Chain Models and Finite Element Analysis

Richard S. Palmer*

TR 94-1406
January 1994

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

* This author's work was supported by the Advanced Research Projects Agency of the Department of Defense under ONR Contract N00014-92-J-1899, by ONR Contract N00014-92-J-1839 and by NSF Contract IRI-9006137.
Chain Models and Finite Element Analysis

Richard S. Palmer*
Department of Computer Science
Cornell University
Ithaca, NY 14853
rick@cs.cornell.edu

January 12, 1993

Abstract
Algebraic-topological chains defined over finite cell complexes have been proposed as a uniform computational means of representing physical objects, systems and properties. In this article, we introduce CHAINS, an algebraic-topological computer language for representing and computing the properties of a wide variety physical systems. In particular, we develop a CHAINS program that implements a finite element approximation to linear elasticity. In the process we illustrate the relationship between finite element analysis and the chain models methodology, showing that while finite element analysis is a specific numerical approximation scheme, CHAINS is a computer language for specifying and representing a variety of physical systems and approximations, of which finite element computations are one example.

1 Introduction

Chain models[PS93] have been proposed as a unifying formalism for representing physical systems, which can be used for a variety of purposes, including analysis of existing systems, and synthesis of new ones. Algebraic topological chains are used in this formalism to provide a bridge between geometry and physics, which allows statements about a physical system (e.g., conservation statements) to be expressed as algebraic expressions, much as we use algebraic expressions to represent the process of adding and multiplying scalars and vectors. We call such a symbolic specification of the algebraic properties of a physical system a physical element. Once defined, a physical element may be repeatedly and systematically applied to any desired regions of space to create a model of the physics in that region. In this article, we introduce a computer language, CHAINS, for programming with algebraic-topological mathematical objects, and use it to create a finite element code for approximating a linear elastic solid.

1.1 A computer language for physical systems

One might ask “Why do we need a language for representing physical systems?” After all, we already have:

1. Hundreds of years of investigation of the physical world, mainly using the “language” of ordinary and partial differential equations, integral equations, etc.

*This author’s work was supported by the Advanced Research Projects Agency of the Department of Defense under ONR Contract N00014-92-J-1989, by ONR Contract N00014-92-J-1839 and by NSF Contract IRI-9006137.
2. Methods for formulating discrete approximations to (1), e.g., finite elements, finite differences, finite volumes, boundary elements, lumping, etc.

3. A variety of methods and codes for solving these discrete approximations, such as direct and iterative linear systems solves, with variations for dense and sparse matrices, as well as nonlinear systems solvers and optimization codes. Additionally, we have multigrid, Fourier, and wavelet methods, etc.

4. Well known and understood computer languages for implementing the methods in (3), such as FORTRAN and C.

What is the purpose of adding a specialized "physics" language to all of the above, and what would such a language look like? These are the questions we address in this article. We begin by stating that the CHAINS language is not a substitute for any of the above. In fact, CHAINS itself is implemented using languages such as those in (4), and is intended to provide a higher level, more physics-focused means of expressing (2) and (3). Finally, CHAINS provides a symbolic construct, the physical element, which may be viewed as a discrete analog to the mathematical objects in (1).

A first question might be, "What is a computer language for physics?" A computer language is, in essence, a mapping from some set of "strings" (the expressions in the language) to some set of "meanings." For instance, the language of integer arithmetic may be viewed as a map from strings such as "(12+16) * 19" to numbers – the "meaning" of the preceding expression is the integer 532. The best analog to a computer language for physics is the "language" of geometric modeling. Geometric modelers are, in essence, maps from strings in some language (e.g., CSG[RV81]) to (computer representations of mathematical models of) geometric shapes. For physical systems, we desire a language that defines a map from some set of strings to (computer representations of mathematical models of) physical systems. We propose CHAINS as such a language, and the chain models formalism described in [PS93] as the target (i.e., computer representations and mathematical models) semantics for CHAINS.

