent(in) :: t, dt
integer, intent(out) :: rcv
doubleprecision :: maoi
doubleprecision :: k1(noce), k2(noce), k3(noce), k4(noce),
&                  k5(noce), k6(noce), mfs4(noce), mfs5(noce),
&                  aoi(noce), ybar(noce)
call func(k1,y,t,stress,material,nost,nocmp,noce)
call func(k2,y+dt/4*k1,t+dt/4,stress,material,nost,nocmp,noce)
call func(k3,y+dt/32*(3*k1+9*k2),t+3*dt/8,stress,material,
&          nost,nocmp,noce)
call func(k4,y+dt/2179*(1932*k1-7200*k2+7296*k3),t+12*dt/13,
&          stress,material,nost,nocmp,noce)
```

```
    call func(k5,y+dt/4104*(8341*k1-32832*k2+29440*k3-845*k4),t+dt
&           ,stress,material,nost,nocmp,noce)
    call func(k6,y+dt*(-(8/27)*k1+2*k2-(3544/2565)*k3+(1859/4104)
&           *k4-(11/40)*k5),t+dt/2,stress,material,nost,nocmp,
&           noce)
    mfs4 = (25/216)*k1+(1408/2565)*k3+(2197/4104)*k4-(1/5)*k5
    mfs5 = (16/135)*k1+(6656/12825)*k3+(28561/56430)*k4-(9/50)*k5
&       -(2/55)*k6
    ybar = y+mfs4*dt
    y = y+mfs5*dt
    aoi = abs(y-ybar)
    maoi = maxval(aoi)
    if (maoi<1d-5) then
     rcv = 0
     else
        rcv = 1
        end if
        return
        end subroutine RKF


        end module lib_add
```

Appendix D: User Guidance for Software HITSI

User guidance and tutorial

(c) DEZHENG LIU, 2014, School of Computing & Engineering,

University of Huddersfield, Huddersfield, HD1 3DH.

Tel: (+44) (0) 789-582 0829

E-mail: U1372661@hud.ac.uk

Contents

This user guidance contains a brief description of the in-house FE software **HITSI**, step-by-step instructions for a demonstration (using the data files included) and a tutorial with 4 examples illustrating data preparation for the features of this in-house FE software.

The technical manual describes the features of **HITSI** in full, with descriptions of element types, problem types, and the data sections. A summary of the format of these data sections and general information about data entry is given in the help facility of the program.

1) Introduction
2) Analysis types
3) Tutorial on data preparation
4) Tutorial on output results

A set of lecture notes on the FEM for creep deformation and problems, with exercises and examples for the use of **HITSI**, is also available.

1. Introduction

HITSI is a FEM based CDM approach in-house software for the analysis of creep deformation and damage. It is easy to use, and capable of solving different problems of stress, strain, creep damage and deformation analysis, and the prediction of the failure time of components (**plane stress**, **plane strain**, **axisymmetric** and **three-dimensional**). **HITSI** was designed for two-dimensional creep damage analysis, but it is suitable for practical use on a range of problems, offering multiple element types include 2D and 3D (not yet fully implemented) and additional features such as different creep constitutive damage equations and integration methods.

HITSI comprises the *main executable file* ***.EXE, a *data file* ***.DAT (a table of standard materials), a *blocks project file* ***.CBP, a *depend file* ***.DEPEND, a *results file* ***.RES, a *layout file* ***.LAYOUT, and a number of *object files* called ***.OBJ.

The sequence of operation: 1) a *data file* should be created or modified; 2) click the *main executable file*. Once the finite element mesh, materials constants, boundary conditions and loads information are acceptable, the finite element analysis is done, and finally the results are stored in a *results file*. If errors occurred, return to the data file and check the format of the mesh, materials constants, boundary conditions and loads information and run the main program again. Here, the *blocks project file*, *depend file* and *layout file* need to be deleted before executing the main program.

The editing rule for the different analysis types (**plane stress**, **plane strain**, **axisymmetric** and **three-dimensional**) is incorporated, for creating or modifying data files, and this gives the appropriate format and some data checking. The user can use this rule to create or modify files. The most efficient way to do this is to create a simple file with the pre-processing and post-processing FE software FEMGV, then modify it via a pre-processing data transfer program and use the editing rules to check the format in *data file*. FEMGV contains facilities to interpolate sequences of nodes and elements, and to replicate blocks, enabling the input of quite large meshes with few lines of data. Hence, less time and fewer errors result if the above steps are used to create *data file*.

