Software Requirements Specification, Module Guide, and Module Interface Specification for Partial Differential Equation Solver Module in Two and Three Dimensional Dynamic Model of Soil-Water-Structure Interaction

Prepared by:
Brandon Karchewski (karcheba@mcmaster.ca)
Ph.D. Candidate
Department of Civil Engineering

Prepared for:
Dr. Spencer Smith
CES 741 - Development of Scientific Computing Software
Department of Computational Engineering and Science

McMaster University
Hamilton, Ontario, Canada

Ver. DynSWS-PDE-1.0
April 14, 2012

Table of Contents

Table of Symbols

Table 0.1: Document list prefixes

Symbol	Description
A	Assumption
AC	Anticipated Change
IIM	Internal Instanced Model
UC	Unlikely Change
UG	User Group

Acronyms and Abbreviations

2-D/3-D: Two-dimensional/three-dimensional; refers to the dimension of the coordinate system used to solve the problem.

DynSWS: Dynamic model of Soil-Water-Structure interaction; the software product described herein.

MG: Module guide; the document that presents the high-level design of the software product (see reference [1] as well).

MIS: Module interface specification; the name of the document that specifies the syntax for interaction between modules (see reference [2] as well).

PDE: Partial differential equation. Models of physical phenomena are typically stated mathematically as systems of this type of equation that must be integrated in order to obtain the solution. Initial and/or boundary conditions are also required for a given problem.

SRS: Software requirements specification; the document that specifies the requirements for a software product (see reference [3] as well).

Quick Reference Tables

Table 0.2: Index of User Groups

Item	Description	Page
UG1	Developers	2
UG2	Maintainers	2
UG3	Reviewers	2

Table 0.3: Index of Assumptions

Item	Description	Page
A1	Relatively small discretization error	4
A2	Field variables may be interpolated between nodal values	4
A3	Linear interpolation of displacement, velocity, and acceleration	5
A4	Constant stress and strain within a body element	5
A5	Linear interpolation of surface tractions	5
A6	Constant material density within a body element	5
A7	Constant elastic properties within a body element	5
A8	Constant applied acceleration within a body element	5
A9	Damping is a linear combination of mass and stiffness	5
A10	Four term Taylor series approximation of change in displacement	5
A11	Three term Taylor series approximation of change in velocity	5
A12	Linear change in acceleration over a time-step	5
A13	Input data has a constant time-step	5

Table 0.4: Index of Internal Instanced Models

Table 0.5: Index of Anticipated Changes

Item	Description	Page
AC1	Homogeneity of materials	19
AC2	Isotropy of materials	19
AC3	Small strain assumption	19
AC4	Linear elastic model for structure subdomain	19
AC5	Cartesian coordinate system	19
AC6	Plane strain assumption for 2-D model	19
AC7	Algorithm of PDE solver	20
AC8	Algorithm of linear system of equations solver	20
AC9	Spatial discretization technique	20
AC10	Constitutive matrix for structural subdomain	20
AC11	Body element interpolation functions	20
AC12	Traction element interpolation functions	20
AC13	Linear differential operator	20
AC14	Stress-strain (kinematic) matrix	20
AC15	Algorithm for computing element area	20
AC16	Algorithm for computing kinematic matrix	20
AC17	Form of mass matrix	20
AC18	Integration algorithm for mass matrix	20
AC19	Integration algorithm for stiffness matrix	21
AC20	Integration algorithm for traction vector	21
AC21	Integration algorithm for body force vector	21
AC22	Integration algorithm for initial stress field	21
AC23	Integration algorithm for initial strain field	21
AC24	Algorithm for computing damping matrix	21
AC25	Temporal discretization technique	21
AC26	Values of temporal discretization constants	21
AC27	Algorithm for computing initial acceleration	21
AC28	Integration algorithm for load vector	21
AC29	Algorithm for updating acceleration field	21
AC30	Algorithm for updating displacement and velocity fields	21
AC31	Algorithm for updating stress and strain fields	21

Table 0.6: Index of Unlikely Changes

Item	Description	Page
UC1	Time-varying load input	22
UC2	Types of subdomain	22
UC3	Functional goals	22
UC4	Isothermality of domain	22
UC5	No internal material sources or sinks	22
UC6	Neglect of relativistic effects	22
UC7	Continuum mechanics framework	22
UC8	Use of Rayleigh damping	22
UC9	Direct time-domain solution technique	22

Table 0.7: Index of External Leaf Modules

Item	Description	Page
1.2.2	Integer Operations	
1.2.3	Floating Point Operations	
1.2.4	Memory Access	
2.4.1	Log Message Control	
3.1	System Constants	
3.2.1	Domain Data	
3.2.2	Boundary Data	
3.2.3	Material Property Data	

Table 0.8: Index of Internal Leaf Modules

Item	Description	Page
3.3	PDE Solver Control	27
3.4	PDE Solver Constants	27
3.5.1	Dense Matrix ADT	27
3.5.2	Banded Symmetric Matrix ADT	27
3.5.3	Vector ADT	28
3.6.1	Body Element Integration	28
3.6.2	Traction Element Integration	28
3.7.1	Body Element Interpolation	28
3.7.2	Traction Element Interpolation	28
3.8.1	Linear Elastic Model	29
3.8.2	Constitutive Matrix	29
3.8.3	Kinematic Matrix	29
3.9.1	Linear Solver	29

1 Introduction

This section introduces the SRS, MG, and MIS for the PDE Solver module within the DynSWS software product. Readers not yet acquainted with DynSWS should see reference [3], which is the software requirements specification for the software product described herein. Section 1.1 describes the purpose of this document. Section 1.2 explains the transition from the high-level design of the MG for DynSWS [1] to the internal design presented herein; in particular, the reasoning for creating this additional document is summarized. Section 1.3 relates the scope of DynSWS presented in this document to the scope of the software requirements specification (if there is any difference). Section 1.4 identifies the intended audience for this document. Section 1.5 outlines the organization of the document.

1.1 Purpose

This document presents the internal design and specification of the PDE Solver module within DynSWS (see reference [1] for how this module fits into the external design). The internal modularization is based on the principle of information hiding (see references [1] and [4] for more information). Note that while the syntax of internal modules is specified within this document, the top-level syntax and behaviour of the PDE Solver module must satisfy the MIS for DynSWS (see reference [2]).

1.2 Bridge Between High-Level Design and Internal Design and Specification

The MG for DynSWS specifies the high-level design of the software product. Referring to **AC28** in the MG, the algorithm for solving systems of PDEs is an anticipated change for DynSWS. Therefore, one of the key modules in the design of DynSWS is the PDE Solver. The selection, design, and implementation of a numerical algorithm for solving PDEs is non-trivial in itself and may involve a number of anticipated internal changes that ultimately do not influence the rest of the design of DynSWS. As such, the author has seen fit to prepare this document so that the PDE Solver module benefits from internal modularization without cluttering the MG for DynSWS with anticipated changes and modules that pertain only to the PDE Solver.

1.3 Scope

The scope of the design of DynSWS presented in this document is reduced from that of the SRS and the MG. While the SRS and MG document the complete end goal for DynSWS, this document will focus on the numerical algorithm for the structural subdomain in 2-D only. As a quick reminder, the structural subdomain is an impermeable, single phase, solid domain governed by small strain theory and a linear elastic plane strain constitutive model (see reference [3] for further details).

1.4 Intended Audience

The three main groups that this document is intended for use by are:

UG1. Developers. Users in this group are involved in the actual implementation of the requirements of DynSWS. This will certainly include the author, but may include others in the future if the software product proves useful and the functionality continues to be extended over time. This group can use this document as a reference to the internal design of the PDE Solver , which presents the selected numerical algorithm, its modularization, and its syntax. If users from this group modify the internal design of the PDE Solver, they must always satisfy the behaviour specified in the MG and the top-level syntax specified in the MIS; developers should also update this document to reflect any modifications to the internal design and syntax of the PDE Solver module.

UG2. Maintainers. Users in this group maintain the software product over time. This may include activities such as performing tests, fixing bugs, and reorganizing the module hierarchy to reflect design modifications. Again, this will initially be just the author, but in the future may include others. If the design is modified by users in this group, changes should be documented herein.

UG3. Reviewers. Users in this group have the task of ensuring that DynSWS meets all requirements and that the results produced by the software product are correct (insofar as correctness can be determined). This includes the author, but also the author's supervisory committee as they will be responsible for verifying the correctness and accuracy of the model contained in DynSWS. This document will be useful for this group in understanding the numerical algorithm used to solve PDEs, which is a key component in the overall design of DynSWS.

It should be noted that this document is not necessarily intended for end users of the software product. This document presents the internal design and syntax of the PDE Solver without going into detail on the requirements or the high-level design of DynSWS. Readers interested in the requirements specification and the high-level design of DynSWS should see references [3] and [1], respectively.

1.5 Organization of the Document

This document is essentially a combined SRS, MG, and MIS at a level internal to the PDE Solver module. For more details on the organization of these documents, see references [3], [1], and [2], respectively. Section 2 picks up from the instanced models presented in the SRS and develops the numerical algorithm selected to solve them, presenting additional background theory and instanced models as necessary. Section 3 outlines the internal module guide and Section 4 provides the internal module interface specification for the PDE Solver.

2 Numerical Algorithm

This section presents the models required to implement the numerical algorithm for the PDE Solver in DynSWS. Section 2.1 provides terminology definitions relevant to the numerical algorithm. Section 2.2 lists any assumptions made for the numerical models (beyond the assumptions presented in the SRS). Section 2.3 presents the instanced models required to solve the PDEs over both space and time.

2.1 Terminology

Constitutive: Refers to the manner in which a material behaves when undergoing stress and strain.

Determinant: A scalar property of a matrix. Useful in computing areas and transforming coordinate systems.

Discretization/Discretized/Discrete: Discretization in the context of DynSWS refers to the process of dividing a continuous domain (of space or time) into finite segments. Once the process is complete, the domain is referred to as discretized. A discrete point is a node within the discretized domain.

Element: A finite segment of a discretized domain. Includes body elements and surface elements.

Gaussian Quadrature: A numerical integration technique that approximates the value of an integral by a sum of the values of the function at a set of "integration points" multiplied by appropriate weighting constants. Named after Carl Friedrich Gauss.

Global/Local Coordinates: Refer to the sets of coordinates useful for describing the overall domain and the domain of an individual element, respectively.

Interpolation: The process of determining the value of a function between two points where the value is known through an assumed set of functions.

Jacobian of Transformation: Name given to both the matrix and its determinant that are used in transforming a function from one coordinate system to another. Named after mathematician Carl Gustav Jacob Jacobi.

Kinematic: Within the context of continuum mechanics and the FEM, refers to the relationship between an appropriate field variable and strain (or strain rate). For solid mechanics, this is the strain-displacement relationship.

Linear Differential Operator: An operator that is a linear function of the differentiation operator. The function takes a (possibly vector valued) function and returns another function.

Node/Nodal: A point (in space or time) within a discretized domain. Node is a noun, nodal is an adjective.

Newmark Time Integration: A parametric family of numerical integration techniques for solving differential equations. Named after civil engineer Nathan M. Newmark.

Rayleigh Damping: A technique for estimating the viscous damping properties of a system by taking a linear combination of the mass and stiffness. Named after physicist John William Strutt, 3rd Baron Rayleigh (sometimes shortened to Lord Rayleigh).

Spatial Domain: The domain of space (geometry).

Taylor Series: An infinite series that represents the value of a function at a point based on the value of the function (and its derivatives) at another point, the distance between the points, and constant coefficients. Named after mathematician Brook Taylor.

Temporal Domain: The domain of time.

Time-step: A finite period of time defining the discretized temporal domain.

Traction: A distributed surface stress.

Virtual Work: A virtual quantity obtained by moving a set of forces through an arbitrary displacement (or other appropriate term). Systems at equilibrium have the property that the sum of virtual work contributions from all terms will vanish.

Weak Form of Equilibrium: A modified expression of equilibrium that involves integrating the contributions of virtual work over the solution domain. It is referred to as "weak" because it no longer guarantees equilibrium at every point within the domain. The consequence is that equilibrium will be satisfied for each element in the discretized domain, but not at every point within each element.

2.2 Assumptions

This following is a list of assumptions made in developing the numerical algorithm for the PDE Solver:

A1. The errors in the solution caused by discretization of the spatial and temporal domains are small compared to errors due to the approximation of nature by mathematical models and the estimation of material properties. Discretization is necessary to solve the PDEs that describe the model contained in DynSWS for all but the simplest of geometric and loading configurations. As such, one does not often have a choice of whether or not to use a discretized approximation. However, it is prudent to make this assumption explicit so that there is no ambiguity with regard to the expected capability of the numerical model.

A2. The values of field variables may be obtained by direct interpolation between a set of discretized nodal values. This is a key assumption in the development of the spatial discretization technique.

A3. The distribution of the displacement, velocity, and acceleration fields within a body element is linear. This assumption is associated with the constant stress triangular body element.

A4. The stress and strain within a body element is constant. This assumption is associated with the constant stress triangular body element.

A5. The distribution of applied tractions a traction element is linear. This assumption is associated with the linear interpolation line traction element.

A6. The material density within a body element is constant. This simplifies the implementation of the integration scheme for the mass matrix and body force component of the load vector. Given the assumptions of material homogeneity and isotropy, the domain may always be discretized in a manner that accommodates this assumption.

A7. The elastic material properties within a body element are constant. This assumption is made for similar reasons to those described in **A6**.

A8. The applied acceleration field is constant within a body element for a given point in time. This assumption is made for convenience. Considering that the source of applied accelerations will be seismic ground movement, the applied acceleration field will, in fact, be constant over the entire domain for a given point in time.

A9. The structural subdomain exhibits viscous damping that may be determined as a linear combination of mass and stiffness terms. Since the true nature of damping in structural systems is difficult to determine, this is a common technique used to estimate the influence of damping, which is sometimes referred to as Rayleigh damping [5].

A10. The change in the discretized displacement over a time-step may be approximated by the first four terms in its Taylor series expansion. This is a key assumption in the Newmark family of time-stepping techniques.

A11. The change in the discretized velocity field over a time-step may be approximated by the first three terms in its Taylor series expansion. Similar reasons as those described in **A10**.

A12. The discretized acceleration field varies linearly over a time step. Similar reasons as those described in **A10**.

A13. A constant time-step is used for the input data. This includes boundary conditions and body accelerations.

2.3 Internal Instanced Models

This section presents the instanced mathematical models required to implement the PDE Solver module. Section 2.3.1 presents the preliminary formulation necessary to prepare the equilibrium expression for the structural subdomain for discretization. Section 2.3.2 presents the selected spatial discretization element and derives the equilibrium expression for the discretized domain. Section 2.3.3 gives the algorithm for stepping through the temporal domain. Section 2.3.4 summarizes the input and output variables for the PDE Solver module. Note that both the spatial and temporal discretization techniques solve the problem by dividing the solution domain into segments and solving for the numerical values at discrete points (with assumed interpolation between these points). Consequently, the solution obtained using these algorithms include error due to the discretization process. Referring to **A1**, it is assumed that the discretization errors are small compared to other sources of error such as approximation of natural processes with mathematical models and estimation of material properties.

2.3.1 Weak (or Integrated) Form of Equilibrium

The numerical technique selected for solving the PDEs involved in DynSWS over the spatial domain for a given point in time is the finite element method (FEM). In order to derive the necessary terms for the FEM formulation, the equilibrium expression for the structural subdomain must be converted to the weak (or integrated) form.