In contrast to a special purpose representation language such as CSG, general purpose computer languages such as FORTRAN and C support computation on numbers and complex data structures, with complex control structures, such as goto and function calls. We may view languages in terms of a hierarchy of focus: For instance, assembly language focuses on moving information between parts of the computer, and on performing primitive operations on primitive data types, such bits, bytes, and words. FORTRAN extends these operations to focus on numerical operations on focuses floating point numbers. CHAINS, however, is focused on the algebraic-topological structure that underlies physical (and other) systems. Thus it is possible to "goto 30" in FORTRAN but this has no meaning in a physical context. On the other hand, FORTRAN does not contain any built-in data or control structure that is capable of naturally capturing the concept of conservation of energy. While it is possible to operate on individual floating point numbers in CHAINS, it is just as easy to operate on an entire region of space.

Because CHAINS represents physical systems at a high symbolic level, in terms of physical quantities and topological invariants such as conservation of mass, equilibrium, etc., it is possible to have the computer perform the symbolic manipulations necessary to construct executable models of physical systems, steps that are currently carried out by humans. This is in contrast to current physical model building paradigms, where the physical content is quickly lost, as the scientist or engineer is forced to immediately descend to the level of FORTRAN or C to express physical computations. Because these general purpose programming languages are not specialized to representing physical systems, much of the engineer's knowledge is implicit in the code, and can
not be operated on by the computer. This is the primary reason for the lack of “interoperability” and “reusability” in scientific and physical computing – the languages in which the concepts are typically expressed to not lend themselves to expressing high level abstract knowledge of the computation, and therefore this knowledge is seldom represented, and thus can not be effectively used, either by computer or human.

1.2 Previous work

1.2.1 Previous algebraic-topological approaches

Algebraic-topological approaches to systems modeling have been popular since Kirchoff, especially in the “lumped parameter” domains, roughly, those physics domains that are reasonably well modeled by time varying discrete mathematical systems (and thus by ordinary differential equations) (see, e.g., [BS90, Ols43, PC92, Pay61]). Applications of algebraic-topological methods to distributed parameter problems, that is, problems typically modeled with partial differential equations, such as the elasticity problem addressed in this article have also been extensively investigated [Bra66, EvD74, Kro89, Kro45, Kro59, Nic25, Ton75], mostly from a theoretical point of view, although physical analogies were used in the precomputer age to design circuits for approximating physical behavior.

When the finite element method began to gain popularity in the 1960’s, there was an interest in understanding how the topological methods described, e.g., in [Kro59] and [Pay61] were related to FEM. For instance, Bjørke [Bjø81] compared the finite element method to electrical networks, viewing each finite element as a “component” with “terminals” which are wired together to create a network. Similarly, the finite element method has been compared to the bond graph methodology [Pay61].

The approach described in this article builds on the work described in the references above, but in contrast to previous work, algebraic-topological structure of physical problems is placed in the context of computer languages and systems. Rather than simply show a mathematical relationship (or analogy) between two methodologies or physical models, we show how the algebraic-topological structure that is common to physical problems may be exploited by using the CHAINS language to create executable code.

1.2.2 Object oriented finite element analysis

More recently, the popularity of object oriented programming, and in particular, the C++ programming language, has sparked an interest in using object oriented techniques to finite element problems [Mac92, OTAY92, RK93]. Indeed there is a close relationship between the chain models methodology and object oriented methods. In particular, both approaches focus on the defining a systematic decomposition of complex systems into simpler interacting parts. In the case of chain models, however, the focus is on the means of specification of the decomposition, as opposed to choosing a particular problem decomposition and implementing it in an object oriented programming language such as C++. That is, rather than use a general purpose object oriented programming language to implement an object oriented FEM system, we define a special purpose language, which incorporates the modularity associated with object oriented systems, and which provides specialized data structures particularly suited for representing physical systems.
1.3 Chain model concepts

Just as a language such as C has basic components to define the kinds of data (such as int, char, and struct), and control statements (such as for, switch, and the function call mechanism), the CHAINS language has basic components, which are designed for representing physical systems.