The calculation module implements eight types of element: 1) three, six and fifteen node triangles element type; 2) four, eight and nine node quadrilaterals element type; 3) eight and twenty node 3D brick element type. The different analysis types are carried

out by the different constitutive matrix; and the different subroutines are used in element stiffness assembly, integration and the solution of equilibrium equation.

The graphics are carried out by a post-processing data transfer program and the pre-processing and post-processing FE software FEMGV. The results are output according to the output rules in main program. Then, the format of the results file can be transferred with the post-processing data transfer program and can be read by FEMGV.

This user guidance primarily consists of four main parts: 1) Introduction; 2) Analysis types; 3) Tutorial on data preparation; 4) Tutorial on output results. The four parts should be joined together in the use of in-house FE software **HITSI**.

2. Analysis types

HITSI is developed for solving the creep damage problem and it covers **plane stress, plane strain, axisymmetric** and **three-dimensional** analysis types. The FE codes for the spatial discretisation by finite elements, element stiffness integration, element stiffness assembly, solution of equilibrium equation, creep damage constitutive equation, the numerical time integration method, the stress and creep damage field variables updating are used in the analysis of the creep damage problem under the different constitutive matrix. Thus, the analysis type should be defined before the use of **HITSI**.

The analysis types of this software are summarised in following:

1. **Plane stress**

In plane stress problem, the condition prevails in a flat plate in the *x* and *y* plane, loaded only in its own plane and without z-direction restraint, so that $\sigma_x = \tau_{xy} = \tau_{zx} = 0$. Thus, the constitutive matrix is:

$$\mathbf{E} = \frac{E}{1-\nu^2}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & (1-\nu)/2 \end{bmatrix}$$

Where the *E* is the Young's modulus and ν is the Poisson's ratio.

Users need to define the nature of the case before building the FE model. If the nature of problem satisfies the plane stress constitutive matrix; user should select the **plane stress** index in **HITSI**.

2. **Plane strain**

In plane strain problem, the condition prevailing is defined as a deformation state in which total potential energy is zero everywhere and *u* and *v* are functions of *x* and *y* but not of *z*. Thus, a typical slice of an underground tunnel that lies along the *z* axis might deform in essentially plane strain conditions. The constitutive matrix is:

$$\mathbf{E} = \frac{E}{(1+\nu)(1-2\nu)}\begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & (1-2\nu)/2 \end{bmatrix}$$

Where the *E* is the Young's modulus and ν is the Poisson's ratio. If needed, σ_z can be obtained from the relation $\varepsilon_z = 0 = (\sigma_z - \nu\sigma_y - \nu\sigma_x) / E$ after σ_x and σ_y are known.

If the nature of problem satisfies the plane strain constitutive matrix; user should select the **plane strain** index in **HITSI**.

3. Axisymmetric

In axisymmetric problem, the condition considers a constant value of displacement in the circumferential direction. The constitutive matrix is:

$$\mathbf{E} = \frac{E}{(1+\nu)(1-2\nu)}\begin{bmatrix} 1-\nu & \nu & \nu & 0 \\ \nu & 1-\nu & \nu & 0 \\ \nu & \nu & 1-\nu & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix}$$

Where the *E* is the Young's modulus and ν is the Poisson's ratio.

If the nature of problem satisfies the axisymmetric constitutive matrix; user should select the **axisymmetric** index in HITSI.

4. Three-dimensional

In three-dimensional problem, the condition considers the stereo features, *u*, *v* and *w* being the displacements in the *x*, *y* and *z* directions respectively. The constitutive matrix is:

$$\mathbf{E} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)}\begin{bmatrix} 1, & \nu/(1-\nu), & \nu/(1-\nu), & 0, & 0, & 0 \\ & 1, & \nu/(1-\nu), & 0, & 0, & 0 \\ & & 1, & 0, & 0, & 0 \\ SYM.... & & & (1-2\nu)/2(1-\nu), & 0, & 0 \\ & & & & (1-2\nu)/2(1-\nu), & 0 \\ & & & & & (1-2\nu)/2(1-\nu) \end{bmatrix}$$

Where the *E* is the Young's modulus and ν is the Poisson's ratio.