As stated in Section 1.3, the scope has been limited to the solution of the instanced models for the structural subdomain in 2-D at this time. Recalling **IM1** of the SRS, the set of dynamic equilibrium equations for the structural subdomain are given by:

IIM1. Dynamic Equilibrium of Structural Subdomain

$$\rho_s \ddot{\boldsymbol{u}}_{\boldsymbol{s}} = \nabla \cdot \boldsymbol{\sigma}_{\boldsymbol{s}} + \rho_s \boldsymbol{f} \tag{2.1}$$

where ρ is the material density, $\boldsymbol{u}$ is the displacement vector field, $\boldsymbol{\sigma}$ is the stress tensor field, $\boldsymbol{f}$ is the body acceleration vector field, $\nabla\cdot$ is the divergence operator, subscript s refers to the structural subdomain, bold-face indicates a vector or tensor, and superimposed dots indicate derivatives with respect to time. Since the model scope is currently limited to 2-D, the displacement vector field is defined, as in **IM10** of the SRS, as:

IIM2. Displacement Vector

$$\boldsymbol{u}_{\boldsymbol{s}} = \begin{Bmatrix} u_x \\ u_y \end{Bmatrix} \tag{2.2}$$

where subscripts x, y refer to the global coordinate axis directions. Similarly, the body acceleration field vector is defined as:

IIM3. Body Acceleration Vector

$$\boldsymbol{f} = \begin{Bmatrix} f_x \\ f_y \end{Bmatrix} \tag{2.3}$$

Owing to isotropy of materials (**A7** of the SRS), the stress field tensor is symmetric and may be defined in compact vector form, as in **IM8** of the SRS, as:

IIM4. Stress Tensor in Compact Vector Form

$$\boldsymbol{\sigma_s} = \begin{Bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{Bmatrix} \tag{2.4}$$

where subscripts xx, yy indicate normal stress in the global coordinate directions and subscript xy indicates shear stress. Note that, although **IIM4** is written as a vector, it is understood that any coordinate transformations of $\boldsymbol{\sigma_s}$ must still follow the rules for a second order tensor. Applying the principle of virtual work, at equilibrium the work done by the inertial forces (left hand side of **IIM1**) through a virtual displacement, $\delta\boldsymbol{u_s}$, must equal the work done by the internal forces and the applied forces (right hand side of **IIM1**) through the same virtual displacement. Mathematically, this is expressed as:

IIM5. Virtual Work Expression for Structural Subdomain

$$\delta\boldsymbol{u_s}^T\rho_s\ddot{\boldsymbol{u}}_{\boldsymbol{s}} = \delta\boldsymbol{u_s}^T(\nabla\cdot\boldsymbol{\sigma_s}) + \delta\boldsymbol{u_s}^T\rho_s\boldsymbol{f} \tag{2.5}$$

where superscript T indicates the transpose operation. Since the principle of virtual work must be true at all points in the spatial domain, one may integrate **IIM5** over the domain to yield:

$$\int_V \delta\boldsymbol{u_s}^T\rho_s\ddot{\boldsymbol{u}}_{\boldsymbol{s}}dV = \int_V \delta\boldsymbol{u_s}^T(\nabla\cdot\boldsymbol{\sigma_s})dV + \int_V \delta\boldsymbol{u_s}^T\rho_s\boldsymbol{f}dV \tag{2.6}$$

where V indicates volume integration. Integrating equation (2.6) by parts using the divergence theorem yields:

$$\int_V \delta\boldsymbol{u_s}^T\rho_s\ddot{\boldsymbol{u}}_{\boldsymbol{s}}dV = -\int_V \nabla\delta\boldsymbol{u_s}^T\boldsymbol{\sigma_s}dV + \int_S \delta\boldsymbol{u_s}^T\boldsymbol{\sigma_s}\cdot\boldsymbol{n_s}dS + \int_V \delta\boldsymbol{u_s}^T\rho_s\boldsymbol{f}dV \tag{2.7}$$

where S indicates integration over a boundary surface and $\boldsymbol{n_s}$ represents the outward unit normal from the boundary of the structural subdomain at a point. Recognizing that the gradient of a virtual displacement field is a virtual strain field and defining $\boldsymbol{t_s} = \boldsymbol{\sigma_s}\cdot\boldsymbol{n_s}$ as the surface traction vector field for the structural subdomain gives:

$$\int_V \delta\boldsymbol{u_s}^T\rho_s\ddot{\boldsymbol{u}}_{\boldsymbol{s}}dV = -\int_V \delta\boldsymbol{\varepsilon_s}^T\boldsymbol{\sigma_s}dV + \int_S \delta\boldsymbol{u_s}^T\boldsymbol{t_s}dS + \int_V \delta\boldsymbol{u_s}^T\rho_s\boldsymbol{f}dV \tag{2.8}$$

The strain vector is defined, as in **IM8** of the SRS, as:

IIM6. Strain Tensor in Compact Vector Form

$$\boldsymbol{\varepsilon_s} = \begin{Bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \gamma_{xy} \end{Bmatrix} \tag{2.9}$$

where the subscripts have the same meaning as in **IIM4** and $\boldsymbol{\varepsilon_s}$ must still transform as a second order tensor. The traction vector, which is defined only over the surface of the structural subdomain, is given by:

IIM7. Traction Vector

$$\boldsymbol{t_s} = \begin{Bmatrix} t_x \\ t_y \end{Bmatrix} \tag{2.10}$$

where the subscripts have the same meaning as in **IIM2** and **IIM3**. In general, the domain may have a non-zero initial stress and strain field, so the stress-strain (constitutive) relationship is:

IIM8. Stress-Strain (Constitutive) Relationship

$$\boldsymbol{\sigma_s} = \boldsymbol{D_s}(\boldsymbol{\varepsilon_s} - \boldsymbol{\varepsilon_0}) + \boldsymbol{\sigma_0} \tag{2.11}$$

where $\boldsymbol{\sigma_0}$ and $\boldsymbol{\varepsilon_0}$ are the initial stress and strain fields, respectively. The constitutive matrix, $\boldsymbol{D_s}$, which defines the stress-strain relationship, as in **IM8** of the SRS, is:

IIM9. Constitutive Matrix for the Structural Subdomain

$$\boldsymbol{D_s} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \frac{1}{2}(1-2\nu) \end{bmatrix} \tag{2.12}$$

where E and ν are the elastic modulus and Poisson's ratio of the material, respectively. Combining equation (2.8) with **IIM8**, the weak form of equilibrium for the structural subdomain is given by:

IIM10. Weak Form of Equilibrium for the Structural Subdomain

$$\begin{aligned} &\int_V \delta\boldsymbol{u_s}^T \rho_s \ddot{\boldsymbol{u}}_{\boldsymbol{s}} dV + \int_V \delta\boldsymbol{\varepsilon_s}^T \boldsymbol{D_s}\boldsymbol{\varepsilon_s} dV \\ &\qquad = \int_S \delta\boldsymbol{u_s}^T \boldsymbol{t_s} dS + \int_V \delta\boldsymbol{u_s}^T \rho_s \boldsymbol{f} dV - \int_V \delta\boldsymbol{\varepsilon_s}^T \boldsymbol{\sigma_0} dV + \int_V \delta\boldsymbol{\varepsilon_s}^T \boldsymbol{D_s}\boldsymbol{\varepsilon_0} dV \end{aligned} \tag{2.13}$$

2.3.2 Spatial Discretization

This section begins with the weak form of equilibrium and very briefly develops the instanced models for the spatial domain using the FEM. This section certainly does not serve as an exhaustive treatise on the underlying theory of the FEM. Readers interested in learning more about the theory and application of the FEM should see references [6] and [7] for an introduction.

It is useful to rewrite the weak form of equilibrium as a summation of integrations over discrete elements. Two element types are adopted in the present formulation: a body element (denoted by subscript e) and a traction element (denoted by subscript t). The weak form of equilibrium presented in **IIM10** is therefore equivalent to:

IIM11. Weak Form of Equilibrium (Summation over Elements)

$$\begin{aligned}\sum_{e=1}^{N_{ele}}\left(\int_{V_e}\delta\boldsymbol{u_e}^T\rho_e\ddot{\boldsymbol{u}}_{\boldsymbol{e}}dV_e+\int_{V_e}\delta\boldsymbol{\varepsilon_e}^T\boldsymbol{D_e}\boldsymbol{\varepsilon_e}dV_e\right)&\\=\sum_{t=1}^{N_{trc}}\left(\int_{S_t}\delta\boldsymbol{u_t}^T\boldsymbol{t_t}dS_t\right)&\\+\sum_{e=1}^{N_{ele}}\left(\int_{V_e}\delta\boldsymbol{u_e}^T\rho_e\boldsymbol{f_e}dV_e\right.&\\\left.-\int_{V_e}\delta\boldsymbol{\varepsilon_e}^T\boldsymbol{\sigma_{0e}}dV_e+\int_{V_e}\delta\boldsymbol{\varepsilon_e}^T\boldsymbol{D_e}\boldsymbol{\varepsilon_{0e}}dV_e\right)&\end{aligned}\tag{2.14}$$

where D_e is the same as D_s and the parameters N_{ele} and N_{trc} are the number of body elements and traction elements, respectively. Note that the summations of these vector expressions must take into account the connectivity between elements as only values at corresponding nodes may be added together.

Within the context of the FEM, the displacement, velocity, and acceleration solution is obtained for discrete nodal values with the field between nodes being defined by interpolation functions. For a body element, these interpolation functions have the following functional form:

$$\boldsymbol{u_e}=\boldsymbol{N_e}\boldsymbol{a_e}\tag{2.15}$$
$$\dot{\boldsymbol{u}}_{\boldsymbol{e}}=\boldsymbol{N_e}\dot{\boldsymbol{a}}_{\boldsymbol{e}}\tag{2.16}$$
$$\ddot{\boldsymbol{u}}_{\boldsymbol{e}}=\boldsymbol{N_e}\ddot{\boldsymbol{a}}_{\boldsymbol{e}}\tag{2.17}$$
$$\delta\boldsymbol{u_e}=\boldsymbol{N_e}\delta\boldsymbol{a_e}\tag{2.18}$$

where $\boldsymbol{N_e}$ is a vector of interpolation functions for the body element, $\boldsymbol{a_e}$ is a vector of discretized displacement field values at the nodes of the body element, and other operations are as defined previously. Similarly, the interpolation functions for the traction element have the following functional form:

$$\boldsymbol{u_t}=\boldsymbol{N_t}\boldsymbol{a_t}\tag{2.19}$$
$$\delta\boldsymbol{u_t}=\boldsymbol{N_t}\delta\boldsymbol{a_t}\tag{2.20}$$

where $\boldsymbol{N_t}$ is a vector of interpolation functions for the traction element and $\boldsymbol{a_t}$ is a vector of discretized displacement field values at the nodes of the traction element. Note that the form of $\boldsymbol{N_e}$ and $\boldsymbol{N_t}$ will change depending on the type of element selected.

Recalling **IM10**, the linear differential operator that relates the strain field and the displacement field in the 2-D model is:

IIM12. Linear Differential Operator (Strain-Displacement)

$$\boldsymbol{L} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \tag{2.21}$$

Combining equation (2.15) with **IM10** from the SRS allows the strain field for a body element to be written in terms of the discretized element displacement field:

IIM13. Discretized Strain-Displacement (Kinematic) Relationship

$$\boldsymbol{\varepsilon_e} = \boldsymbol{L N_e a_e} = \boldsymbol{B_e a_e} \tag{2.22}$$

where $\boldsymbol{B_e} = \boldsymbol{L N_e}$ is sometimes referred to as the kinematic matrix.

The body element selected to facilitate the spatial discretization of the PDEs involved in DynSWS is the constant stress triangular element (see Figure 2.1).

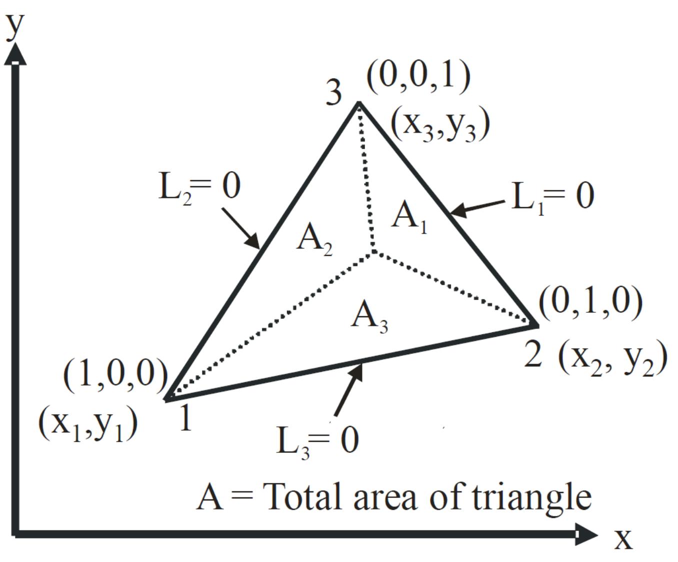

Figure 2.1: Constant stress triangular element

The element shown in Figure 2.1 has three nodes, which implies linear interpolation of the displacement, velocity, and acceleration fields as well as constant stress and strain in an element (as the name suggests). The discretized displacement field vector for the element is defined as:

IIM14. Element Displacement Field Vector

$$\boldsymbol{a_e} = \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{Bmatrix} \tag{2.23}$$

where u_i and v_i represent the displacements in the global x and y directions at each of the nodes, respectively. Interpolation within the element is most efficiently achieved using the area coordinates $\{L_1, L_2, L_3\}$. These coordinates are related to the nodal coordinates by [7]:

IIM15. Area Coordinates

$$\begin{bmatrix} L_1 \\ L_2 \\ L_3 \end{bmatrix} = \frac{1}{2A_e} \begin{Bmatrix} A_{23} & b_1 & c_1 \\ A_{31} & b_2 & c_2 \\ A_{12} & b_3 & c_3 \end{Bmatrix} \begin{bmatrix} 1 \\ x \\ y \end{bmatrix} \tag{2.24}$$

where A_e is the area of the element, given by [7]:

IIM16. Element Area

$$A_e = \frac{1}{2} \begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} \tag{2.25}$$

and A_{ij}, b_i, and c_i are defined as [7]:

IIM17. Element Parameters

$$A_{ij} = x_i y_j - x_j y_i \qquad \textit{for } \{i,j\} \in \{1,2,3\} \tag{2.26}$$

$$b_i = y_j - y_k \qquad \textit{for } \{i,j,k\} = \{\{1,2,3\},\{2,3,1\},\{3,1,2\}\} \tag{2.27}$$

$$c_i = x_k - x_j \qquad \textit{for } \{i,j,k\} = \{\{1,2,3\},\{2,3,1\},\{3,1,2\}\} \tag{2.28}$$

The interpolation functions for the element are defined as [7]:

IIM18. Element Interpolation Functions

$$\boldsymbol{N_e} = \begin{bmatrix} L_1 & 0 & L_2 & 0 & L_3 & 0 \\ 0 & L_1 & 0 & L_2 & 0 & L_3 \end{bmatrix} \tag{2.29}$$

Combining **IIM13** with **IIM18** gives the kinematic matrix for the element as:

IIM19. Element Kinematic Matrix

$$\boldsymbol{B_e} = \boldsymbol{L}\boldsymbol{N_e} = \frac{1}{2A_e} \begin{bmatrix} b_1 & 0 & b_2 & 0 & b_3 & 0 \\ 0 & c_1 & 0 & c_2 & 0 & c_3 \\ c_1 & b_1 & c_2 & b_2 & c_3 & b_3 \end{bmatrix} \tag{2.30}$$

The traction element selected to facilitate solution of the PDEs in DynSWS is a linear interpolation line element (see Figure 2.2).