Here we enumerate and briefly describe the basic chain model terms that will be used in this article. The first set of terms are the primary data structures in CHAINS:

**Cell** The basic topological abstract type, which may be used to represent a simple region of space.

**Cell complex** A set of cells, with well-defined intersection properties, that are used to represent more complicated regions of space.

**Chain** An algebraic object, used to associate physical quantities with the cells of a complex, e.g., to represent the position or mass of the cells.

**Chain constraint** A constraint on chains that may be used to represent mathematical and physical relationships and properties, such as equilibrium, conservation, and constitutive relations.

**Chain model** A computational model of a physical system based on cells, complexes, chains, and constraints.

**Physical element** A symbolic definition of a physical object or system, expressed in terms of abstract (unembedded) cells, complexes, chains, and constraints, which may be applied to (all or part of) a cell complex to create a chain model representing a physical behavior in the region decomposed by the complex.

In CHAINS, control is specified (using essentially declarative syntax) in two ways: First, causality is specified – this is equivalent to specifying which variables in a chain constraint are viewed as independent and which dependent. Second, a physical element may itself have conditions of validity, together with transitions to other elements. This form of control may be viewed as similar to a finite automaton control structure, where transitions between states are determined by (local) chain constraints. Discussion of the control aspects of CHAINS is minimized in this article.

A physical element may be viewed as a discrete, symbolic counterpart to continuum descriptions of physical behavior such as partial differential or integral equations, or differential forms. In contrast to such continuum equations, however, a physical element contains, by definition, all of the information necessary to create an executable computer model of the behavior. It is this reduction of the geometric and topological properties of a physical system to a well-defined algebraic expression that provides the power of the methodology: this enables us to develop abstract algebraic representations of physical behavior that can be applied to arbitrary cell decompositions of space to create models of the physics in that space. This process is illustrated in Figure 1, which shows the construction of a combined domain (coupled phenomena) model. The model includes two phenomena – a fluid and a solid, representing a fluid flowing through an elastic nozzle, and is constructed by applying physical elements representing 1) inviscid compressible fluid flow and 2) linear elasticity. Because both physical phenomena are expressed in terms of compatible physical elements¹, the interaction across the interface of the two domains may systematically constructed when the respective physical elements are applied to adjacent regions. The linear elastic physical element is developed in this article. A physical element for fluid flow, as well as experiments

---

¹Physical elements are compatible if they have been defined using a common set of physical quantities, or if a compatibility map has been defined between them.
conducted using these elements to create a combined domain simulator, will be described in a future paper.

There are many questions that arise when a "new" technology or methodology, such as chain models, is proposed. Among the questions regarding chain models are:

- What is the relationship between chain models and current numerical methods (and codes) for approximating physical behavior, such as finite element and finite difference methods?

- What advantages, if any, does the chain model formalism offer over current methodology for building physical models? What disadvantages, if any?

- Does using chain models mean jettisoning current methods and codes?

- What can be said about the performance of chain model based codes?

In this article we endeavor to address some of these questions, largely by way of example. In particular, we show how a standard finite element approximation to linear elasticity (using triangular quadratic elements) can be expressed in terms of algebraic-topological chains, by creating and using a symbolic physical element, and showing how this elastic physical element may be used to create executable code.

There are several advantages to be gained from expressing computations in terms of chains. For instance, a new type of finite element may be introduced into a physical modeling system by writing a few lines of high level algebraic (chain) expressions instead of expressing the computation in terms of a particular vendor’s choice of data structures (which may or may not be optimal or even possible), or by writing code in FORTRAN or C. The ability to express physical computations
at a high semantic level, directly in terms of physical quantities, properties, and invariants also encourages development of models that include interactions between physical domains, such as a fluid interacting with an elastic solid, all within a single unified framework.