If the nature of problem satisfies the three-dimensional constitutive matrix; user should select the **3D** index in **HITSI**.

3. Tutorial on data preparation

Full instructions are given in this section for the demonstration of the analysis types, using data files provided.

1) **Plane stress**
2) **Plane strain**
3) **Axisymmetric**
4) **Three-dimensional**

The format, mesh information, material constants, boundary conditions and loads information for the above problem types in *data file* are demonstrated. Then, four examples are presented in order to understand the editing rules and to show the features of HITSI.

If the selected main program has been compiled together with the example *data file*, make sure the current directory contains the *main executable file* and *data file* to associate the information of different files; and the program will be able to save the *output files*.

3.1 Data preparation for plane stress problem

3.1.1 Data preparation

The sequence of data preparation is as follows:

1) In main program: *read (10,*) element, nels, nn, nip, nodof, nod, nst, ndim, oppo*. The corresponding command in *data file*: element type, number of elements, number of nodes in the mesh, number of integration points, number of degrees of freedom per node, number of nodes per element, number of stress terms, number of dimensions and number of parameters in creep damage constitutive equation.
2) In main program: *read (10,*) nprops, np_types*. The corresponding command in *data file*: number of material property and number of different property type.
3) In main program: *read (10,*) prop.* The corresponding command in *data file*: element properties.
4) In main program: *read (10,*) etype*. The corresponding command in *data file*: element property type vector.
5) In main program: *read (10,*) k, g_coord (:, i).* The corresponding command in *data file*: global nodal coordinates.
6) In main program: *read (10,*) k, g_num (:, i).* The corresponding command in *data file*: global element node number vector.
7) In main program: *read (10,*) nr; if (nr>0) read (10,*) (k, nf (:, k), i=1, nr)*. The corresponding command in *data file*: number of restrained nodes.
8) In main program: *read (10,*) loaded_nodes; allocate (no (loaded_nodes), storkv (loaded_nodes, ndim))*. The corresponding command in *data file*: number of loaded nodes.
9) In main program: *read (10,*) (no (i), storkv (i, :), i=1, loaded_nodes).* The corresponding command in *data file*: nodal loads information.

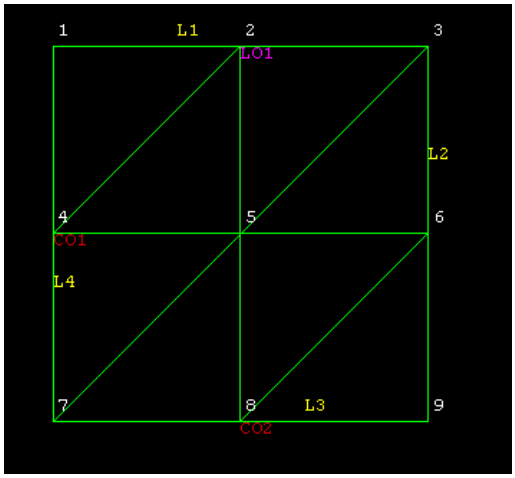
The creep damage constitutive equation subroutines for the Kachanov-Rabotnov-Hayhurst, the Kachanov-Rabotnov and the Kachanov-Rabotnov-Hayhurst-Xu models are available in this solver. The corresponding commands in main program: the *subroutine KRH*, the *subroutine KR* and *subroutine KRHX*. User can select the different constitutive equation models with a simple *call* statement in main program.

- Kachanov-Rabotnov-Hayhurst equation: *call subroutine KRH* in main program; the order of parameters (*oppo* = 7) in creep damage constitutive equation in *data file* can be represented by: *A, B C, h, H*,* K_C and *v*.
- Kachanov-Rabotnov equation: *call subroutine KR* in main program; the order of parameters (*oppo* = 7) in creep damage constitutive equation in *data file* can be represented by: *K, n, m, M, Φ,* χ and *a*.
- Kachanov-Rabotnov-Hayhurst-Xu equation: *call subroutine KRHX* in main program; the order of parameters (*oppo* = 11) in creep damage constitutive equation in *data file* can be represented by: *A, B, C, h, H*,* K_C*, v, a, b, p* and *q*.