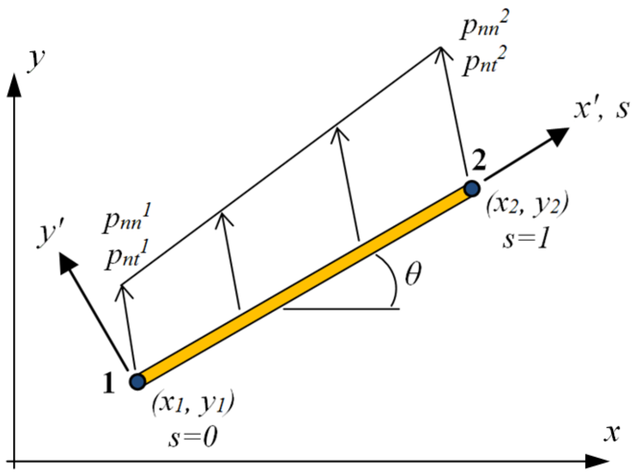

Figure 2.2: Linear interpolation line element

The element shown in Figure 2.2 has two nodes with a local coordinate s running from $s = 0$ at node 1 to $s = 1$ at node 2. The local tractions may be written as:

IIM20. Element Traction Interpolation

$$\boldsymbol{t'_t} = \boldsymbol{N_t}\boldsymbol{\bar{t}_t'} = \begin{bmatrix} (1-s) & 0 & s & 0 \\ 0 & (1-s) & 0 & s \end{bmatrix} \begin{Bmatrix} p^1_{nt} \\ p^1_{nn} \\ p^2_{nt} \\ p^2_{nn} \end{Bmatrix} \tag{2.31}$$

where p^i_{nt} and p^i_{nn} represent shear and normal tractions at node i, respectively. Referring to **IIM11**, it is convenient to rewrite the traction element integral as:

$$\int_{S_t} \delta(\boldsymbol{u_t})^T \boldsymbol{t_t} dS_t = \int_{S_t} \delta(\boldsymbol{u'_t})^T \boldsymbol{t'_t} dS_t \tag{2.32}$$

where $\boldsymbol{u'_t}$ represents the element displacement field in local coordinates. This equivalence is due to the fact that equation (2.32) represents the virtual work done by the tractions for a traction element, which is a scalar quantity that is independent of coordinate system. The displacement field in local coordinates may be transformed back to global coordinates through:

$$\boldsymbol{u'_t} = \boldsymbol{T}\boldsymbol{u_t} \tag{2.33}$$

where $\boldsymbol{T}$ is the transformation matrix, which is defined as:

IIM21. Transformation Matrix

$$\boldsymbol{T} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \tag{2.34}$$

and θ is given by:

$$\tan\theta = \frac{y_2 - y_1}{x_2 - x_1} \tag{2.35}$$

as shown in Figure 2.2. Substituting equations (2.19) and (2.33) along with **IIM20** into equation (2.32) and performing a change of coordinates gives the traction term as:

$$\int_{S_t} \delta(\boldsymbol{u_t})^T \boldsymbol{t_t} dS_t = \delta\boldsymbol{a_t}^T \int_0^1 \boldsymbol{N_t}^T \boldsymbol{T}^T \boldsymbol{N_t} \boldsymbol{\bar{t}'_t} l_t ds \tag{2.36}$$

where it is noted that the discretized virtual element displacements, $\delta\boldsymbol{a_t}$, may be factored out of the integral since they are constant for a given point in time and the Jacobian of transformation is l_t, which is the length of the traction element.

Combining **IIM11**, **IIM13**, **IIM19**, and equation (2.36) yields the summation of the weak form of equilibrium over the elements in discretized form as:

$$\begin{aligned}\sum_{e=1}^{N_{ele}} \left(\delta\boldsymbol{a_e}^T \int_{V_e} \boldsymbol{N_e}^T \rho_e \boldsymbol{N_e} \boldsymbol{\ddot{a}_e} dV_e + \delta\boldsymbol{a_e}^T \int_{V_e} \boldsymbol{B_e}^T \boldsymbol{D_e} \boldsymbol{B_e} \boldsymbol{a_e} dV_e\right) \\ = \sum_{t=1}^{N_{trc}} \left(\delta\boldsymbol{a_t}^T \int_0^1 \boldsymbol{N_t}^T \boldsymbol{T}^T \boldsymbol{N_t} \boldsymbol{\bar{t}'_t} l_t ds\right) \\ + \sum_{e=1}^{N_{ele}} \left(\delta\boldsymbol{a_e}^T \int_{V_e} \boldsymbol{N_e}^T \rho_e \boldsymbol{f_e} dV_e \right. \\ \left. -\delta\boldsymbol{a_e}^T \int_{V_e} \boldsymbol{B_e}^T \boldsymbol{\sigma_{0e}} dV_e + \delta\boldsymbol{a_e}^T \int_{V_e} \boldsymbol{B_e}^T \boldsymbol{D_e} \boldsymbol{\varepsilon_{0e}} dV_e\right)\end{aligned} \tag{2.37}$$

When the summations in equation (2.37) are performed, accounting for element connectivity appropriately, the equilibrium expression can be written as:

$$\delta\boldsymbol{a_s}^T \left(\boldsymbol{M}\boldsymbol{\ddot{a}_s} + \boldsymbol{K}\boldsymbol{a_s}\right) = \delta\boldsymbol{a_s}^T \boldsymbol{F} \tag{2.38}$$

where $\boldsymbol{M}$ is the global mass matrix, $\boldsymbol{K}$ is the global stiffness matrix, and $\boldsymbol{F}$ is the global load vector. Note that the subscript s indicates that the equation now refers to the entire structural subdomain, as is also implied by the term "global" for the mass, stiffness, and load terms. The global mass matrix is given by:

IIM22. Global Mass Matrix

$$\boldsymbol{M} = \sum_{e=1}^{N_{ele}} \left(\int_{A_e} \boldsymbol{N_e}^T \rho_e \boldsymbol{N_e} dA_e\right) \tag{2.39}$$

where the volume integral is equivalent to an area integral for a unit thickness. Owing to the fact that the matrices B_e and D_e are constant the constant stress triangular body element, the global stiffness matrix simplifies to:

IIM23. Global Stiffness Matrix

$$\boldsymbol{K} = \sum_{e=1}^{N_{ele}} \left(\boldsymbol{B_e^T} \boldsymbol{D_e} \boldsymbol{B_e} A_e \right) \tag{2.40}$$

The global load vector for a given point in time is:

IIM24. Global Load Vector

$$\boldsymbol{F} = \sum_{t=1}^{N_{trc}} \left(\int_0^1 \boldsymbol{N_t^T} \boldsymbol{T}^T \boldsymbol{N_t} \boldsymbol{\bar{t}'_t} l_t ds \right) + \sum_{e=1}^{N_{ele}} \left(\int_{A_e} \boldsymbol{N_e^T} \rho_e \boldsymbol{f_e} dA_e - \boldsymbol{B_e^T} \boldsymbol{\sigma_{0e}} A_e + \boldsymbol{B_e^T} \boldsymbol{D_e} \boldsymbol{\varepsilon_{0e}} A_e \right) \tag{2.41}$$

One may observe that $\boldsymbol{M}$ and $\boldsymbol{K}$ are constant since they depend only on material properties and element geometry, while $\boldsymbol{F}$ may vary with time since both the boundary tractions, $\boldsymbol{\bar{t}'_t}$, and the body acceleration field, $\boldsymbol{f_e}$, may vary over time. Since equation (2.38) must hold true for an arbitrary virtual displacement field, the undamped equation of motion is:

$$\boldsymbol{M} \boldsymbol{\ddot{a}_s} + \boldsymbol{K} \boldsymbol{a_s} = \boldsymbol{F} \tag{2.42}$$

Finally, it is common to include a viscous damping term, which gives a force component that is proportional to the velocity field [5]. As such, the form of the equation of motion adopted for the PDE Solver module is:

IIM25. Equation of Motion for the Structural Subdomain

$$\boldsymbol{M} \boldsymbol{\ddot{a}_s} + \boldsymbol{C} \boldsymbol{\dot{a}_s} + \boldsymbol{K} \boldsymbol{a_s} = \boldsymbol{F} \tag{2.43}$$

where $\boldsymbol{C}$ is the damping matrix. Since the nature of damping in dynamic structural systems is not well understood, the most common method for estimating the damping matrix is by writing it as a linear combination of the mass matrix and the stiffness matrix [5]:

IIM26. Damping Matrix for Structural Subdomain (Rayleigh Damping)

$$\boldsymbol{C} = \alpha \boldsymbol{M} + \xi \boldsymbol{K} \tag{2.44}$$

where α and ξ are scalar constants. This form of damping is sometimes referred to as Rayleigh damping [5]. This damping model will be adopted in the PDE Solver module, but with the caveat that selection of the terms α and ξ will be left to the user as they are problem dependent and require engineering judgement. Reference [5] provides an excellent overview on how to select reasonable values for structural systems.

2.3.3 Temporal Discretization

The numerical technique selected for solving the PDEs involved in DynSWS over the temporal domain is the Newmark family of time-stepping techniques. This section provides an overview of the instanced models for the temporal domain solver, but does not delve into the implications of some of the details of the technique. Readers interested in learning more should see the original paper by Newmark [8]. Reference [9] and [10] provide an excellent summary of various time-integration schemes used in structural dynamics using similar notation for easier comparison (the Newmark family is presented in reference [10]). In addition, references [9] and [10] provide practical advice on the implementation of time-integration schemes in scientific computing, as does reference [5]. The reader should note regardless of the time-integration scheme adopted, **IIM1** and, by extension, **IIM25** must be satisfied at all points in time.

The Newmark family of time-integration schemes begins by writing the truncated Taylor series for the displacement and velocity fields (truncated after the time derivative of acceleration) as:

$$\boldsymbol{a_s}^{t+\Delta t} = \boldsymbol{a_s}^t + \Delta t \dot{\boldsymbol{a}}_{\boldsymbol{s}}^t + \tfrac{1}{2}\Delta t^2 \ddot{\boldsymbol{a}}_{\boldsymbol{s}}^t + \beta \Delta t^3 \dddot{\boldsymbol{a}}_{\boldsymbol{s}}^t \tag{2.45}$$

$$\dot{\boldsymbol{a}}_{\boldsymbol{s}}^{t+\Delta t} = \dot{\boldsymbol{a}}_{\boldsymbol{s}}^t + \Delta t \ddot{\boldsymbol{a}}_{\boldsymbol{s}}^t + \gamma \Delta t^2 \dddot{\boldsymbol{a}}_{\boldsymbol{s}}^t \tag{2.46}$$

where β and γ are scalar coefficients, Δt is the time-step, superscript t and $t + \Delta t$ indicate the value at the current time and the value after one time-step, respectively, and all other terms are as defined previously. Next, it is assumed that acceleration varies linearly over a time step, which may be written as a forward difference:

$$\dddot{\boldsymbol{a}}_{\boldsymbol{s}}^t = \frac{\ddot{\boldsymbol{a}}_{\boldsymbol{s}}^{t+\Delta t} - \ddot{\boldsymbol{a}}_{\boldsymbol{s}}^t}{\Delta t} \tag{2.47}$$

Substituting equation (2.47) into equations (2.45) and (2.46) yields the equations for updating displacement and velocity over a time-step:

IIM27. Update Equations for Displacement and Velocity Fields

$$\boldsymbol{a_s}^{t+\Delta t} = \boldsymbol{a_s}^t + \Delta \boldsymbol{a_s}^t \tag{2.48}$$

$$\dot{\boldsymbol{a}}_{\boldsymbol{s}}^{t+\Delta t} = \dot{\boldsymbol{a}}_{\boldsymbol{s}}^t + \Delta \dot{\boldsymbol{a}}_{\boldsymbol{s}}^t \tag{2.49}$$

where the incremental changes in displacement and velocity are given by:

IIM28. Incremental Displacement and Velocity Fields

$$\Delta \boldsymbol{a_s}^t = \Delta t \dot{\boldsymbol{a}}_{\boldsymbol{s}}^t + \tfrac{1}{2}\Delta t^2 \left[(1-2\beta)\ddot{\boldsymbol{a}}_{\boldsymbol{s}}^t + 2\beta \ddot{\boldsymbol{a}}_{\boldsymbol{s}}^{t+\Delta t}\right] \tag{2.50}$$

$$\Delta \dot{\boldsymbol{a}}_{\boldsymbol{s}}^t = \Delta t \left[(1-\gamma)\ddot{\boldsymbol{a}}_{\boldsymbol{s}}^t + \gamma \ddot{\boldsymbol{a}}_{\boldsymbol{s}}^{t+\Delta t}\right] \tag{2.51}$$

The stress and strain at the next time step are given by:

IIM29. Update Equations for Stress and Strain Fields

$$\boldsymbol{\sigma_s}^{t+\Delta t} = \boldsymbol{\sigma_s}^t + \Delta \boldsymbol{\sigma_s}^t \tag{2.52}$$

$$\boldsymbol{\varepsilon_s}^{t+\Delta t} = \boldsymbol{\varepsilon_s}^t + \Delta \boldsymbol{\varepsilon_s}^t \tag{2.53}$$

where, making use of **IIM13**, the incremental strain in an element is:

IIM30. Incremental Strain Field

$$\Delta \boldsymbol{\varepsilon}_{\boldsymbol{s}}^{t} = \sum_{e=1}^{N_{ele}} \Delta \boldsymbol{\varepsilon}_{\boldsymbol{e}}^{t} = \sum_{e=1}^{N_{ele}} \left(\boldsymbol{B}_{\boldsymbol{e}} \Delta \boldsymbol{a}_{\boldsymbol{e}}^{t} \right) \tag{2.54}$$

and, making use of **IIM8**, the incremental stress is:

IIM31. Incremental Stress Field

$$\Delta \boldsymbol{\sigma}_{\boldsymbol{s}}^{t} = \sum_{e=1}^{N_{ele}} \left(\boldsymbol{D}_{\boldsymbol{e}} \Delta \boldsymbol{\varepsilon}_{\boldsymbol{e}}^{t} \right) \tag{2.55}$$

Note that, for implementation purposes, the stress and strain increments must be evaluated at the spatial discretization element level and summed over all element taking into account the connectivity between elements to obtain the incremental stress and strain for the total structural subdomain.

Writing the equation of motion (**IIM25**) at time $t + \Delta t$ gives:

$$\boldsymbol{M} \ddot{\boldsymbol{a}}_{\boldsymbol{s}}^{t+\Delta t} + \boldsymbol{C} \dot{\boldsymbol{a}}_{\boldsymbol{s}}^{t+\Delta t} + \boldsymbol{K} \boldsymbol{a}_{\boldsymbol{s}}^{t+\Delta t} = \boldsymbol{F}^{t+\Delta t} \tag{2.56}$$

Substituting **IIM27** into equation (2.56) and rearranging gives the update equation for the acceleration field:

IIM32. Update Equation for Acceleration Field

$$\boldsymbol{M'} \ddot{\boldsymbol{a}}_{\boldsymbol{s}}^{t+\Delta t} = \boldsymbol{P}^{t+\Delta t} \tag{2.57}$$

where $\boldsymbol{M'}$ and $\boldsymbol{P}^{t+\Delta t}$ are given by:

IIM33. Modified Mass Matrix and Load Vector

$$\boldsymbol{M'} = \boldsymbol{M} + \gamma \Delta t \boldsymbol{C} + \beta \Delta t^2 \boldsymbol{K} \tag{2.58}$$

$$\boldsymbol{P}^{t+\Delta t} = \boldsymbol{F}^{t+\Delta t} - \boldsymbol{C} \left[\dot{\boldsymbol{a}}_{\boldsymbol{s}}^{t} + \Delta t (1-\gamma) \ddot{\boldsymbol{a}}_{\boldsymbol{s}}^{t} \right] - \boldsymbol{K} \left[\boldsymbol{a}_{\boldsymbol{s}}^{t} + \Delta t \dot{\boldsymbol{a}}_{\boldsymbol{s}}^{t} + \tfrac{1}{2} \Delta t^2 (1-2\beta) \ddot{\boldsymbol{a}}_{\boldsymbol{s}}^{t} \right] \tag{2.59}$$

The reader should note that the modified mass matrix is constant in time provided that a constant time-step is adopted. Also note that selection of the parameters β and γ has implications for the accuracy and stability of the technique. In terms of accuracy, the Newmark family of time-stepping algorithms is second order accurate if $\gamma = \frac{1}{2}$ [8, 10]. Unconditional stability is guaranteed for $2\beta \geq \gamma \geq \frac{1}{2}$ [10], otherwise, the family of techniques is conditionally stable at best. Two popular members of the Newmark family are those for which $\gamma = \frac{1}{2}$, $\beta = \frac{1}{4}$ (implicit, unconditionally stable) and $\gamma = \frac{1}{2}$, $\beta = \frac{1}{6}$ (implicit, conditionally stable) [10]. In addition, the Newmark formulation reduces to the central difference time-stepping scheme for $\gamma = \frac{1}{2}$, $\beta = 0$ [10].