1.4 Chain models and finite elements

Over the last 30 years the finite element method has shown itself to be an invaluable tool to the numerical analyst as well as the engineer. As Strang[Str88] states, "... the finite element method has made it possible to use more of the power of the computer — in constructing a discrete approximation, solving it, and displaying the results — than any other technique in scientific computation."

As we shall see, the chain models methodology is, (among other things,) a means of 1) computerizing the symbolic processing that characterizes the early stages of an FEM derivation, which are usually performed "on paper," 2) providing a uniform representation scheme for the computations, especially those that span the spatial, physical, and numerical domains (and thus a means of coupling FEM computations with other numerical methods), and 3) providing a high level language for specifying physical system behavior, which enables construction of "whole system" models comprising a multitude of independently defined components and domains of physics.

Most "mathematical" descriptions of the finite element method concentrate on the process of transforming the continuum descriptions of physical behavior into a discrete linear system[CME89, ZL89, Str88]. Rather than repeat such a derivation in this article, we shall assume the reader is familiar with the finite element method (FEM), and with the standard methods of deriving finite elements (e.g., variational and weighted residual methods). Instead, we focus on relationship between FEM, chain models, and physical elements. In particular, we focus on how CHAINS may be used to encode such a derivation, so that, instead of a manually translated FORTRAN code, the derivation itself becomes the "program,"

1.5 The CHAINS implementation

The algebraic-topological functionality described previously has been prototyped in the computer system CHAINS, which is implemented in COMMON LISP using the Common Lisp Object System (CLOS). CHAINS uses MATHEMATICA[Wol91] for symbolic computations (for instance to compute with chains having polynomials as coefficients), as well as FORTRAN libraries for numerical routines. The system generates optimized C, FORTRAN, and Common Lisp code, and examples to date have used numerical libraries including ODEPACK[Hin83], SPARSPA}[GL81], LINPACK[DBMS79], and MEXX[Lub90, LNPE92]. The software architecture of the current CHAINS prototype described in this article is illustrated in Figure 2.

The general goal of the CHAINS (as well as its predecessor, SIMLAB(PC91)) has been to raise the semantic level at which physical systems analyses may be "programmed" while maximizing performance of the resulting programs. For this reason we have partitioned the various tasks and assigned them to the computing subsystems most appropriate to their requirements. Thus we do not perform large numerical computations in MATHEMATICA or CLOS, nor do we attempt to implement abstract algebra or symbolic calculus in FORTRAN. Our design methodology is to represent the computation at the highest practical level (in symbolic form), and then perform symbolic calculation to prepare numerical codes in FORTRAN or C. The symbolic process is generally performed once, to create the simulator codes. These codes may then be repeatedly executed to perform calculations. These divisions are not immutable, however. We view each of these steps as generating "more optimized" (or "more executable") versions of the more symbolic
Figure 2: The software architecture of the prototype CHAINS language implementation
forms. The higher level, more abstract representations are "refined" (information is added and choices made) to get fast implementations.

Samples of CHAINS code are used in this article to develop a physical element for elasticity. While the current CHAINS system is only a prototype, it demonstrates a core functionality for everyday use in engineering, simulation, and design.

1.6 Syntax and semantics

The choice of syntax (the structure of the expressions) is independent of the semantics (the meaning of expressions) of a language. The best choice of syntax depends on the concepts, objects, and operations it must express, as well as the the background of the user (or reader) using it. The syntactic aspects addressing the concepts, objects, and operations might be referred to as the abstract syntax of the language, while the user-oriented aspects might be referred to as the concrete syntax. Depending on his or her background, an engineer may be most comfortable with FORTRAN, C or Mathematica expressions, a numerical analyst with MATLAB, and a computer scientist with COMMON LISP, ML, or PROLOG. For this reason, we choose to view the concrete syntax as an "inessential" part a computer language, to be modified as desired to address different communities. In particular, CHAINS is currently implemented with two syntaxes: COMMON LISP[Ste90] style-sexp syntax, and Mathematica[Wol91] style syntax. Expressions in either syntax are translated to the internal CHAINS representation, and computed results can likewise be viewed in either syntax. The presentation syntax for CHAINS used in this article is that of Mathematica. As an example, the following two forms are equivalent. First we have the sexp style syntax,

```
(setf cm1 (simplicial-complex '((0.0 1.0) (1.5 3.0)) '(1 1)))
```

and then the Mathematica style syntax,

```
Cm1 = SimplicialComplex[{0, 1.}, {1.5, 3.}, {1, 1}]`
```