3.1.2 Example

A uni-axial tension FE model which is adopted from Smith's version linear elastic FE program: geotech / software / prog_fe / P50.F90 in (Smith and Griffiths, 2005) with uniform 3-node triangular elements numbered in the *x*-direction is used for the demonstration of data preparation.

The FE model is shown as follows:



In *data file*, the input information corresponds to the sequence of data preparation in Section 3.1.1 and is presented as follows:

1)

'triangle' 8 9 3 2 3 4 2 7

2)

9

3)

2

4)

100000.0 .3 2.93965d-12 4.3680 -0.2031 1.15878d-9 4.9667 2.8554 0.4298

100000.0 .3 2.93965d-12 4.3680 -0.2031 1.15878d-9 4.9667 2.8554 0.4298

5)

1 1 1 1

2 2 2 2

6)

Node 1 0.0000E+00 0.0000E+00

Node 2 0.5000E+00 0.0000E+00

Node 3 0.1000E+01 0.0000E+00

Node 4 0.0000E+00 -0.5000E+00

Node 5 0.5000E+00 -0.5000E+00

Node 6 0.1000E+01 -0.5000E+00

Node 7 0.0000E+00 -0.1000E+01

Node 8 0.5000E+00 -0.1000E+01

Node 9 0.1000E+01 -0.1000E+01

7)

Element 1 1 2 4

Element 2 5 4 2

Element 3 2 3 5

Element 4 6 5 3

Element 5 4 5 7

Element 6 8 7 5

Element 7 5 6 8

Element 8 9 8 6

8)

5

1 0 1 4 0 1 7 0 0 8 1 0 9 1 0

9)

3

1 0.0 10.0

2 0.0 20.0

```
3  0.0  10.0
```

This model contains 8 elements and 9 nodes. The length of a side is set to 1 metre. The Young's modulus E and Poisson's ratio υ are set to 1,000 MPa and 0.3 respectively. A uniformly linear distributed load of 10 KN/m is applied on the top line of this uni-axial tension model.

3.2 Data preparation for plane strain problem

3.2.1 Data preparation

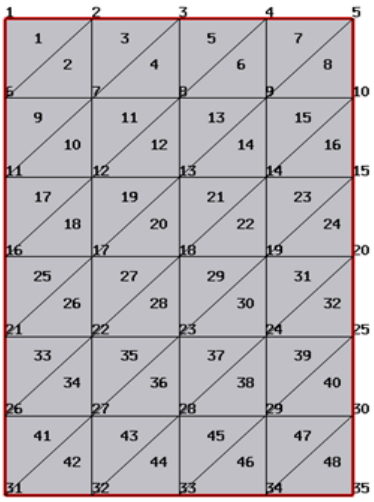
The sequence of data preparation is as follows:

1) In main program: *read (10,*) element, nels, nn, nip, nodof, nod, nst, ndim, oppo*. The corresponding command in *data file*: element type, number of elements, number of nodes in the mesh, number of integration points, number of degrees of freedom per node, number of nodes per element, number of stress terms, number of dimensions and number of parameters in creep damage constitutive equation.
2) In main program: *read (10,*) nprops, np_types*. The corresponding command in *data file*: number of material property and number of different property type.
3) In main program: *read (10,*) prop.* The corresponding command in *data file*: element properties.
4) In main program: *read (10,*) etype*. The corresponding command in *data file*: element property type vector.
5) In main program: *read (10,*) k, g_coord (:, i).* The corresponding command in *data file*: global nodal coordinates.
6) In main program: *read (10,*) k, g_num (:, i).* The corresponding command in *data file*: global element node number vector.
7) In main program: *read (10,*) nr; if (nr>0) read (10,*) (k, nf (:, k), i=1, nr)*. The corresponding command in *data file*: number of restrained nodes.
8) In main program: *read (10,*) loaded_nodes; allocate (no (loaded_nodes), storkv (loaded_nodes, ndim))*. The corresponding command in *data file*: number of loaded nodes.