In broad strokes, the time-stepping algorithm based on the Newmark family proceeds as follows (note that superscript 0 refers to the value at $t = 0$, not raising to the power of 0):

1. Select Δt, β, and γ.
2. Begin with initial conditions $\boldsymbol{a}_{\boldsymbol{s}}^{0}$ and $\dot{\boldsymbol{a}}_{\boldsymbol{s}}^{0}$.
3. Solve $\boldsymbol{M}\ddot{\boldsymbol{a}}_{\boldsymbol{s}}^{0} = \boldsymbol{F}^{0} - \boldsymbol{C}\dot{\boldsymbol{a}}_{\boldsymbol{s}}^{0} - \boldsymbol{K}\boldsymbol{a}_{\boldsymbol{s}}^{0}$ for $\ddot{\boldsymbol{a}}_{\boldsymbol{s}}^{0}$.
4. Compute $\boldsymbol{M'}$.
5. Compute $\boldsymbol{P}^{t+\Delta t}$.
6. Solve $\boldsymbol{M'}\ddot{\boldsymbol{a}}_{\boldsymbol{s}}^{t+\Delta t} = \boldsymbol{P}^{t+\Delta t}$ for $\ddot{\boldsymbol{a}}_{\boldsymbol{s}}^{t+\Delta t}$.
7. Compute $\Delta\boldsymbol{a}_{\boldsymbol{s}}^{t}$ and $\Delta\dot{\boldsymbol{a}}_{\boldsymbol{s}}^{t}$.
8. Compute $\Delta\boldsymbol{\varepsilon}_{\boldsymbol{s}}^{t}$ and $\Delta\boldsymbol{\sigma}_{\boldsymbol{s}}^{t}$.
9. Compute $\boldsymbol{a}_{\boldsymbol{s}}^{t+\Delta t}$, $\dot{\boldsymbol{a}}_{\boldsymbol{s}}^{t+\Delta t}$, $\boldsymbol{\varepsilon}_{\boldsymbol{s}}^{t+\Delta t}$, and $\boldsymbol{\sigma}_{\boldsymbol{s}}^{t+\Delta t}$.
10. Update the values of t, $\boldsymbol{a}_{\boldsymbol{s}}^{t}$, $\dot{\boldsymbol{a}}_{\boldsymbol{s}}^{t}$, $\ddot{\boldsymbol{a}}_{\boldsymbol{s}}^{t}$, $\boldsymbol{\varepsilon}_{\boldsymbol{s}}^{t}$, and $\boldsymbol{\sigma}_{\boldsymbol{s}}^{t}$.
11. Repeat steps 5–10 until the end of the analysis period.

2.3.4 Summary of Inputs and Outputs

Table 2.1 provides a summary of the input data for the PDE Solver. Table 2.2 provides a summary of the output data from the PDE Solver.

Table 2.1: Summary of input data

Symbol	Name	Units	Type
N_{nod}	Number of nodes	unitless	$\mathbb{N}$
N_{ele}	Number of body elements	unitless	$\mathbb{N}$
N_{trc}	Number of traction elements	unitless	$\mathbb{N}$
nodes	Geometry nodes	L	set of tuples of $\mathbb{R}$
elements	Connectivity of body elements	unitless	set of tuples of $\mathbb{N}$
tractions	Connectivity of traction elements	unitless	set of tuples of $\mathbb{N}$
ρ	Density	$M \cdot L^{-3}$	set of $\mathbb{R}$
E	Elastic modulus	$M^{-1} \cdot L \cdot T^{-2}$	set of $\mathbb{R}$
ν	Poisson's ratio	unitless	set of $\mathbb{R}$
Δt	Time step	T	$\mathbb{R}$
β	Time-stepping coefficient	unitless	$\mathbb{R}$
γ	Time-stepping coefficient	unitless	$\mathbb{R}$
$\boldsymbol{a_s^0}$	Initial displacement field	L	vector of $\mathbb{R}$
$\boldsymbol{\dot{a}_s^0}$	Initial velocity field	$L \cdot T^{-1}$	vector of $\mathbb{R}$
$\boldsymbol{\sigma_s^0}$	Initial stress field	$M^{-1} \cdot L \cdot T^{-2}$	vector of $\mathbb{R}$
$\boldsymbol{\varepsilon_s^0}$	Initial strain field	unitless	vector of $\mathbb{R}$
$\boldsymbol{f_e}$	Time varying applied acceleration	$L \cdot T^{-2}$	set of vectors of $\mathbb{R}$
$\boldsymbol{\bar{t}_t'}$	Time varying tractions	$M^{-1} \cdot L \cdot T^{-2}$	set of vectors of $\mathbb{R}$

Table 2.2: Summary of output data

Symbol	Name	Units	Type
$\boldsymbol{a_s}$	Time varying displacement field	L	set of vectors of $\mathbb{R}$
$\boldsymbol{\dot{a}_s}$	Time varying velocity field	$L \cdot T^{-1}$	set of vectors of $\mathbb{R}$
$\boldsymbol{\ddot{a}_s}$	Time varying acceleration field	$L \cdot T^{-1}$	set of vectors of $\mathbb{R}$
$\boldsymbol{\sigma_s}$	Time varying stress field	$M^{-1} \cdot L \cdot T^{-2}$	set of vectors of $\mathbb{R}$
$\boldsymbol{\varepsilon_s}$	Time varying strain field	unitless	set of vectors of $\mathbb{R}$

3 Module Guide for PDE Solver

3.1 Potential Changes

This section lists changes that may occur in the design. It is important to consider potential changes at this stage since they will have an important influence on the module decomposition. In particular, Section 3.1.1 lists changes that are likely to occur and that the module decomposition will specifically aim to accommodate. Section 3.1.2 lists changes that are possible, but not very likely to occur; although this second list of changes will be kept in mind, the design will not specifically target the ability to easily make the changes that are deemed unlikely.

3.1.1 Anticipated Changes

This section lists changes that are likely to be made to the PDE Solver for DynSWS, which will guide its design, chiefly, the module decomposition. The first set of anticipated changes relate to relate to the theoretical and instanced models (some of which are repeated from the MG):

AC1. **Homogeneity of materials.** Natural materials such as soil and rock often exhibit properties that vary over space (*e.g.* increasing stiffness with depth). In researching the behaviour of these materials, it is likely that a material model accounting for such variation may be required.

AC2. **Isotropy of materials.** Natural materials can also exhibit properties that vary with direction (*e.g.* due to preferred orientation caused by sedimentation). Similar to **AC1**, material models accounting for anisotropy may also be incorporated in DynSWS.

AC3. **Consideration of large deformations and large strains.** The first implementation of DynSWS will not account for large strains, but for analysis of conditions approaching and exceeding failure it is likely that this assumption will need to be modified.

AC4. **The material model for the structural subdomain.** Initially, the structural subdomain will be modelled as linear elastic. Materials such as concrete, of which the type of structures that DynSWS is intended to model are often constructed, only behave in this manner for small strains. Coinciding with **AC3**, the material model for the structural subdomain is likely to change.

AC5. **The use of a Cartesian coordinate system.** As mentioned in the SRS for DynSWS, certain types of geometry for soil-water-structure interaction problems are best represented in coordinate systems other than the Cartesian system (*e.g.* cylindrical coordinates).

AC6. **The plane strain assumption for the 2-D model.** Along with modification of the coordinate system, as mentioned in **AC5**, the assumptions of the 2-D model are likely to change (*e.g.* to axisymmetric conditions).

The second set of anticipated changes relate to the numerical algorithm that will implement the services of the PDE Solver module:

AC7. **The top-level algorithm for solving the systems of partial differential equations representing each subdomain.** It is foreseeable that certain techniques for solving partial differential equations may be more amenable to one type of subdomain than others, or that changing the formulation from small strain to large strain (see **AC3**) may require a different algorithm.

AC8. **The algorithm for solving for linear systems of equations.** Most algorithms for solving partial differential equations involve setting up a system of linear equations that must be solved. Depending on the characteristics of the partial differential equation solver, certain types of linear solver may be more efficient than others.

AC9. **The technique used to discretize the system in the spatial domain for a given point in time.** The technique currently selected for the structural subdomain is the finite element method, but this may change for different subdomains and/or for analysis approaching and post-failure.

AC10. **The form of the stress-strain (constitutive) matrix.** This item is likely to change with the material model and the dimension of the spatial domain.

AC11. **Interpolation functions for body element.** This item will change depending on the type and shape of discretization element.

AC12. **Interpolation functions for traction element.** This item will change for similar reasons to those described in **AC11**.

AC13. **The form of the linear differential operator relating strains and displacements.** This will change depending on the dimension of the solution domain.

AC14. **The kinematic matrix relating discretized displacements to discretized strains.** This will change along with **AC11** and **AC13**.

AC15. **The algorithm for computing the area of a body element.** This involves computing a determinant.

AC16. **The algorithm for computing the kinematic matrix of a body element.** This involves building a matrix.

AC17. **The form of the mass matrix.** The form currently documented is referred to as the "consistent" mass matrix. It is possible that this will be changed to a "lumped" mass matrix, which has the desirable property of being a diagonal matrix.

AC18. **The algorithm for integrating the mass matrix.** This will always involve integrating at the element level and summing over the elements taking into account connectivity. However, it is foreseeable that the integration at the element level may switch between possible options of closed-form integration and Gaussian quadrature for performance reasons.

AC19. The algorithm for integrating the stiffness matrix. This is likely to change in a similar fashion to that described in **AC18**.

AC20. The algorithm for integrating the traction term in the load vector. This is likely to change in a similar fashion to that described in **AC18**.

AC21. The algorithm for integrating the body force term in the load vector. This is likely to change in a similar fashion to that described in **AC18**.

AC22. The algorithm for integrating the initial stress term in the load vector. This is likely to change in a similar fashion to that described in **AC18**.

AC23. The algorithm for integrating the initial strain term in the load vector. This is likely to change in a similar fashion to that described in **AC18**.

AC24. The technique for computing the damping matrix. This involves computing a linear combination of the mass and stiffness matrices.

AC25. The algorithm used to solve the problem in the temporal domain. The currently selected technique is the Newmark family of time-stepping algorithms. It is foreseeable that other techniques, such as the Runge-Kutta family, may be adopted in the future for performance comparison.

AC26. The values of the constants used in the time-stepping scheme. This is likely to change in order to optimize the performance of the time-stepping component.

AC27. The algorithm for computing the initial acceleration field. This involves setting up and solving a linear system of equations based on the initial displacement and velocity fields, the initial load vector, the mass matrix, the damping matrix, and the stiffness matrix.

AC28. The algorithm for computing the load vector for the next time step. This is likely to change in a similar fashion to that described in **AC18**.

AC29. The algorithm for computing the updated acceleration field. This involves setting up and solving a linear system of equations based on the modified mass matrix and the load vector for the next time step.

AC30. The algorithm for updating the displacement and velocity fields. This involves computing the incremental changes in these field variables and updating the global field variables.

AC31. The algorithm for updating the stress and strain fields. This involves computing the incremental changes in these field variables and updating the global field variables.

3.1.2 Unlikely Changes

This section lists changes that are not considered likely to occur. The design of the PDE Solver for DynSWS will not necessarily ensure that these changes are easy to make. The first set of unlikely changes relate to the theoretical and instanced models (some of which are repeated from the MG):

UC1. **Time dependency of load input.** Since static loading may be considered as a special case of dynamic loading (where the frequency of the loading is zero), there is no reason to modify the code specifically for the case of static loading.

UC2. **The types of subdomain that make up the problem domain: structure, fluid, and soil.** Although the details of the modelling of these subdomains may change, it is not expected that additional types of subdomain will need to be accomodated.

UC3. **The functional goals, which are to compute the displacement, velocity, and acceleration response of the system and to compute the stress and strain fields.** These are the basic goals for any model of the response of physical objects to dynamic loading.

UC4. **The assumption that the domain is isothermal.** Temperature gradients within the problem domain are not expected to have a significant influence on the model.

UC5. **The assumption that there are no sources or sinks of material internal to the domain.** DynSWS is intended to deal with problems where the materials are either present in the model or entering and exiting from the boundaries. It should not be difficult to construct any soil-water-structure interaction problem to accommodate this assumption.

UC6. **The neglect of relativistic effects.** It is inconceivable that the materials modelled using DynSWS would approach even a small fraction of the speed of light.

UC7. **Continuum mechanics modelling framework.** Molecular level interactions are not likely to influence the model.

The second set of unlikely changes relate to the numerical algorithm that will implement the services of the PDE Solver module:

UC8. **The use of a damping matrix that is linearly proportional to the mass and stiffness matrices (Rayleigh damping) for the structural subdomain.** This technique for estimating the damping is common for dynamic analysis of structures [5]. Since the true nature of damping in structures is difficult to ascertain and this is not the focus of the author's studies, it is unlikely that the model implemented in DynSWS will change the form of damping to a more complicated model.

UC9. **The use of a direct time-stepping algorithm to obtain the solution in the temporal domain.** Another option is to reformulate the problem and obtain the solution in the frequency domain. From the frequency domain, the solution in the temporal domain may be obtained through a discrete Fourier transform (DFT). However, this type of analysis is more appropriate for periodic loading. Since one of the primary loading types that will be of interest is that due to seismic activity, it is unlikely that frequency domain analysis will be appropriate since this type of loading tends to be impulsive and/or non-periodic in nature.

3.2 Module Specification

This section presents the modular decomposition of the PDE Solver for DynSWS. Section 3.2.1 summarizes the modular decomposition in tabular form. Section 3.2.2 lists each module and provides the secret, the service, and (optionally) the prefix for each of the modules at the lowest level.

3.2.1 Module Hierarchy

Typical of modular decomposition based on the principle of information hiding are three modules at the highest level: machine hiding, behaviour hiding, and software decision hiding. The machine hiding module involves the interaction between the virtual realm of software and the physical realm of hardware; Table 3.1 shows the machine hiding module decomposition for DynSWS modules used by the PDE Solver module. The behaviour hiding module is concerned with items such as output formatting and text messages; Table 3.2 shows the behaviour hiding module decomposition for DynSWS modules used by the PDE Solver module. The software decision hiding module includes items such as internal data structures and important algorithms; Table 3.3 shows the software decision hiding module decomposition DynSWS modules used by the PDE Solver module and Table 3.4 shows the software decision hiding module decomposition within the PDE Solver module. Note that the services of some of the modules (particularly in the machine hiding module) may not be implemented in DynSWS as they are provided by the programming language or the operating system, but they are listed here nonetheless for completeness as well as awareness of the dependencies of DynSWS on outside systems. In addition, some of the modules listed here are already documented in the MG for DynSWS; these items are marked with an asterisk (*) and are only listed here because the PDE Solver uses these modules.

Table 3.1: Decomposition of the machine hiding module of the DynSWS system (showing components used by PDE Solver)

Level 1	Level 2	Level 3
Machine Hiding	Physical Data Operations	*Integer Operations
		*Floating Point Operations

Table 3.2: Decomposition of the behaviour hiding module of the DynSWS system (showing components used by PDE Solver)

Level 1	Level 2	Level 3
Behaviour Hiding	Log Message Handling	*Log Message Control

Table 3.3: Decomposition of the software decision hiding module of the DynSWS system (showing components used by PDE Solver)

Level 1	Level 2	Level 3
Software Decision Hiding	*System Constants	
	Data Structures	*Domain Data
		*Boundary Data
		*Material Property Data
	PDE Solver	

Table 3.4: Decomposition of the software decision hiding module of the PDE Solver

Level 1	Level 2	Level 3	Level 4
Software Decision Hiding	PDE Solver Control		
	PDE Solver Constants		
	Data Structures	Dense Matrix	
		Banded Symmetric Matrix	
		Vector	
	Integration Algorithms	Body Element Integration	
		Traction Element Integration	
	Interpolation	Body Element Interpolation	
		Traction Element Interpolation	
	Material Model	Linear Elastic Model	
		Constitutive Matrix	
		Kinematic Matrix	
	Linear Algebra	Linear Solver	

3.2.2 Module Decomposition

This section details each of the lowest level modules ("leaf" modules) in the design of the PDE Solver of DynSWS. In accordance with the design principle of information hiding, each leaf module has one secret and provides one service. The goal is to keep the scope of each leaf module relatively small and self-contained so that each can be viewed as a work assignment. The fact that each module maintains a secret allows different modules to be worked on in parallel, provided that the interface to the module is specified. That is, the implementation details of the module's service are isolated. This type of design also facilitates future changes to the software product as an individual change is ideally isolated to a single leaf module (provided that it comes from the list of anticipated changes in Section 3.1.1). Finally, some leaf modules are assigned a naming convention prefix to avoid naming conflicts in the implementation.

3.2.2.3 Software Decision Hiding

3.2.2.3.3 PDE Solver Control

Secret: The algorithm for solving a system of partial differential equations.