It is important to note that these expressions, whether expressed in sexp or Mathematica syntax, should not be interpreted as COMMON LISP or Mathematica statements, and can not generally be interpreted independently of the CHAINS system.

2 Chain models and physical elements

Chain models and physical elements were introduced and are described in detail in [PS93], which also describes the motivation and several potential uses that are beyond the scope of this article. For this reason, we shall give a brief, intuitive description of the mathematical components of a chain model: cells, complexes, and algebraic-topological chains, which comprise the main tools we use to represent physical systems. The reader is referred to [PS93] for more detail.

After defining cells, complexes, and chains, we show how these relate to the components and processes of the finite element method, such as a finite element, the assembly process, etc. We will show that these finite element objects and concepts may be expressed in terms of analogous chain model constructs.

\[^2\text{This statement is one of the CHAINS expressions that was used to generate one of the illustrations in this article. Its effect is to create an embedded simplicial complex with two maximal dimensional 2-cells that together define the set } S = \{(x, y) | (0 \leq x \leq 1), (1.5 \leq y \leq 3)\}.\]
Figure 3: $n$-simplices, $n$-balls, and $n$-cubes for $n = 0, 1, 2, 3$. The arrows in this figure indicate that two "sets" are homeomorphic. The top row illustrates, from left to right, a 0-simplex, a 0-ball, and a 0-cube (all of which are homeomorphic to a point). Both $n$-simplices and $n$-cubes may be represented by an ordered list of their (possibly unembedded) vertices (their "corner points"). An $n$-simplex is represented by $n + 1$ vertices, while an $n$-cube is represented by $2^n$ vertices. Thus we see that for $n = 0, 1$, (when $n + 1 = 2^n$) the representation for $n$-simplices and $n$-cubes is identical, while this is not the case for $n > 1$.

2.1 Cells

In describing the concepts and components of the chain models methodology, we first consider $n$-cells (Definition A.4), which are topologically equivalent to unit balls in $\mathbb{R}^n$. A unit $n$-ball (the set of points $p \in \mathbb{R}^n$ whose distance from the origin is less than or equal to 1, see Definition A.1) is thus the canonical, and the most simply defined, $n$-cell. Such $n$-cells provide us with a "simple" prototypical topological region, upon which we may describe the desired behavior of a given physical problem.

All $n$-cells are homeomorphic to a $n$-ball (and hence to each other), and it is this equivalence that allows behavior defined in terms of a single cell to be systematically applied to a variety of geometric shapes. In this article, we never actually create a $n$-ball, or consider the map between a cell and a $n$-ball. However, this definition is the fundamental tool that allows us to apply symbolic definitions expressed in terms of abstract (unembedded) cells (i.e., physical elements) to arbitrary regions of space that are homeomorphic to a $n$-ball.

Unit $n$-balls for $n = 0, 1, 2, 3$ are illustrated at the center of Figure 3, which also illustrates two
classes of cells, the \textit{n-simplices} and \textit{n-cubes}.

We may extend the definition of the point set boundary of an \(n\)-ball \((\partial(B^n) = S^n\), see Definition A.2) to an \(n\)-cell \(c\) by defining \(\partial(c)\) to be the set of points mapped to \(\partial(B^n)\) by \(h\) of Definition A.4. (See Definition A.5.) The boundary of an \(n\)-cell is used in the definition of a cell complex (Definition A.7).