9) In main program: *read (10,*) (no (i), storkv (i, :), i=1, loaded_nodes).* The corresponding command in *data file*: nodal loads information.

3.2.2 Example

A uni-axial tension FE model with uniform 3-node triangular elements numbered in the *x*-direction is used for the demonstration of data preparation.

The FE model is shown as follows:



In *data file*, the input information corresponds to the sequence of data preparation in Section 3.2.1 and is presented as follows:

1)

'triangle' 48 35 3 2 3 4 2

2)

2

3)

3

4)

1000000.0 .3

1000000.0 .3

1000000.0 .3

5)

1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1

2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 2

3 3 3 3 3 3 3 3

3 3 3 3 3 3 3 3

6)

Node 1 0.0000E+00 0.1000E+01

Node 2 0.1000E+01 0.0000E+00

Node 3 0.2000E+01 0.0000E+00

Node 4 0.3000E+01 0.0000E+00

Node 5 0.4000E+01 0.0000E+00

Node 6 0.0000E+00 -0.1000E+01

Node 7 0.1000E+01 -0.1000E+01

Node 8 0.2000E+01 -0.1000E+01

Node 9 0.3000E+01 -0.1000E+01

Node 10 0.4000E+01 -0.1000E+01

Node 11 0.0000E+00 -0.2000E+01

Node 12 0.1000E+01 -0.2000E+01

Node 13 0.2000E+01 -0.2000E+01

Node 14 0.3000E+01 -0.2000E+01

Node 15 0.4000E+01 -0.2000E+01

Node 16 0.0000E+00 -0.3000E+01

Node 17 0.1000E+01 -0.3000E+01

Node 18 0.2000E+01 -0.3000E+01

Node 19 0.3000E+01 -0.3000E+01

Node 20 0.4000E+01 -0.3000E+01

Node 21 0.0000E+00 -0.4000E+01

Node 22 0.1000E+01 -0.4000E+01

Node 23 0.2000E+01 -0.4000E+01

Node 24 0.3000E+01 -0.4000E+01

Node 25 0.4000E+01 -0.4000E+01

Node 26 0.0000E+00 -0.5000E+01

Node 27 0.1000E+01 -0.5000E+01

Node 28 0.2000E+01 -0.5000E+01

Node 29 0.3000E+01 -0.5000E+01

Node 30 0.4000E+01 -0.5000E+01

Node 31 0.0000E+00 -0.6000E+01

Node 32 0.1000E+01 -0.6000E+01

Node 33 0.2000E+01 -0.6000E+01

Node 34 0.3000E+01 -0.6000E+01

Node 35 0.4000E+01 -0.6000E+01

7)

Element 1 1 2 6

Element 2 7 6 2

Element 3 2 3 7

Element 4 8 7 3

Element 5 3 4 8

Element 6 9 8 4

Element 7 4 5 9

Element 8 10 9 5

Element 9 6 7 11

Element 10 12 11 7

Element 11 7 8 12

Element 12 13 12 8

Element 13 8 9 13

Element 14 14 13 9

Element 15 9 10 14

Element 16 15 14 10

Element 17 11 12 16

Element 18 17 16 12

Element 19 12 13 17

Element 20 18 17 13

Element 21 13 14 18

Element 22 19 18 14

Element 23 14 15 19

Element 24 20 19 15

Element 25 16 17 21

Element 26 22 21 17

Element 27 17 18 22

Element 28 23 22 18

Element 29 18 19 23

Element 30 24 23 19

Element 31 19 20 24

Element 32 25 24 20

Element 33 21 22 26

Element 34 27 26 22

Element 35 22 23 27

Element 36 28 27 23

Element 37 23 24 28

Element 38 29 28 24

Element 39 24 25 29

Element 40 30 29 25

Element 41 26 27 31

Element 42 32 31 27

Element 43 27 28 32

Element 44 33 32 28

Element 45 28 29 33

Element 46 34 33 29

Element 47 29 30 34

Element 48 35 34 30

8)

11

1 0 1 6 0 1 11 0 1 16 0 1 21 0 1 26 0 1 31 0 0 32 1 0 33 1 0 34 1 0 35 1 0

9)

5

1 0.0 5.0

2 0.0 10.0

3 0.0 10.0

4 0.0 10.0

5 0.0 5.0

This model contains 48 elements and 35 nodes. The width of this model is set to 5 metres. The Young's modulus E and Poisson's ratio υ are set to 1,000 MPa and 0.3 respectively. A uniformly linear distributed load of 10 KN/m is applied on the top line of this uni-axial tension model.