Service: Compute the solution to a system of partial differential equations.

Prefix: pde_

3.2.2.3.4 PDE Solver Constants

Secret: The values of constants involved in the PDE Solver algorithm.

Service: Return the values of constants involved in the PDE Solver algorithm.

Prefix: N/A

3.2.2.3.5 Data Structures

3.2.2.3.5.1 Dense Matrix ADT

Secret: The data structure for a dense (not banded or sparse) matrix.

Service: Provide access routines for dense matrix data type.

Prefix: dm_

3.2.2.3.5.2 Banded Symmetric Matrix ADT

Secret: The data structure for a banded symmetric matrix.

Service: Provide access routines for banded symmetric matrix data type.

Prefix: bsm_

3.2.2.3.5.3 Vector ADT

Secret: The data structure for a vector.

Service: Provide access routines for vector data type.

Prefix: vec_

3.2.2.3.6 Integration Algorithms

3.2.2.3.6.1 Body Element Integration

Secret: The algorithm for integrating properties over body elements.

Service: Integrate a quantity over a body element.

Prefix: bodyint_

3.2.2.3.6.2 Traction Element Integration

Secret: The algorithm for integrating properties over traction elements.

Service: Integrate a quantity over a traction element.

Prefix: tracint_

3.2.2.3.7 Interpolation

3.2.2.3.7.1 Body Element Interpolation

Secret: The interpolation algorithm for body elements.

Service: Compute the value of a quantity within a body element.

Prefix: bodyinterp_

3.2.2.3.7.2 Traction Element Interpolation

Secret: The interpolation algorithm for traction elements.

Service: Compute the value of a quantity within a traction element.

Prefix: tracinterp_

3.2.2.3.8 Material Models

3.2.2.3.8.1 Linear Elastic Model

Secret: The algorithm for computing stress and strain in a linear elastic material.

Service: Update the stress and strain at a point.

Prefix: N/A

3.2.2.3.8.2 Constitutive Matrix

Secret: The algorithm for computing the constitutive matrix.

Service: Compute the constitutive matrix.

Prefix: N/A

3.2.2.3.8.3 Kinematic Matrix

Secret: The algorithm for computing the kinematic matrix.

Service: Compute the kinematic matrix.

Prefix: N/A

3.2.2.3.9 Linear Algebra

3.2.2.3.9.1 Linear Solver

Secret: The algorithm for solving a system of linear equations.

Service: Compute the solution to a system of linear equations (*i.e.* given $[\boldsymbol{A}]\{x\} = \{b\}$ where $[\boldsymbol{A}]$ is constant, find $\{x\}$).

Prefix: N/A

3.2.3 Uses Hierarchy

This section shows how the various modules in the PDE Solver of DynSWS are interrelated. Figure 3.1 shows the uses hierarchy for the PDE Solver system. Note that only leaf modules are shown as these are the only modules that will actually be implemented (or used from an external source). Also, note that while the uses hierarchy implies the control flow of the program, it does not explicitly display the order in which the modules are called; the uses hierarchy simply shows which modules use other modules. It is important to observe that there are no "closed loops" in the uses hierarchy. This is important as such situations make both implementation and change difficult due to the circular nature of the dependencies. Note that some modules in Figure 3.1 at the lower level of the hierarchy (such as data structures and basic mathematical operations) are used by essentially all other modules. To simplify the presentation of the uses hierarchy, such modules have been surrounded with dashed boxes.

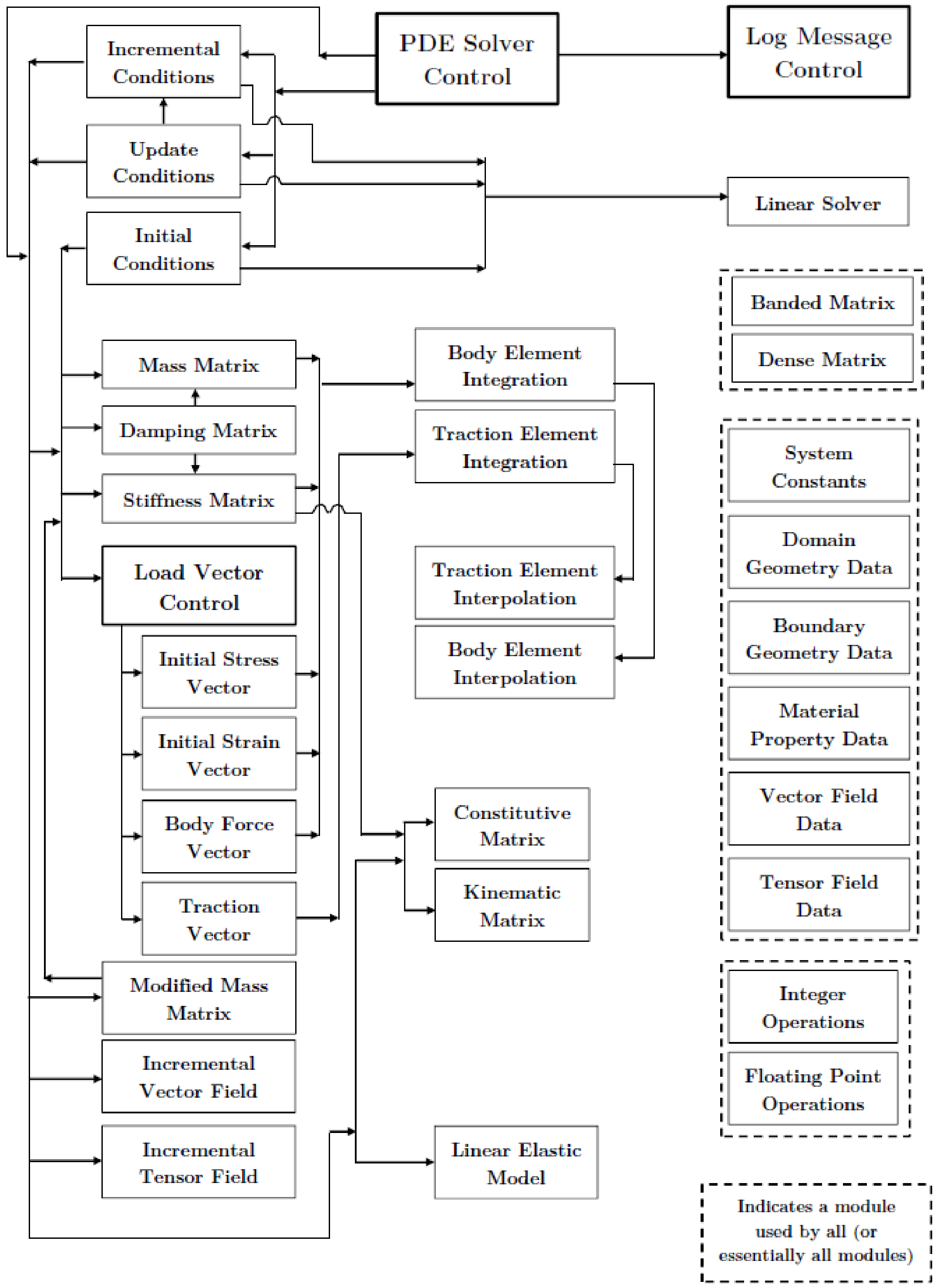

Figure 3.1: Uses hierarchy for modular decomposition of PDE Solver

3.3 Traceability Matrices

Figure 3.2: Traceability matrix for anticipated changes, part 1 of 3

Top-Level Module	Leaf Module	Anticipated Change									
		AC1	**AC2**	**AC3**	**AC4**	**AC5**	**AC6**	**AC7**	**AC8**	**AC9**	**AC10**
1 (Machine Hiding)	1.2.2										
	1.2.3										
2 (Behaviour Hiding)	2.4.1										
3 (Software Decision Hiding)	3.1										
	3.2.1										
	3.2.2										
	3.2.3										
	3.2.4										
	3.2.5										
	3.3			X	X	X		X		X	
	3.4										
	3.5.1										
	3.5.2										
	3.5.3										
	3.5.4										
	3.5.5.1										
	3.5.5.2										
	3.5.5.3										
	3.5.5.4										
	3.5.5.5										
	3.5.6										
	3.5.7										
	3.5.8										X
	3.5.9										
	3.6.1										
	3.6.2										
	3.7.1										
	3.7.2										
	3.7.3										
	3.8.1										
	3.8.2										
	3.9.1	X	X	X	X	X	X				
	3.10.1.1										
	3.10.1.2										
	3.10.2								X		

Figure 3.3: Traceability matrix for anticipated changes, part 2 of 3

Top-Level	Leaf	Anticipated Change									
Module	Module	**AC11**	**AC12**	**AC13**	**AC14**	**AC15**	**AC16**	**AC17**	**AC18**	**AC19**	**AC20**
1 (Machine Hiding)	1.2.2										
	1.2.3										
2 (Behaviour Hiding)	2.4.1										
3 (Software Decision Hiding)	3.1										
	3.2.1					X					
	3.2.2										
	3.2.3										
	3.2.4										
	3.2.5										
	3.3										
	3.4										
	3.5.1							X			
	3.5.2										
	3.5.3										
	3.5.4										
	3.5.5.1										
	3.5.5.2										
	3.5.5.3										
	3.5.5.4										
	3.5.5.5										
	3.5.6										
	3.5.7										
	3.5.8										
	3.5.9			X	X		X				
	3.6.1								X	X	
	3.6.2										X
	3.7.1										
	3.7.2										
	3.7.3										
	3.8.1	X									
	3.8.2		X								
	3.9.1										
	3.10.1.1										
	3.10.1.2										
	3.10.2										

Figure 3.4: Traceability matrix for anticipated changes, part 3 of 3

Top-Level Module	Leaf Module	Anticipated Change										
		AC21	**AC22**	**AC23**	**AC24**	**AC25**	**AC26**	**AC27**	**AC28**	**AC29**	**AC30**	**AC31**
1 (Machine Hiding)	1.2.2											
	1.2.3											
2 (Behaviour Hiding)	2.4.1											
3 (Software Decision Hiding)	3.1											
	3.2.1											
	3.2.2											
	3.2.3											
	3.2.4											
	3.2.5											
	3.3											
	3.4						X					
	3.5.1											
	3.5.2											
	3.5.3											
	3.5.4				X							
	3.5.5.1								X			
	3.5.5.2											
	3.5.5.3											
	3.5.5.4											
	3.5.5.5											
	3.5.6											
	3.5.7											
	3.5.8											
	3.5.9											
	3.6.1	X	X	X								
	3.6.2											
	3.7.1					X		X				
	3.7.2					X					X	X
	3.7.3					X				X		
	3.8.1											
	3.8.2											
	3.9.1											
	3.10.1.1											
	3.10.1.2											
	3.10.2											

Figure 3.5: Traceability matrix for unlikely changes

Top-Level Module	Leaf Module	Unlikely Change								
		UC1	**UC2**	**UC3**	**UC4**	**UC5**	**UC6**	**UC7**	**UC8**	**UC9**
1 (Machine Hiding)	1.2.2									
	1.2.3									
2 (Behaviour Hiding)	2.4.1									
3 (Software Decision Hiding)	3.1						X		X	
	3.2.1					X		X		
	3.2.2							X		
	3.2.3				X	X	X	X	X	X
	3.2.4	X		X	X	X	X	X	X	X
	3.2.5	X		X	X	X	X	X	X	X
	3.3	X	X	X	X	X	X	X	X	X
	3.4									
	3.5.1		X		X	X	X	X	X	
	3.5.2		X		X	X	X	X	X	
	3.5.3		X		X	X	X	X	X	
	3.5.4		X		X	X	X	X	X	
	3.5.5.1	X			X	X	X	X		
	3.5.5.2						X	X		
	3.5.5.3						X	X		
	3.5.5.4	X			X		X	X		
	3.5.5.5	X					X	X		
	3.5.6							X		
	3.5.7							X		
	3.5.8		X		X	X	X	X		
	3.5.9		X			X	X	X		
	3.6.1							X		
	3.6.2							X		
	3.7.1	X		X			X	X		
	3.7.2	X		X			X	X		
	3.7.3	X		X			X	X		
	3.8.1							X		
	3.8.2							X		
	3.9.1		X	X	X		X	X		
	3.10.1.1									
	3.10.1.2									
	3.10.2									

Figure 3.6: Traceability matrix for goal statements

Top-Level Module	Leaf Module	Goal G1	Goal G2
1 (Machine Hiding)	1.2.2		
	1.2.3		
2 (Behaviour Hiding)	2.4.1		
3 (Software Decision Hiding)	3.1		
	3.2.1		
	3.2.2		
	3.2.3		
	3.2.4		
	3.2.5		
	3.3	X	X
	3.4	X	X
	3.5.1	X	X
	3.5.2	X	X
	3.5.3	X	X
	3.5.4	X	X
	3.5.5.1	X	X
	3.5.5.2	X	X
	3.5.5.3	X	X
	3.5.5.4	X	X
	3.5.5.5	X	X
	3.5.6	X	
	3.5.7		X
	3.5.8	X	X
	3.5.9	X	X
	3.6.1		
	3.6.2		
	3.7.1	X	X
	3.7.2	X	X
	3.7.3	X	X
	3.8.1		
	3.8.2		
	3.9.1		X
	3.10.1.1		
	3.10.1.2		
	3.10.2		

Figure 3.7: Traceability matrix for non-functional requirements, part 1 of 2

Top-Level Module	Leaf Module	Non-Functional Requirement						
		NFR1	**NFR2**	**NFR3**	**NFR4**	**NFR5**	**NFR6**	**NFR7**
1 (Machine Hiding)	1.2.2							
	1.2.3							
2 (Behaviour Hiding)	2.4.1							
3 (Software Decision Hiding)	3.1							
	3.2.1							
	3.2.2							
	3.2.3							
	3.2.4							
	3.2.5							
	3.3		X	X	X	X	X	
	3.4							
	3.5.1							
	3.5.2							
	3.5.3							
	3.5.4							
	3.5.5.1							
	3.5.5.2							
	3.5.5.3							
	3.5.5.4							
	3.5.5.5							
	3.5.6							
	3.5.7							
	3.5.8							
	3.5.9							
	3.6.1							
	3.6.2							
	3.7.1		X	X				
	3.7.2		X	X				
	3.7.3		X	X				
	3.8.1							
	3.8.2							
	3.9.1				X	X	X	
	3.10.1.1							
	3.10.1.2							
	3.10.2							

Figure 3.8: Traceability matrix for non-functional requirements, part 2 of 2

Top-Level Module	Leaf Module	Non-Functional Requirement						
		NFR8	**NFR9**	**NFR10**	**NFR11**	**NFR12**	**NFR13**	**NFR14**
1 (Machine Hiding)	1.2.2							
	1.2.3							
2 (Behaviour Hiding)	2.4.1	X			X		X	
3 (Software Decision Hiding)	3.1							
	3.2.1			X		X		
	3.2.2			X		X		
	3.2.3			X		X		
	3.2.4			X		X		
	3.2.5			X		X		
	3.3		X	X				
	3.4							
	3.5.1							
	3.5.2							
	3.5.3							
	3.5.4							
	3.5.5.1							
	3.5.5.2							
	3.5.5.3							
	3.5.5.4							
	3.5.5.5							
	3.5.6					X		
	3.5.7					X		
	3.5.8			X				
	3.5.9			X				
	3.6.1			X				
	3.6.2			X				
	3.7.1		X	X				
	3.7.2		X	X				
	3.7.3		X	X				
	3.8.1			X				
	3.8.2			X				
	3.9.1			X				
	3.10.1.1							
	3.10.1.2							
	3.10.2							

4 Module Interface Specification for PDE Solver

4.3.3 PDE Solver Control

Uses

Modules:
Banded Symmetric Matrix ADT
Boundary Data
Body Element Integration
Dense Matrix ADT
Field Data
Floating Point Operations
Integer Operations
Log Message Control
Log Messages
Material Property Data
PDE Solver Constants
System Constants
Traction Element Integration
Vector ADT