2.2 Simplices

Although the current \texttt{CHAINS} implementation defines and uses implementations of both \(n\)-simplices and \(n\)-cubes, only the simplices are used in the elasticity derivation described in this article. Hence most implementation descriptions apply only to simplices, as opposed to more general cells.

An \textit{n-simplex} is an \(n\)-dimensional generalization of the triangle (a triangle is a 2-simplex, while a tetrahedron is a 3-simplex). There are various possible definitions of embedded and abstract \(n\)-simplices[HY61, Mun84].

Definition A.4 defines an \(n\)-cell as an (uncountably infinite) point set. In order to represent such cells on a computer, we must find a (small) finite representation. We choose an abstract definition:

\textbf{Definition 2.1} An abstract \textit{n-simplex} \(c_n\) is a set of \(n + 1\) elements from some set \(V\):

\[ c_n = \{v_0, \ldots, v_n\}, v_i \in V, \quad \text{for } 0 \leq i \leq n \]

The elements of \(V\) are called \textit{vertices}, and \(v_0, \ldots, v_n\) are called the vertices of \(c_n\).

2.2.1 Representing \(n\)-simplices

\textbf{Representation 1 (SIMPLEX)} Based on Definition 2.1, we define a representation scheme for \(n\)-simplices in \texttt{CHAINS}: An \(n\)-simplex is represented as an ordered list of \(n + 1\) vertices. In \texttt{CHAINS}, a vertex is simply an element of some named abstract set. In other words, vertices (and hence simplices) in \texttt{CHAINS} need not be embedded in space. Embedding is achieved by defining chains, as described later. Thus, if the vertices are chosen from the letters of the alphabet, the \(\{A\}, \{C, B\}, \{A, B, C\},\) and \(\{A, D, C, E\}\) would defined 0, 1, 2, and 3-simplices, respectively. Note that \(A\) is a vertex while \(\{A\}\) is a 0-cell. Because of the obvious isomorphism between vertices and 0-cells, we do not often distinguish them when speaking. However, mathematically they are distinct objects, and therefore have distinct representations in \texttt{CHAINS}.

The abstract combinatorial simplex representation described in Representation 1 is readily extended to point set cells by considering the convex combinations of the vertices of a simplex. For instance, if we associate a point \(p_i \in R^3\) with each vertex \(v_i\), the cell may be viewed as defining the set of points:

\[ \{p = b_0p_0 + \ldots + b_np_n \mid \text{CVC}(b_0, \ldots, b_n)\} \]  \hspace{1cm} (1)

The predicate CVC defines the set of convex combinations of a discrete set of points, and is defined formally in Definition A.6. The \(b_i\) are called \textit{barycentric coordinates} of the point \(p^3\).

2.3 Cell complexes

If a cell describes a simple region in space, the notion of complex of cells provides a well defined means of "sewing together" these simple regions to represent more complicated regions.

\footnote{Often referred to as \textit{area coordinates} in the FEM literature.}
A cell complex \( K \) is a set of cells \( c \) of various dimensions. The key feature of a cell complex is that the intersection of any two cells in \( K \) must itself be a cell in \( K \) as well. This “closure property” of cell complexes allows us to map between algebraic expressions and geometry, as discussed in below in Property 1. A formal definition of cell complexes is given in Definition A.7.

The \textit{faces} of a cell \( c_i \in K \) are the cells in \( K \) that comprise its boundary. The \textit{subfaces} of a cell \( c_i \) defined as follows: 1) \( c_i \) is a subface of itself, and 2) if \( c_j \) is a face of a subface of \( c_i \), then \( c_j \) is a subface of \( c_i \). (That is, the set of subfaces of \( c_i \) is the \textit{transitive closure} of the face relation in \( K \).) Formal definitions of face and subface are given in Definition A.8 and Definition A.9, respectively, while Figure 4 illustrates the face relation in the complex generated by a single 2-simplex (i.e., all the subfaces of a 2-simplex). The arrows represent a path from a given cell to its faces. The coface relation may be visualized by reversing the arrows in Figure 4. The face/coface relationships between cells capture all incidence information in a cell complex.