3.3 Data preparation for axisymmetric problem

3.3.1 Data preparation

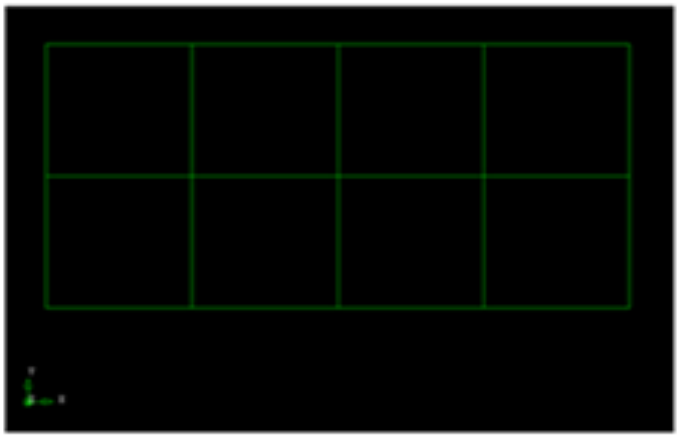
The sequence of data preparation is as follows:

1) In main program: *read (10,*) element, nels, nn, nip, nodof, nod, nst, ndim, oppo*. The corresponding command in *data file*: element type, number of elements, number of nodes in the mesh, number of integration points, number of degrees of freedom per node, number of nodes per element, number of stress terms, number of dimensions and number of parameters in creep damage constitutive equation.
2) In main program: *read (10,*) nprops, np_types*. The corresponding command in *data file*: number of material property and number of different property type.
3) In main program: *read (10,*) prop.* The corresponding command in *data file*: element properties.
4) In main program: *read (10,*) etype*. The corresponding command in *data file*: element property type vector.
5) In main program: *read (10,*) k, g_coord (:, i).* The corresponding command in *data file*: global nodal coordinates.
6) In main program: *read (10,*) k, g_num (:, i).* The corresponding command in *data file*: global element node number vector.
7) In main program: *read (10,*) nr; if (nr>0) read (10,*) (k, nf (:, k), i=1, nr)*. The corresponding command in *data file*: number of restrained nodes.
8) In main program: *read (10,*) loaded_nodes; allocate (no (loaded_nodes), storkv (loaded_nodes, ndim))*. The corresponding command in *data file*: number of loaded nodes.
9) In main program: *read (10,*) (no (i), storkv (i, :), i=1, loaded_nodes).* The corresponding command in *data file*: nodal loads information.

3.3.2 Example

A uni-axial tension FE model with uniform 4-node quadrilateral elements numbered in the *x*-direction is used for the demonstration of data preparation.

The FE model is shown as follows:



In *data file*, the input information corresponds to the sequence of data preparation in Section 3.3.1 and is presented as follows:

1)

'quadrilateral' 8 15 4 2 4 4 2 7

2)

2

3)

2

4)

1000000.0 .3

1000000.0 .3

5)

1 1 1 1 1 1 1 1

2 2 2 2 2 2 2 2

6)

Node 1 0.0000E+00 0.0000E+00

Node 2 0.0000E+00 -0.4000E+01

Node 3 0.0000E+00 -0.8000E+01

Node 4 0.4000E+01 0.0000E+00

Node 5 0.4000E+01 -0.4000E+01

Node 6 0.4000E+01 -0.8000E+01

Node 7 0.8000E+01 0.0000E+00

Node 8 0.8000E+01 -0.4000E+01

Node 9 0.8000E+01 -0.8000E+01

Node 10 0.1200E+02 0.0000E+00

Node 11 0.1200E+02 -0.4000E+01

Node 12 0.1200E+02 -0.8000E+01

Node 13 0.1600E+02 0.0000E+00

Node 14 0.1600E+02 -0.4000E+01

Node 15 0.1600E+02 -0.8000E+01

7)

Element 1 2 1 4 5

Element 2 3 2 5 6

Element 3 5 4 7 8

Element 4 6 5 8 9

Element 5 8 7 10 11

Element 6 9 8 11 12

Element 7 11 10 13 14

Element 8 12 11 14 15

8)

5

3 0 0 6 1 0 9 1 0 12 1 0 15 1 0

9)

5

1 .0 1. 4 .0 6. 7 .0 12. 10 .0 18. 13 .0 11

This model contains 8 elements and 15 nodes. The Young's modulus E and Poisson's ratio υ are set to 1,000 MPa and 0.3 respectively. The length of each element is 4m. A uniformly distributed tensile force 0.375 KN/m^2 is applied on the top line of this uni-axial tension model.