Syntax

Exported Constants
N/A

Exported Types
N/A

Exported Functions

Table 4.1: Exported function interfaces for PDE Solver Control module

Name	Input	Output	Exceptions
pde_init			
pde_clean			
pde_buildMassMatrix			
pde_buildStiffMatrix			
pde_buildDampMatrix	real, real		
pde_buildModMassMatrix			
pde_buildLoadVector	integer		
pde_initAcc			
pde_incAcc			
pde_incDisp			
pde_incVel			
pde_incStrain			
pde_incStress			
pde_updateAcc			
pde_updateDisp			
pde_updateVel			
pde_updateStrain			
pde_updateStress			

Semantics

State Variables

hbw : integer
nnod : integer
nel : integer
nelb : integer
ndof : integer
mass : bandSymMatrixT
modMass : bandSymMatrixT
stiff : bandSymMatrixT
damp : bandSymMatrixT
initStress : vectorT
initStrain : vectorT
body : vectorT
trac : vectorT
load : vectorT
prevDisp : vectorT
incDisp : vectorT
newDisp : vectorT
prevVel : vectorT
incVel : vectorT
newVel : vectorT
prevAcc : vectorT
newAcc : vectorT
prevStress : vectorT
incStress : vectorT
newStress : vectorT
prevStrain : vectorT
incStrain : vectorT
newStrain : vectorT

State Invariants
N/A

Assumptions

1. The function pde_init() will be called before all other functions in this module.

Access Routine Semantics

pde_init():

transition: $ndof$:= fld_numDof()

```
    $hbw$ := compute_hbw()
    $nel$ := fld_numElem()
    $nnod$ := fld_numNode()
    $mass$ := new bandSymMatrixT($ndof$,$hbw$)
    $stiff$ := new bandSymMatrixT($ndof$,$hbw$)
    $initStress$ := new vectorT($ndof$)
    $initStrain$ := new vectorT($ndof$)
    $body$ := new vectorT($ndof$)
    $trac$ := new vectorT($ndof$)
    $load$ := new vectorT($ndof$)
    $prevDisp$ := new vectorT($ndof$)
    $incDisp$ := new vectorT($ndof$)
    $newDisp$ := new vectorT($ndof$)
    $prevVel$ := new vectorT($ndof$)
    $incVel$ := new vectorT($ndof$)
    $newVel$ := new vectorT($ndof$)
    $prevAcc$ := new vectorT($ndof$)
    $newAcc$ := new vectorT($ndof$)
        $\forall i \in [1..ndof]$
        {
            ($\forall j \in [1..nnod]$
            {
                $\forall k \in$ [1..NDIM]
                {
                    $i$ = fld_getDof($j$,$k$) $\rightarrow$
                        $prevDisp$.vec_set( $i$, fld_getDisp($j$,$k$) )
                        $prevVel$.vec_set( $i$, fld_getVel($j$,$k$) )
                }
            } )
        }
    $prevStress$ := new vectorT($nel\times$ NTNS)
    $incStress$ := new vectorT($nel\times$ NTNS)
    $newStress$ := new vectorT($nel\times$ NTNS)
    $prevStrain$ := new vectorT($nel\times$ NTNS)
    $incStrain$ := new vectorT($nel\times$ NTNS)
    $newStrain$ := new vectorT($nel\times$ NTNS)
        $\forall i \in [1..nel]$
        {
            $j := (i-1)\times$ NTNS
            $prevStress$.vec_set($j+1$, fld_getStressElem($i$, 11) )
        $prevStress$.vec_set($j+2$, fld_getStressElem($i$, 22) )
        $prevStress$.vec_set($j+3$, fld_getStressElem($i$, 12) )
        $prevStrain$.vec_set($j+1$, fld_getStrainElem($i$, 11) )
        $prevStrain$.vec_set($j+2$, fld_getStrainElem($i$, 22) )
        $prevStrain$.vec_set($j+3$, fld_getStrainElem($i$, 12) )
    }
```

exception: none

pde_clean():

transition: $nnod := 0$

```
    $nel := 0$
    $nelb := 0$
    $ndof := 0$
    $hbw := 0$
    $mass$.bsm_clean()
    $modMass$.bsm_clean()
    $stiff$.bsm_clean()
    $damp$.bsm_clean()
    $initStress$.vec_clean()
    $initStrain$.vec_clean()
    $body$.vec_clean()
    $trac$.vec_clean()
    $load$.vec_clean()
    $prevDisp$.vec_clean()
    $incDisp$.vec_clean()
    $newDisp$.vec_clean()
    $prevVel$.vec_clean()
    $incVel$.vec_clean()
    $newVel$.vec_clean()
    $prevAcc$.vec_clean()
    $newAcc$.vec_clean()
    $prevStress$.vec_clean()
    $incStress$.vec_clean()
    $newStress$.vec_clean()
    $prevStrain$.vec_clean()
    $incStrain$.vec_clean()
    $newStrain$.vec_clean()
```

exception: none

pde_buildMassMatrix():

```
transition: $nel$ := fld_numElem()

    $\forall i \in [1..nel]$
    {
        $emass$ := bint_emass($i$)
        $ind$ := ind($i$)
        $massMat$.bsm_mappedAdd($emass$,$ind$)
    }

exception: none
```

pde_buildStiffMatrix():

```
transition: $nel$ := fld_numElem()

    $\forall i \in [1..nel]$
    {
        $estiff$ := bint_estiff($i$)
        $ind$ := ind($i$)
        $stiffMat$.bsm_mappedAdd($emass$,$ind$)
    }

exception: none
```

pde_buildDampMatrix(a,b):

```
transition: $factMass$ := $massMat$.bsm_scalMul($a$)
    $factStiff$ := $stiffMat$.bsm_scalMul($b$)
    $dampMat$ := $factMass$.bsm_add($factStiff$)

exception: none
```

pde_buildModMassMatrix():

```
transition: $dt$ := fld_timeStep()
    $factDamp$ := $dampMat$.bsm_scalMul(GAMMA $\times$ $dt$)
    $factStiff$ := $stiffMat$.bsm_scalMul(BETA $\times$ $dt^2$)
    $modMassMat$ := $massMat$.bsm_add($factDamp$)
    $modMassMat$ := $modMassMat$.bsm_add($factStiff$)

exception: none
```

pde_buildLoadVector(t):

```
transition: $nel$ := fld_numElem()

    $nelb$ := bnd_numBoundElem()
    $ndof$ := fld_numDof()
    $dt$ := fld_timeStep()
    ($t = 0 \rightarrow$
        $\forall j \in [1..nel]$
        {
            $ind$ := ind($j$)
            $estress$ := bint_estress($j$)
            $initStressVec$.vec_mappedAdd($estress$,$ind$)
            $estrain$ := bint_estrain($j$)
            $initStrainVec$.vec_mappedAdd($estrain$,$ind$)
        }
        $initStressVec$ := $initStressVec$.vec_scalMul($-1$) )
    $bodyForceVec$ := new vectorT($ndof$)
    $\forall j \in [1..nel]$
    {
        $eacc$ := bint_eacc($j$)
        $ind$ := ind($j$)
        $bodyForceVec$.mappedAdd($eacc$,$ind$)
    }
    $tracVec$ := new vectorT($ndof$)
    $\forall j \in [1..nelb]$
    {
        $etrac$ := tint_etrac($j$)
        $ind$ := ind_t($j$)
        $tracVec$.mappedAdd($etrac$,$ind$)
    }
    $loadVec$ := $tracVec$.vec_add($bodyForceVec$)
    $loadVec$ := $loadVec$.vec_add($initStressVec$)
    $loadVec$ := $loadVec$.vec_add($initStrainVec$)
    ($t = 0 \rightarrow$
        $fieldVec$ := $dampMat$.bsm_vecMul($prevVel$)
        $fieldVec$ := $fieldVec$.vec_scalMul($-1$)
        $loadVec$ := $loadVec$.vec_add($fieldVec$)
        $fieldVec$ := $stiffMat$.bsm_vecMul($prevDisp$)
        $fieldVec$ := $fieldVec$.vec_scalMul($-1$)
        $loadVec$ := $loadVec$.vec_add($fieldVec$)
    | $t \neq 0 \rightarrow$
        $factAcc$ := $prevAcc$.vec_scalMul( $dt \times (1-$ GAMMA$)$ )
        $fieldVec$ := $prevVel$.vec_add($factAcc$)
        $fieldVec$ := $dampMat$.bsm_vecMul($fieldVec$)
        $fieldVec$ := $fieldVec$.vec_scalMul($-1$)
        $loadVec$ := $loadVec$.vec_add($fieldVec$)
        $factAcc$ := $prevAcc$.vec_scalMul( $0.5 \times dt^2 \times (1 - 2\times$ BETA$)$ )
        $factVel$ := $prevVel$.vec_scalMul($dt$)
        $fieldVec$ := $prevDisp$.vec_add($factVel$)
        $fieldVec$ := $fieldVec$.vec_add($factAcc$)
        $fieldVec$ := $stiffMat$.bsm_vecMul($fieldVec$)
        $fieldVec$ := $fieldVec$.vec_scalMul($-1$)
        $loadVec$ := $loadVec$.vec_add($fieldVec$) )
```

exception: none

pde_initAcc():

transition: $prevAcc$:= lin_solve($massMat$, $loadVec$)

```
  $\forall i \in [1..ndof]$
  {
    ($\forall j \in [1..nnod]$
    {
      $\forall k \in [1..$NDIM$]$
      {
        $i$ = fld_getDof($j$,$k$) $\rightarrow$
          fld_setAcc($j$,$k$, $prevDisp$.vec_get($i$) )
      }
    } )
  }
```

exception: none

pde_incAcc():

transition: $newAcc$:= lin_solve($modMassMat$, $loadVec$)

exception: none

pde_incDisp():

```
transition: $dt$ := fld_timeStep()
    $factAcc_1$ := $prevAcc$.vec_scalMul( $1 - 2\times$ BETA )
    $factAcc_2$ := $newAcc$.vec_scalMul( $2\times$ BETA )
    $factVel$ := $prevVel$.vec_scalMul( $dt$ )
    $incDisp$ := $factAcc_1$.vec_add($factAcc_2$)
    $incDisp$ := $incDisp$.vec_scalMul( $0.5 \times dt^2$ )
    $incDisp$ := $incDisp$.vec_add($factVel$)
```

exception: none

pde_incVel():

```
transition: $dt$ := fld_timeStep()
    $factAcc_1$ := $prevAcc$.vec_scalMul( $1-$ GAMMA )
    $factAcc_2$ := $newAcc$.vec_scalMul( GAMMA )
    $incVel$ := $factAcc_1$.vec_add($factAcc_2$)
    $incVel$ := $incVel$.vec_scalMul( $dt$ )
```

exception: none

pde_incStrain():

```
transition: $nel$ := fld_numElem()

    $incStrain$ := new vectorT( $nel\times$ NTNS )
    $\forall i \in [1..nel]$
    {
        $dDisp$ := new vectorT( NDIM $\times$ NNODEL )
        $\forall j \in$ [1..NNODEL]
        {
            $\forall k \in$ [1..NDIM]]
            {
                $l$ := fld_getDof( fld_getConnect($i$,$j$), $k$ )
                ( $l \neq 0 \rightarrow$
                    $dDisp$.vec_set( $(j-1)\times$ NDIM $+k$, $incDisp$.vec_get($l$) )
            }
        }
        $\boldsymbol{B}$ := bmatrix($i$)
        $dStrain$ := $\boldsymbol{B}$.dm_vecMul($dDisp$)
        $ind$ := [$(i-1)\times$ NTNS $+1..i\times$ NTNS]
        $incStrain$.vec_mappedAdd($dStrain$,$ind$)
    }
```

exception: none

pde_incStress():

transition: nel := fld_numElem()

```
    $incStress$ := new vectorT( $nel \times$ NTNS )
    $\forall i \in [1..nel]$
    {
        $dStrain$ := new vectorT( NTNS )
        $\forall j \in [1..\text{NTNS}]$
        {
            $dStrain$.vec_set( $j$, $incStrain$.vec_get($(i-1)\times$ NTNS $+j$) )
        }
        $m$ := fld_getMaterial($i$)
        $E$ := mtl_getEmod($m$)
        $\nu$ := mtl_getPois($m$)
        $dStress$ := linearElastic($E$,$\nu$,$dStrain$)
        $ind$ := $[(i-1)\times$ NTNS $+1..i\times$ NTNS$]$
        $incStress$.vec_mappedAdd($dStress$,$ind$)
    }
```

exception: none

pde_updateAcc():

transition: $ndof$:= fld_numDof()

```
    $nnod$ := fld_numNode()
    $\forall i \in [1..ndof]$
    {
        ($\forall j \in [1..nnod]$
        {
            $\forall k \in [1..\text{NDIM}]$
            {
                $i$ = fld_getDof($j$,$k$) $\rightarrow$
                    fld_setAcc( $j$, $k$, $newAcc$.vec_get($i$) )
            }
        } )
    }
    $prevAcc$ := $newAcc$
```

exception: none

pde_updateDisp():

transition: $newDisp$:= $prevDisp$.vec_add($incDisp$)

```
    $ndof$ := fld_numDof()
    $nnod$ := fld_numNode()
    $\forall i \in [1..ndof]$
    {
        ($\forall j \in [1..nnod]$
        {
            $\forall k \in [1..\mathrm{NDIM}]$
            {
                $i$ = fld_getDof($j$,$k$) $\rightarrow$
                    fld_setDisp( $j$, $k$, $newDisp$.vec_get($i$) )
            }
        } )
    }
    $prevDisp$ := $newDisp$
```

exception: none

pde_updateVel():

transition: $newVel$:= $prevVel$.vec_add($incVel$)

```
    $ndof$ := fld_numDof()
    $nnod$ := fld_numNode()
    $\forall i \in [1..ndof]$
    {
        ($\forall j \in [1..nnod]$
        {
            $\forall k \in [1..\mathrm{NDIM}]$
            {
                $i$ = fld_getDof($j$,$k$) $\rightarrow$
                    fld_setVel( $j$, $k$, $newVel$.vec_get($i$) )
            }
        } )
    }
    $prevVel$ := $newVel$
```

exception: none

pde_updateStrain():

transition: *newStrain* := *prevStrain*.vec_add(*incStrain*)

```
    $nel$ := fld_numElem()
    $\forall i \in [1..nel]$
    {
        fld_setStrainElem( $i$, 11, $newStrain$.vec_get( $(i-1)\times$ NTNS +1 ) )
        fld_setStrainElem( $i$, 22, $newStrain$.vec_get( $(i-1)\times$ NTNS +2 ) )
        fld_setStrainElem( $i$, 12, $newStrain$.vec_get( $(i-1)\times$ NTNS +3 ) )
    }
    $prevStrain$ := $newStrain$
```

exception: none

pde_updateStress():

transition: *newStress* := *prevStress*.vec_add(*incStress*)

```
    $nel$ := fld_numElem()
    $\forall i \in [1..nel]$
    {
        fld_setStressElem( $i$, 11, $newStress$.vec_get( $(i-1)\times$ NTNS +1 ) )
        fld_setStressElem( $i$, 22, $newStress$.vec_get( $(i-1)\times$ NTNS +2 ) )
        fld_setStressElem( $i$, 12, $newStress$.vec_get( $(i-1)\times$ NTNS +3 ) )
    }
    $prevStress$ := $newStress$
```

exception: none

Local Functions

max: real $\times$ real $\rightarrow$ real
max(a,b) $\equiv$ ($a \geq b \rightarrow a \mid a < b \rightarrow b$)

hbw: () $\rightarrow$ integer
hbw():

transition: *result* := 0

```
$\forall i \in$ [1..fld_numElem]
{
  $\forall j \in$ [1..NNODEL]
  {
    $\forall k \in$ [1..NDIM]
    {
      $\forall l \in$ [($k <$ NDIM $\rightarrow j \mid k =$ NDIM $\rightarrow j + 1$)..NNODEL]
      {
        $\forall m \in$ [($k <$ NDIM $\rightarrow k + 1 \mid k =$ NDIM $\rightarrow 1$)..NNODEL]
        {
          $result$ :=
            ( fld_getDof(fld_getConnect($i$,$j$),$k$) $= 0 \rightarrow result$
            | fld_getDof(fld_getConnect($i$,$l$),$m$) $= 0 \rightarrow result$
            | fld_getDof(fld_getConnect($i$,$j$),$k$) $\neq 0$
              $\wedge$ fld_getDof(fld_getConnect($i$,$l$),$m$) $\neq 0$
              $\rightarrow$ max($result$,|fld_getDof(fld_getConnect($i$,$j$),$k$)
                $-$ fld_getDof(fld_getConnect($i$,$l$),$m$)| )
        }
      }
    }
  }
}
```

output: *out* := *result*

exception: none

ind: integer → sequence of integer
ind(i):

transition: *result* := sequence [NDIM × NNODEL] of integer

```
    $\forall j \in$ [1..NNODEL]
    {
        $\forall k \in$ [1..NDIM]
        {
            $result[(j-1)\times$ NDIM $+k]$ := fld_getDof( fld_getConnect($i$,$j$), $k$ )
        }
    }
```

output:

exception: none

ind_t: integer → sequence of integer
ind_t(i):

transition: *result* := sequence [NDIM × NNODELB] of integer

```
    $\forall j \in$ [1..NNODELB]
    {
        $\forall k \in$ [1..NDIM]
        {
            $result[(j-1)\times$ NDIM $+k]$ := fld_getDof( bnd_getConnect($i$,$j$), $k$ )
        }
    }
```

output:

exception: none

Local Types
N/A

Local Constants
N/A

Considerations
N/A

4.3.4 PDE Solver Constants

Uses

System Constants

Syntax

Exported Constants

Table 4.2: Exported constants for PDE Solver Constants module

Name	Type	Value
NGAUSS_ELEM	integer	1
NGAUSS_BOUND	integer	1
GAUSS_PT_ELEM	sequence of real	$[\frac{1}{3}, \frac{1}{3}]$
GAUSS_WT_ELEM	real	1.0
GAUSS_PT_BOUND	real	0.5
GAUSS_WT_BOUND	real	1.0
BETA	real	0.25
GAMA	real	0.5

Exported Types

N/A

Exported Functions

N/A

Semantics

State Variables
N/A

State Invariants
N/A

Assumptions
N/A

Access Routine Semantics
N/A

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

4.3.5.1 Dense Matrix ADT

Uses

Modules:
Floating Point Operations
Integer Operations
Log Message Control
Log Messages
Memory Access
Vector ADT

Syntax

Exported Constants
N/A

Exported Types
matrixT = ?