\textbf{Notation} Let \( F(c) \) denote the faces of \( c \) and let \( CF(c) \) denote the cofaces of \( c \).

\textbf{Notation} Let \( F^*(c) \) denote the subfaces of \( c \).

**Representation 2 (Simplex Subfaces)** In terms of the simplex representation described in Representation 1, it is straightforward to verify that the set of faces of an \( n \)-simplex \( c = \{v_0, \ldots, v_n\} \) consists of all subsets of \( c \) that have \( n \) vertices. (Recall that an \( n \)-simplex \( c \) has \( n + 1 \) vertices, so \( c \) has \( C_n^{n+1} = n + 1 \) faces. Similarly, the \( p \)-subfaces of \( c \) (the subfaces of \( c \) of dimension \( p \)) are given by the \( \{s|s \subseteq c \land |s| = p + 1\} \). There are thus \( C_{p+1}^n \) \( p \)-subfaces of an \( n \)-simplex. It also follows that the set of subfaces of a \( n \)-simplex is isomorphic to the power set of its vertices.

Figure 5 is a schematic representation of a simplicial complex, showing the cells of various dimensions. The points represent 0-cells, the line segments represent 1-cells, and the triangles represent 2-cells. It is schematic in the sense that the 0-cells are actually the endpoints of the 1-cells (and the corner points of the 2-cells), while the 1-cells are the on boundary of the 2-cells (and thus interior 1-cells are the intersection of 2-cells). The schematic representation emphasizes that a \( K \) cell complex is not simply a graph (which would include only vertices and edges) or a set of triangles, but includes cells of all dimensions up to a given dimension \( K \).

**Property 1** The conditions that allow a set of cells to be called a cell complex provide an important property: if a cell complex \( K \) decomposes a region of space \( R \) (i.e., the union of the cells is equal to \( R \)), then every point in \( R \) may be associated with a unique lowest dimensional cell. This will prove useful in representing quantities distributed in space. Beyond this, the interaction between the physical behavior occurring in two adjacent cells \( c_i \) and \( c_j \) may be associated with their intersection, which by (Definition A.7), is a unique cell in \( K \) that is a subface of both \( c_i \) and \( c_j \).

We now consider \textit{oriented} cells. Orientations are used to partition the representations of cells into two equivalence classes, which may be used, for instance, to distinguish “inside” from “outside.”

**Representation 3 (Oriented Simplex)** We saw in Representation 1 that a simplex can be represented as a set of vertices. We may partition the \( n! \) orderings of these vertices into two classes, those that are even permutations of some reference ordering (which we will view as positively oriented), and those that are odd permutations (which we will view as negatively oriented) [Mun84]. Thus \textit{Chains} represents an oriented simplex as an ordered list of vertices.
Figure 4: The face relation in a simplicial complex. This figure uses a schematic representation of the simplicial complex generated by a single 2-simplex. The points represent 0-cells, the (straight) line segments represent 1-cells, and the triangle represents a 2-cell. It is schematic in the sense that (if the 0-cells are embedded in space as illustrated) the 0-cells are actually the endpoints of the 1-cells (and the corner points of the 2-cell), while the 1-cells are the on boundary of the 2-cell. The schematic representation emphasizes that a cell complex is not simply a graph (which would include only vertices and edges) or a set of triangles, but includes cells of all dimensions up to a given dimension $n$. The curved lines with arrowheads represent the face relation defined on $n$-simplices. For instance, the 2-simplex has three faces (the 1-simplices), pointed to by the arrows emanating from its center. Similarly, each of the 1-cells has two faces (0-simplices), which are pointed to by the arrows emanating from their centers.
Representation 4 (Simplicial complex) The properties of a cell complex (Definition A.7) and the subfaces of a cell (Definition A.9), together with the representation chosen for simplices (Representation 1) guarantee that the set of highest dimensional cells is sufficient to represent a simplicial complex. (A cell $c$ is highest dimensional in $K$ if it is not a face of any cell in $K$.) It is clear that Representation 2 defines a procedure for creating any subface of a face $c$. By Definition A.7, these are all the cells of $K$. For this reason, Chains represents a simplicial complex as a set of highest dimensional cells.