3.4 Data preparation for three-dimensional problem

3.4.1 Data preparation

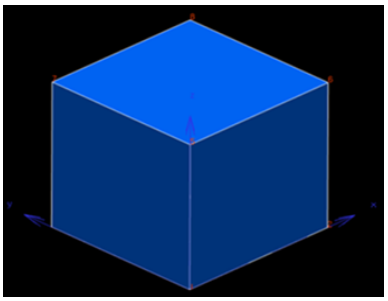
The sequence of data preparation is as follows:

1) In main program: *read (10,*) element, nels, nn, nip, nodof, nod, nst, ndim*. The corresponding command in *data file*: element type, number of elements, number of nodes in the mesh, number of integration points, number of degrees of freedom per node, number of nodes per element, number of stress terms, number of dimensions and number of parameters in creep damage constitutive equation.
2) In main program: *read (10,*) nprops, np_types*. The corresponding command in *data file*: number of material property and number of different property type.
3) In main program: *read (10,*) prop.* The corresponding command in *data file*: element properties.
4) In main program: *read (10,*) etype*. The corresponding command in *data file*: element property type vector.
5) In main program: *read (10,*) k, g_coord(:, i).* The corresponding command in *data file*: global nodal coordinates.
6) In main program: *read (10,*) k, g_num(:, i).* The corresponding command in *data file*: global element node number vector.
7) In main program: *read (10,*) nr; if (nr>0) read (10,*) (k, nf (:, k), i=1, nr)*. The corresponding command in *data file*: number of restrained nodes.
8) In main program: *read (10,*) loaded_nodes; allocate (no (loaded_nodes), storkv (loaded_nodes, ndim))*. The corresponding command in *data file*: number of loaded nodes.
9) In main program: *read (10,*) (no (i), storkv (i, :), i=1, loaded_nodes).* The corresponding command in *data file*: nodal loads information.

3.4.2 Example

A uni-axial tension FE model which is adopted from Smith's version linear elastic FE program: geotech / software / prog_fe / P58.F90 in (Smith and Griffiths, 2005) with a brick three-dimensional element is used for the demonstration of data preparation.

The FE model is shown as follows:



In *data file*, the input information corresponds to the sequence of data preparation in Section 3.4.1 and is presented as follows:

1)

'hexahedron' 1 20 27 3 20 6 3

2)

6

3)

1

4)

1000000. .3 30. 0. 0. -20.