Exported Functions

Table 4.3: Exported function interfaces for Dense Matrix ADT module

Name	Input	Output	Exceptions
new matrixT	integer, integer	matrixT	ALLOC, SZE
dm_clean			
dm_numRows		integer	
dm_numCols		integer	
dm_get	integer, integer	real	POSIT
dm_set	integer, integer, real		POSIT
dm_add	matrixT	matrixT	DIMEN
dm_scalMul	real	matrixT	
dm_vecMul	vectorT	vectorT	DIMEN
dm_matMul	matrixT	matrixT	DIMEN
dm_transpose		matrixT	

Semantics

State Variables

dat : sequence of sequence of real

State Invariants
N/A

Assumptions
N/A

Access Routine Semantics

new matrixT(m,n):

transition: *dat* := sequence [m,n] of real s.t.

```
$\forall i \in [1..m]$
{
    $\forall j \in [1..n]$
    {
        $dat[i, j] := 0.0$
    }
}
```

output: *out* := *self*

exception: *exc* := (Amount of memory required for *dat* < mem_getAvailMem()
$\rightarrow$ ALLOC
| $m \leq 0 \rightarrow$ SZE
| $n \leq 0 \rightarrow$ SZE)

dm_clean():

transition: Deallocate memory for *dat*

exception: none

dm_numRows():

output: *out* := $|dat|$

exception: none

dm_numCols():

output: $out := |dat[1]|$

exception: none

dm_get(i,j):

output: $out := dat[i, j]$

exception: exc := ($i \notin$ [1..*self*.dm_numRows()] $\rightarrow$ POSIT
| $j \notin$ [1..*self*.dm_numCols()] $\rightarrow$ POSIT)

dm_set(i,j,v):

transition: $dat[i, j] := v$

exception: exc := ($i \notin$ [1..*self*.dm_numRows()] $\rightarrow$ POSIT
| $j \notin$ [1..*self*.dm_numCols()] $\rightarrow$ POSIT)

dm_add(*other*):

transition: *result* := new matrixT(*self*.dm_numRows() , *self*.dm_numCols())

```
$\forall i \in$ [1..result.dm_numRows()]
{
    $\forall j \in$ [1..result.dm_numCols()]
    {
        result.dm_set(i,j, self.dm_get(i,j) + other.dm_get(i,j) )
    }
}
```

output: out := *result*

exception: exc := (*self*.dm_numRows() $\neq$ *other*.dm_numRows() $\rightarrow$ DIMEN
| *self*.dm_numCols() $\neq$ *other*.dm_numCols() $\rightarrow$ DIMEN)

dm_scalMul(k):

transition: *result* := new matrixT(*self*.dm_numRows() , *self*.dm_numCols())

```
$\forall i \in$ [1..result.dm_numRows()]
{
    $\forall j \in$ [1..result.dm_numCols()]
    {
        result.dm_set( i , j , $k \times$ self.dm_get(i,j) )
    }
}
```

output: out := *result*

exception: none

dm_vecMul(*other*):

transition: *result* := new vectorT(*self*.dm_numRows())

```
    $\forall i \in$ [1..*result*.vec_length()]
    {
        $\forall j \in$ [1..*self*.dm_numCols()]
        {
            *result*.vec_set($i$, *result*.vec_get($i$) + *self*.dm_get($i$,$j$) $\times$ *other*.vec_get($j$) )
        }
    }
```

output: *out* := *result*

exception: *exc* := (*self*.dm_numCols() $\neq$ *other*.vec_length() $\rightarrow$ DIMEN)

dm_matMul(*other*):

transition: *result* := new matrixT(*self*.dm_numRows() , *other*.dm_numCols())

```
    $\forall i \in$ [1..*result*.dm_numRows()]
    {
        $\forall j \in$ [1..*result*.dm_numCols()]
        {
            $\forall k \in$ [1..*self*.dm_numCols()]
            {
                *result*.dm_set($i$,$j$, *result*.dm_get($i$,$j$) + *self*.dm_get($i$,$k$) $\times$ *other*.dm_get($k$,$j$) )
            }
        }
    }
```

output: *out* := *result*

exception: *exc* := (*self*.dm_numCols() $\neq$ *other*.dm_numRows() $\rightarrow$ DIMEN) dm_transpose():

transition: *result* := new matrixT(*self*.dm_numRows() , *self*.dm_numCols())

```
    $\forall i \in [1..result$.dm_numRows()]
    {
        $\forall j \in [1..result$.dm_numCols()]
        {
            $result$.dm_set( $i$,$j$, $self$.dm_get($j$,$i$) )
        }
    }
```

output: *out* := *result*

exception: none

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

4.3.5.2 Banded Symmetric Matrix ADT

Uses

Modules:
Dense Matrix ADT
Floating Point Operations
Integer Operations
Log Message Control
Log Messages
Memory Access
Vector ADT

Syntax

Exported Constants
N/A

Exported Types
bandSymMatrixT = ?

Exported Functions

Table 4.4: Exported function interfaces for Banded Symmetric Matrix ADT module

Name	**Input**	**Output**	**Exceptions**
new bandSymMatrixT	integer, integer	bandSymMatrixT	ALLOC, SZE
bsm_clean			
bsm_numRows		integer	
bsm_halfBW		integer	
bsm_get	integer, integer	real	POSIT
bsm_set	integer, integer, real		POSIT
bsm_setDecomp	seq of seq of real		DIMEN
bsm_add	bandSymMatrixT	bandSymMatrixT	DIMEN
bsm_mappedAdd	matrixT, seq of integer		DIMEN, POSIT
bsm_scalMul	real	bandSymMatrixT	
bsm_vecMul	vectorT	vectorT	DIMEN

Semantics

State Variables

dat : sequence of sequence of real
decomp : sequence of sequence of real
is_decomposed : boolean

State Invariants
N/A

Assumptions

1. The matrixT object passed to bsm_mappedAdd() is symmetric.
2. There are no repeated indices in the sequence of integers passed to bsm_mappedAdd().

Access Routine Semantics

new bandSymMatrixT(*hbw*,*n*):

transition: *dat* := sequence [*hbw*,*n*] of real

```
    $decomp$ := sequence [$hbw$,$n$] of real
    $is\_decomposed$ := FALSE
    $\forall i \in [1..hbw]$
    {
        $\forall j \in [1..n]$
        {
            $dat[i,j] := 0.0$
            $decomp[i,j] := 0.0$
        }
    }
```

output: *out* := *self*

exception: *exc* := (Amount of memory required for *dat* < mem_getAvailMem()
→ ALLOC
| Amount of memory required for *decomp* < mem_getAvailMem() → ALLOC
| $hbw \leq 0 \rightarrow$ SZE
| $n \leq 0 \rightarrow$ SZE)

bsm_clean():

transition: Deallocate memory for *dat* and *decomp*
is_decomposed := FALSE

exception: none

bsm_numRows():

output: $\mathit{out} := |\mathit{dat}[1]|$

exception: none

bsm_halfBW():

output: $\mathit{out} := |\mathit{dat}|$

exception: none

bsm_get(i,j):

transition: $r, c :=$ *self*.packed_index(i,j)

output: $\mathit{out} := (c - r < \mathit{self}.\text{bsm\_halfBW}() \rightarrow \mathit{dat}[r, c] \mid c - r \geq \mathit{self}.\text{bsm\_halfBW}() \rightarrow 0.0)$

exception: $\mathit{exc} := (i \notin [1..\mathit{self}.\text{bsm\_numRows}()] \rightarrow \text{POSIT}$
$|j \notin [1..\mathit{self}.\text{bsm\_numRows}()] \rightarrow \text{POSIT})$

bsm_set(i,j,v):

transition: $r, c :=$ *self*.packed_index(i,j)
$\mathit{dat}[r, c] := v$

exception: $\mathit{exc} := (i \notin [1..\mathit{self}.\text{bsm\_numRows}()] \rightarrow \text{POSIT}$
$|j \notin [1..\mathit{self}.\text{bsm\_numRows}()] \rightarrow \text{POSIT}$
$|\ |j - i| \geq \mathit{self}.\text{bsm\_halfBW}() \rightarrow \text{POSIT})$

bsm_setDecomp(d):

transition: $\mathit{decomp} := d$
$\mathit{is\_decomposed} := \text{TRUE}$

exception: $\mathit{exc} := (|d| \neq \mathit{self}.\text{halfBW}() \rightarrow \text{DIMEN}$
$|\ |d(1)| \neq \mathit{self}.\text{numRows}() \rightarrow \text{DIMEN}\ )$

bsm_add(*other*):

```
transition: $result$ := new bandSymMatrixT( $self$.bsm_numRows() ,
        max($self$.bsm_halfBW(),$other$.bsm_halfBW() ) )

        $\forall i \in [1..result$.bsm_numRows()]
        {
            $\forall j \in [i..\min(i + result$.bsm_halfBW()$-1$, $result$.bsm_numRows())]
            {
                $result$.bsm_set($i$,$j$, $self$.bsm_get($i$,$j$) + $other$.bsm_get($i$,$j$) )
            }
        }
```

output: *out* := *result*

exception: *exc* := (*self*.bsm_numRows() $\neq$ *other*.bsm_numRows() $\rightarrow$ DIMEN)

bsm_mappedAdd(*other*,*ind*):

```
transition: $\forall i \in [1..|ind|]$

        {
            $\forall j \in [i..|ind|]$
            {
                $(ind[i] \neq 0 \wedge ind[j] \neq 0$
                    $\rightarrow$ $self$.bsm_set($ind[i]$,$ind[j]$,
                        $self$.bsm_get($ind[i]$,$ind[j]$)
                        + $other$.dm_get($i$,$j$) ) )
            }
        }
```

exception: *exc* := (*other*.dm_numRows() $\neq$ *other*.dm_numCols() $\rightarrow$ DIMEN
| *other*.dm_numRows() $\neq |ind| \rightarrow$ DIMEN
| *other*.dm_numRows() $>$ *self*.bsm_halfBW() $\rightarrow$ DIMEN
| $\exists i \in ind$ s.t. $i \notin [0..self$.bsm_numRows()] $\rightarrow$ POSIT
| $\exists i, j \in ind$ s.t. $i \neq 0 \wedge j \neq 0 \wedge |j - i| \geq self$.bsm_halfBW() $\rightarrow$ POSIT)

bsm_scalMul(k):

transition: *result* := new bandSymMatrixT(*self*.bsm_numRows() , *self*.bsm_halfBW())

```
$\forall i \in [1..result$.bsm_numRows()$]$
{
    $\forall j \in [i..\min(i + result$.bsm_halfBW()$-1, result$.bsm_numRows()$)]$
    {
        $result$.bsm_set( $i$,$j$,$k \times self$.bsm_get($i$,$j$) )
    }
}
```

output: *out* := *result*

exception: none

bsm_vecMul(*other*):

transition: *result* := new vectorT(*self*.bsm_numRows())

```
$\forall i \in [1..result$.vec_length()$]$
{
    $\forall j \in [\max(i - self$.bsm_halfBW()$+1, 1)..\min(i + self$.bsm_halfBW()$-1, self$.bsm_numRows()$)]$
    {
        $result$.vec_set( $i$, $result$.vec_get($i$) + $self$.bsm_get($i$,$j$) $\times$ $other$.vec_get($j$) )
    }
}
```

output: *out* := *result*

exception: *exc* := (*self*.bsm_numRows() $\neq$ *other*.vec_length() $\rightarrow$ DIMEN)

Local Functions

min: real × real → real
min(a,b) ≡ ($a \leq b \rightarrow a \mid a > b \rightarrow b$)

max: real × real → real
max(a,b) ≡ ($a \geq b \rightarrow a \mid a < b \rightarrow b$)

packed_index: integer × integer → integer × integer

packed_index(i,j):

transition: r,c := ($i > j \rightarrow j$,$i \mid i \leq j \rightarrow i$,$j$)
r := *self*.bsm_halfBW() − ($c - r$)

output: *out* := r,c

exception: none

Local Types
N/A

Local Constants
N/A

Considerations
N/A

4.3.5.3 Vector ADT

Uses

Modules:
Floating Point Operations
Integer Operations
Log Message Control
Log Messages
Memory Access

Syntax

Exported Constants
N/A

Exported Types
vectorT = ?

Exported Functions

Table 4.5: Exported function interfaces for Vector ADT module

Name	Input	Output	Exceptions
new vectorT	integer	vectorT	ALLOC, SZE
vec_clean			
vec_length		integer	
vec_get	integer	real	POSIT
vec_set	integer, real		POSIT
vec_add	vectorT	vectorT	DIMEN
vec_mappedAdd	vectorT, seq of integer		DIMEN, POSIT
vec_scalMul	real	vectorT	
vec_dotProd	vectorT	real	DIMEN

Semantics

State Variables

dat : sequence of real

State Invariants
N/A

Assumptions

1. There are no repeated indices in the sequence of integers passed to vec_mappedAdd().

Access Routine Semantics

new vectorT(n):

transition: $dat :=$ sequence $[n]$ of real

```
$\forall i \in [1..n]$
{
    $dat[i] := 0.0$
}
```

output: $out := self$

exception: $exc :=$ (Amount of memory required for $dat <$ mem_getAvailMem()
  $\rightarrow$ ALLOC
  $| \; n \leq 0 \rightarrow$ SZE)

vec_clean():

transition: Deallocate memory for *dat*

exception: none

vec_length():

output: $out := |dat|$

exception: none

vec_get(i):

output: $out := dat[i]$

exception: $exc := (i \notin [1..self.\text{vec\_length}()] \rightarrow$ POSIT)

vec_set(i,v):

transition: $dat[i] := v$

exception: $exc := (i \notin [1..self.\text{vec\_length}()] \rightarrow$ POSIT) vec_add($other$):

```
transition: $result$ := new vectorT( $self$.vec_length() )

        $\forall i \in [1..result.\text{vec\_length}()]$
        {
            $result$.vec_set($i$, $self$.vec_get($i$) + $other$.vec_get($i$) )
        }

output: $out$ := $result$

exception: $exc$ := ($self$.vec_length() $\neq$ $other$.vec_length() $\rightarrow$ DIMEN)
```

vec_mappedAdd($other$,ind):

```
transition: $\forall i \in [1..|ind|]$

        {
            ($ind[i] \neq 0$
                $\rightarrow$ $self$.vec_set($ind[i]$,
                    $self$.vec_get($ind[i]$)
                    + $other$.vec_get($i$) ) )
        }

exception: $exc$ := ($other$.vec_length() $\neq |ind| \rightarrow$ DIMEN
        | $\exists i \in ind$ s.t. $i \notin [0..self.\text{vec\_length}()] \rightarrow$ POSIT)
```

vec_scalMul(k):

```
transition: $result$ := new vectorT( $self$.vec_length() )

        $\forall i \in [1..result.\text{vec\_length}()]$
        {
            $result$.vec_set( $i$, $k \times self$.vec_get($i$) )
        }

output: $out$ := $result$

exception: none
```

vec_dotProd(*other*):

transition: $result := 0.0$

```
$\forall i \in [1..self.\text{vec\_length}()]$
{
    $result := result + self.\text{vec\_get}(i) \times other.\text{vec\_get}(i)$
}
```

output: $out := result$

exception: $exc := (self.\text{vec\_length}() \neq other.\text{vec\_length}() \rightarrow \text{DIMEN})$

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

4.3.6.1 Body Element Integration

Uses

Modules:
Constitutive Matrix
Body Element Interpolation
Dense Matrix ADT
Field Data
Floating Point Operations
Integer Operations
Kinematic Matrix
Material Property Data
PDE Solver Constants
Vector ADT

Syntax

Exported Constants
N/A

Exported Types
N/A

Exported Functions

Table 4.6: Exported function interfaces for Body Element Integration module

Name	Input	Output	Exceptions
bint_emass	integer	matrixT	
bint_estiff	integer	matrixT	
bint_eacc	integer	vectorT	
bint_estress	integer	vectorT	
bint_estrain	integer	vectorT	

Semantics

State Variables
N/A

State Invariants
N/A

Assumptions
N/A

Access Routine Semantics

bint_emass(i):

transition: m := fld_getMaterial(i)
ρ_e := mtl_getDens(m)

output: $out := \int_{A_e} \boldsymbol{N_e}^T \rho_e \boldsymbol{N_e} dA_e$
for body element i where
$\boldsymbol{N_e}$:= bshp_shape(l_1,l_2) and
l_1,l_2 are area coordinates that vary from 0 to 1 over A_e as defined in Section 2.3.2

exception: none

bint_estiff(i):

transition: $\boldsymbol{B_e}$:= bmatrix(i)
m := fld_getMaterial(i)
E := mtl_getEmod(m)
ν := mtl_getPois(m)
$\boldsymbol{D_e}$:= dmatrix(E,ν)

output: $out := \int_{A_e} \boldsymbol{B_e}^T \boldsymbol{D_e} \boldsymbol{B_e} dA_e$ for body element i

exception: none

bint_eacc(i):

transition: m := fld_getMaterial(i)

```
$\rho_e$ := mtl_getDens($m$)
$\boldsymbol{f_e}$ := new vectorT(NDIM $\times$ NNODEL)
$\forall j \in$ [1..NNODEL]
{
    $\forall k \in$ [1..NDIM]
    {
        $\boldsymbol{f_e}$.vec_set( $(j-1)\times$ NDIM $+k$, fld_getBodyAcc( fld_getConnect($i$,$j$), $k$) )
    }
}
```

output: $out := \int_{A_e} \boldsymbol{N_e}^T \rho_e \boldsymbol{f_e} dA_e$
for body element i where
$\boldsymbol{N_e}$:= bshp_shape(l_1,l_2) and
l_1,l_2 are area coordinates that vary from 0 to 1 over A_e as defined in Section 2.3.2

exception: none

bint_estress(i):

transition: $\boldsymbol{B_e}$:= bmatrix(i)
$\boldsymbol{\sigma_{0e}}$:= new vectorT(NTNS)
$\boldsymbol{\sigma_{0e}}$.vec_set(1, fld_getStressElem(i,11))
$\boldsymbol{\sigma_{0e}}$.vec_set(2, fld_getStressElem(i,22))
$\boldsymbol{\sigma_{0e}}$.vec_set(3, fld_getStressElem(i,12))

output: $out := \int_{A_e} \boldsymbol{B_e}^T \boldsymbol{\sigma_{0e}} dA_e$

exception: none

bint_estrain(i):

transition: $\boldsymbol{B_e}$:= bmatrix(i)
m := fld_getMaterial(i)
E := mtl_getEmod(m)
ν := mtl_getPois(m)
$\boldsymbol{D_e}$:= dmatrix(E,ν)
$\boldsymbol{\varepsilon_{0e}}$:= new vectorT(NTNS)
$\boldsymbol{\varepsilon_{0e}}$.vec_set(1, fld_getStrainElem(i,11))
$\boldsymbol{\varepsilon_{0e}}$.vec_set(2, fld_getStrainElem(i,22))
$\boldsymbol{\varepsilon_{0e}}$.vec_set(3, fld_getStrainElem(i,12))

output: $out := \int_{A_e} \boldsymbol{B_e}^T \boldsymbol{D_e} \boldsymbol{\varepsilon_{0e}} dA_e$

exception: none

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

4.3.6.2 Traction Element Integration

Uses

Modules:
Boundary Data
Dense Matrix ADT
Floating Point Operations
Integer Operations
PDE Solver Constants
Traction Element Interpolation
Vector ADT

Syntax

Exported Constants
N/A

Exported Types
N/A

Exported Functions

Table 4.7: Exported function interfaces for Traction Element Integration module

Name	Input	Output	Exceptions
tint_etrac	integer	matrixT	

Semantics

State Variables
N/A

State Invariants
N/A

Assumptions
N/A

Access Routine Semantics

tint_etrac(i):

transition: l_t := bnd_lenBoundElem(i)
 $\boldsymbol{T}$:= tshp_transform(i)
 t_1 := bnd_getTrac(i,1)
 t_2 := bnd_getTrac(i,2)
 $\bar{\boldsymbol{t}}'_{\boldsymbol{t}}$:= new vectorT(NNODELB $\times$ NDIM)
 $\bar{\boldsymbol{t}}'_{\boldsymbol{t}}$.vec_set(1,$t_1.\sigma_{nt}$)
 $\bar{\boldsymbol{t}}'_{\boldsymbol{t}}$.vec_set(2,$t_1.\sigma_{nn}$)
 $\bar{\boldsymbol{t}}'_{\boldsymbol{t}}$.vec_set(3,$t_2.\sigma_{nt}$)
 $\bar{\boldsymbol{t}}'_{\boldsymbol{t}}$.vec_set(4,$t_2.\sigma_{nn}$)

output: $out := \int_0^1 \boldsymbol{N}_{\boldsymbol{t}}^T \boldsymbol{T}^T \boldsymbol{N}_{\boldsymbol{t}} \bar{\boldsymbol{t}}'_{\boldsymbol{t}} l_t ds$ for traction element i where
 $\boldsymbol{N}_{\boldsymbol{t}}$:= tshp_shape(s)

exception: none

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

4.3.7.1 Body Element Interpolation

Uses

Modules:
Dense Matrix ADT
Floating Point Operations
Integer Operations
System Constants

Syntax

Exported Constants
N/A

Exported Types
N/A

Exported Functions

Table 4.8: Exported function interfaces for Body Element Interpolation module

Name	Input	Output	Exceptions
bshp_shape	real, real	matrixT	

Semantics

State Variables
N/A

State Invariants
N/A

Assumptions

1. The sum of the inputs to bshp_shape(l_1,l_2) is less than or equal to unity. That is, $l_1 + l_2 \leq 1$.

Access Routine Semantics

bshp_shape(l_1,l_2):

transition: *result* := new matrixT(NDIM, NDIM $\times$ NNODEL)

```
    result.dm_set(1,1, $l_1$)
    result.dm_set(2,2, $l_1$)
    result.dm_set(1,3, $l_2$)
    result.dm_set(2,4, $l_2$)
    result.dm_set(1,5, $1 - l_1 - l_2$)
    result.dm_set(2,6, $1 - l_1 - l_2$)
```

output: *out* := *result*

exception: none

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

4.3.7.2 Traction Element Interpolation

Uses

Modules:
Boundary Data
Dense Matrix ADT
Field Data
Floating Point Operations
Integer Operations
System Constants

Syntax

Exported Constants
N/A

Exported Types
N/A

Exported Functions

Table 4.9: Exported function interfaces for Traction Element Interpolation module

Name	Input	Output	Exceptions
tshp_shape	real	matrixT	
tshp_transform	integer	matrixT	

Semantics

State Variables
N/A

State Invariants
N/A

Assumptions
N/A

Access Routine Semantics

tshp_shape(s):

transition: *result* := new matrixT(NDIM, NDIM $\times$ NNODELB)

    result.dm_set(1,1, $1 - s$)
    result.dm_set(2,2, $1 - s$)
    result.dm_set(1,3, s)
    result.dm_set(2,4, s)

output: *out* := *result*

exception: none

tshp_transform(i):

transition: x_1 := fld_getCoord(bnd_getConnect(i,1),1)

    y_1 := fld_getCoord(bnd_getConnect(i,1),2)
    x_2 := fld_getCoord(bnd_getConnect(i,2),1)
    y_2 := fld_getCoord(bnd_getConnect(i,2),2)
    $\theta := \tan^{-1}\left(\frac{y_2-y_1}{x_2-x_1}\right)$
    result := new matrixT(NDIM, NDIM)
    result.dm_set(1,1, $\cos\theta$)
    result.dm_set(2,1, $-\sin\theta$)
    result.dm_set(1,2, $\sin\theta$)
    result.dm_set(2,2, $\cos\theta$)

output: *out* := *result*

exception: none

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

4.3.8.1 Linear Elastic Model

Uses

Modules:
Constitutive Matrix
Dense Matrix ADT
Floating Point Operations
Integer Operations
Vector ADT

Syntax

Exported Constants
N/A

Exported Types
N/A

Exported Functions

Table 4.10: Exported function interfaces for Linear Elastic Model module

Name	**Input**	**Output**	**Exceptions**
linearElastic	real, real, vectorT	vectorT	

Semantics

State Variables
N/A

State Invariants
N/A

Assumptions
N/A

Access Routine Semantics

linearElastic(E,ν,$d\varepsilon$):

transition: D := dmatrix(E,ν)

output: out := D.dm_vecMul($d\varepsilon$)

exception: none

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

4.3.8.2 Constitutive Matrix

Uses

Modules:
Dense Matrix ADT
Floating Point Operations
Integer Operations
Log Message Control
Log Messages
System Constants

Syntax

Exported Constants
N/A

Exported Types
N/A

Exported Functions

Table 4.11: Exported function interfaces for Constitutive Matrix module

Name	Input	Output	Exceptions
dmatrix	real, real	matrixT	EXCEED

Semantics

State Variables
N/A

State Invariants
N/A

Assumptions
N/A

Access Routine Semantics

dmatrix(E,ν):

transition: *result* := new matrixT(NTNS,NTNS)

```
    $result$.dm_set(1,1, $1-\nu$)
    $result$.dm_set(2,1, $\nu$)
    $result$.dm_set(1,2, $\nu$)
    $result$.dm_set(2,2, $1-\nu$)
    $result$.dm_set(3,3, $1-2\nu$)
    $result$.dm_scalMul($\frac{E}{(1+\nu)(1-2\nu)}$)
```

output: *out* := *result*

exception: *exc* := ($E <$ E_MIN $\rightarrow$ EXCEED

```
    | $E >$ E_MAX $\rightarrow$ EXCEED
    | $\nu <$ NU_MIN $\rightarrow$ EXCEED
    | $\nu >$ NU_MAX $\rightarrow$ EXCEED)
```

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

4.3.8.3 Kinematic Matrix

Uses

Modules:
Dense Matrix ADT
Field Data
Floating Point Operations
Integer Operations
System Constants

Syntax

Exported Constants
N/A

Exported Types
N/A

Exported Functions

Table 4.12: Exported function interfaces for Kinematic Matrix module

Name	Input	Output	Exceptions
bmatrix	integer	matrixT	

Semantics

State Variables
N/A

State Invariants
N/A

Assumptions
N/A

Access Routine Semantics

bmatrix(i):

transition: *result* := new matrixT(NTNS , NDIM $\times$ NNODEL)

```
    $b_1$ := fld_getCoord(fld_getConnect($i$,2),2) $-$ fld_getCoord(fld_getConnect($i$,3),2)
    $b_2$ := fld_getCoord(fld_getConnect($i$,3),2) $-$ fld_getCoord(fld_getConnect($i$,1),2)
    $b_3$ := fld_getCoord(fld_getConnect($i$,1),2) $-$ fld_getCoord(fld_getConnect($i$,2),2)
    $c_1$ := fld_getCoord(fld_getConnect($i$,3),1) $-$ fld_getCoord(fld_getConnect($i$,2),1)
    $c_2$ := fld_getCoord(fld_getConnect($i$,1),1) $-$ fld_getCoord(fld_getConnect($i$,3),1)
    $c_3$ := fld_getCoord(fld_getConnect($i$,2),1) $-$ fld_getCoord(fld_getConnect($i$,1),1)
    result.dm_set(1,1, $b_1$)
    result.dm_set(2,2, $c_1$)
    result.dm_set(1,3, $b_2$)
    result.dm_set(2,4, $c_2$)
    result.dm_set(1,5, $b_3$)
    result.dm_set(2,6, $c_3$)
    result.dm_set(3,1, $c_1$)
    result.dm_set(3,2, $b_1$)
    result.dm_set(3,3, $c_2$)
    result.dm_set(3,4, $b_2$)
    result.dm_set(3,5, $c_3$)
    result.dm_set(3,6, $b_3$)
    result.dm_scalMul($\frac{1}{2\times \text{fld\_volElem}(i)}$)
```

output: *out* := *result*

exception: none

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

4.3.9.1 Linear Solver

Uses

Modules:
Banded Symmetric Matrix ADT
Floating Point Operations
Integer Operations
Log Message Control
Log Messages
Vector ADT

Syntax

Exported Constants
N/A

Exported Types
N/A

Exported Functions

Table 4.13: Exported function interfaces for Linear Solver module

Name	Input	Output	Exceptions
lin_solve	bandSymMatrixT, vectorT	vectorT	DIMEN

Semantics

State Variables
N/A

State Invariants
N/A

Assumptions
N/A

Access Routine Semantics

lin_solve(A,b):

output: $out := x$ s.t. (x is a vectorT
$\wedge\ x$.length() $= b$.length()
$\wedge\ \frac{||A^{-1}x-b||}{||b||} < \varepsilon_a$)

exception: $exc :=$ (A.numRows() $\neq b$.length() $\rightarrow$ DIMEN)

Local Functions
N/A

Local Types
N/A

Local Constants
$\varepsilon_a := 1 \times 10^{-5}$

Considerations
N/A

References

[1] B. Karchewski, "Module guide for two and three dimensional dynamic model of soil-water-structure interaction," Course Project - CES 741, McMaster University, Mar. 2012.

[2] ——, "Module interface specification for two and three dimensional dynamic model of soil-water-structure interaction," Course Project - CES 741, McMaster University, Mar. 2012.

[3] ——, "Software requirements specification for two and three dimensional dynamic model of soil-water-structure interaction," Course Project - CES 741, McMaster University, Feb. 2012.

[4] D. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, vol. 15, no. 12, Dec.

[5] A. Chopra, *Dynamics of Structures.* Englewood Cliffs, NJ: Prentice-Hall, Inc., 1995.

[6] K.-J. Bathe, *Finite Element Procedures.* Upper Saddle River, NJ: Prentice-Hall, Inc., 1996.

[7] O. Zienkiewicz and R. Taylor, *The Finite Element Method*, 5th ed. Boston, MA: Butterworth-Heinemann, 2000.

[8] N. Newmark, "A method of computation for structural dynamics," *Journal of Engineering Mechanics, ASCE*, vol. 85, no. EM3, pp. 67–94, 1959.

[9] M. Dokainish and K. Subbaraj, "A survey of direct time-integration methods in computational structural dynamics—I. Explicit methods," *Computers and Structures*, vol. 32, no. 6, pp. 1371–1386, 1989.

[10] ——, "A survey of direct time-integration methods in computational structural dynamics—II. Implicit methods," *Computers and Structures*, vol. 32, no. 6, pp. 1387–1401, 1989.