5)

1

6)

Node 1 0.0000E+00 0.0000E+00 0.0000E+00

Node 2 0.5000E+00 0.0000E+00 0.0000E+00

Node 3 0.1000E+01 0.0000E+00 0.0000E+00

Node 4 0.0000E+00 0.0000E+00 -0.5000E+00

Node 5 0.1000E+01 0.0000E+00 -0.5000E+00

Node 6 0.0000E+00 0.0000E+00 -0.1000E+01

Node 7 0.5000E+00 0.0000E+00 -0.1000E+01

Node 8 0.1000E+01 0.0000E+00 -0.1000E+01

Node 9 0.0000E+00 0.5000E+00 0.0000E+00

Node 10 0.1000E+01 0.5000E+00 0.0000E+00

Node 11 0.0000E+00 0.5000E+00 -0.1000E+01

Node 12 0.1000E+01 0.5000E+00 -0.1000E+01

Node 13 0.0000E+00 0.1000E+01 0.0000E+00

Node 14 0.5000E+00 0.1000E+01 0.0000E+00

Node 15 0.1000E+01 0.1000E+01 0.0000E+00

Node 16 0.0000E+00 0.1000E+01 -0.5000E+00

Node 17 0.1000E+01 0.1000E+01 -0.5000E+00

Node 18 0.0000E+00 0.1000E+01 -0.1000E+01

Node 19 0.5000E+00 0.1000E+01 -0.1000E+01

Node 20 0.1000E+01 0.1000E+01 -0.1000E+01

7)

Element 1 6 4 1 2 3 5 8 7 11 9 10 12 18 16 13 14 15 17 20 1916

8)

1 0 0 1 2 1 0 1 3 1 0 1 4 0 0 1 5 1 0 1

6 0 0 0 7 1 0 0 8 1 0 0 9 0 1 1 11 0 1 0

12 1 1 0 13 0 1 1 16 0 1 1 18 0 1 0 19 1 1 0 20 1 1 0

9)

8

1 3 -10.05

2 3 -10.05

3 3 -10.05

9 3 -10.05

10 3 -10.05

13 3 -10.05

14 3 -10.05

15 3 -10.05

This model contains 1 element and 20 nodes. The Young's modulus E and Poisson's ratio υ are set to 1,000 MPa and 0.3 respectively. The length of a side is set to 1 metre and a uniformly distributed load of 5 KN was applied on the top surface of this uni-axial tension model.

4. Tutorial on output results

The running results are stored in dynamic arrays and they can be output in the end of program by a *write* statement. The results can be displayed in *results file*. The sequence of the calculated results is showing as follows:

1) In main program: *write (11, 99998) key1; write (11,*) ndim, nn, nod, nels, element, nst, nip, t0, e, v, key1*. The corresponding command in *results file*: number of dimensions, number of nodes, number of nodes per element, number of elements, element type, number of stress terms, number of integrating points, total time, Young's modulus, Poisson's ratio and output index for node number.
2) In main program: *write (11, 99998) key2; do k=1, nn; write (11,*) k, g_coord (:, k); end do*. The corresponding command in results file: global nodal coordinates.
3) In main program: *write (11, 99998) key3; do k = 1, nels; write (11,*) k, g_num (:, k), key1; end do*. The corresponding command in results file: global element node number matrix.
4) In main program: *write (11, 99998) key4; do k=1, nn; write (11,*) k, loads (nf (:, k)); end do*. The corresponding command in results file: nodal loads and displacement.
5) In main program: *write (11, 99998) key5; do k=1, nn; write (11,*) k, loads (nf (:, k)); end do*. The corresponding command in results file: nodal freedom matrix.
6) In main program: *write (11, 99998) key6; do i=1, nels; write (11,*) I; do j=1, nip; write (11,*) j, tgc (:, j, i); end do; end do*. The corresponding command in results file: The coordinates of integrating points.
7) In main program: *write (11, 99998) key7; do i=1, nels; write (11,*) I; do j=1, nip; write (11,*) j, tsigma(:, j, i); end do; end do*. The corresponding command in results file: The stress terms.
8) In main program: *write (11, 99998) key8; do i=1, nels; write (11,*) I; do j=1, nip; write (11,*) j, teps (:, j, i); end do; end do*. The corresponding command in results file: The strain terms.
9) In main program *write (11, 99998) key9; do i=1, nels; write (11,*) I; do j=1, nip; write (11,*) j, evpt (:, j, i); end do; end do*. The corresponding command in results file: The creep strain terms.
10) In main program: *write (11, 99998) key10; do i=1, nels; write (11,*) I; do j=1, nip; write (11,*) j, tabv (5, j, i); end do; end do*. The corresponding command in results file: The creep damage

11) In main program: *write (11, 99998) key11*. The corresponding command in results file: The output index

The graphics are carried out by a post-processing data transfer program and the pre-processing and post-processing FE software FEMGV. The results are output according to the output rules above. Then, the format of the results file can be transferred with the post-processing data transfer program and read by FEMGV.