2.4 Chains

Where cells and cell complexes provide the mathematical machinery for decomposing space into simple regions, with sufficient structure to represent the relationships between these regions, chains are used to associate physical quantities with these regions. For computer representations, we use finite chains, i.e., chains defined over finite cell complexes.

A finite $p$-chain $ch(K, p, G)$ (see Definition A.11) is an algebraic structure, defined in terms of a complex $K$, and an algebraic group $G$, which we may view as $p$-cells of a complex $K$ to the elements of a group $G$. Notation Let $ch(c_i)$ denote the value of the coefficient (which may be a scalar, vector, or tensor, etc.) associated with the cell $c_i$ in $ch$.

From Definition A.11, we see that two mathematical objects are required to define a $p$-chain (in addition to a choice of some natural number $p$). The first is a complex $K$, which contains all of the discrete topological information (i.e., connectivity) required to define chains and chain operators. The second is an algebraic structure $G$, which is required to satisfy the definition of an abelian group. An abelian group $G = (S, +, 0)$ is a triple, where $S$ is a set, $+$ a binary composition on $S$, and $0$ is an identity in $S$ under $+$ [Jac74]. Examples of abelian groups are the integers, real numbers, and polynomials, all under the appropriate addition. Beyond these, various kinds of vector and tensor spaces under addition defined over integers, real numbers, and polynomials satisfy the abelian group properties (as well as other known algebraic properties not at issue here).

Finite chains have many attractive computational properties. If, in addition to the abelian group $G$ operation $+$, we have a suitably defined multiplication $\times$, the resulting $p$-chains form a vector space. In this case, the $p$-cells of $K$ can be viewed as basis elements for this space (by identifying $c_p$ and $0c_p$, where $0$ is the identity in $G$). For instance, if $G$ is a field, the set $Ch(K, G, p)$ of all $p$-chains over a given complex $K$ with coefficients in $G$ form a vector space, which enables us to use the all of the vector operations on chains. Thus, we may add two $p$-chains, or multiply a $p$-chain by a scalar.
Figure 6: Examples of $p$-chains of dimension $p = 0, 1, 2$ defined over a simplicial complex $K$. In this example, chain coefficients are integers. The top illustration denotes a 2-chain $ch_2$. The number displayed at the center of each 2-cell represents the coefficient of that cell in $ch_2$. Similarly, the center illustration denotes a 1-chain $ch_1$. The number displayed at the center of each 1-cell represents the coefficient of that cell in $ch_1$. The bottom illustration denotes a 0-chain, hence the number displayed at each 0-simplex represents the number associated with that cell.
Figure 6 schematically illustrates $p$-chains $Ch(K, Z, p)$, for $p = 0, 1, 2$, where $K$ is a simplicial complex, and $Z$ is the group of integers under integer addition.

Defining a $p$-chain in $\text{Chains}$ is straightforward. Assuming that $K$ is a simplicial 2-complex, and that $\mathbb{R}$ is the field of real numbers, we may define a 1-chain $Ch = Ch(K, \mathbb{R}, 1)$ as follows:

$$Ch = \text{Chain}[K, \mathbb{R}, 1]$$

2.5 Physical elements

This section describes a computational object, called a physical element, which defines the physical behavior in an abstract cell, much as a system of partial differential equations may be used to define the behavior in an infinitesimal control volume. In contrast to the case with partial differential equations, however, since a physical element is expressed in terms of a finite cell and its surfaces, this definition may be systematically applied to a cell complex that decomposes a region of space to create a model of the physical behavior in that region.

**Definition 2.2** A physical element is

1. The cell complex $K$ generated by a single $n$-cell $c_n$, of a particular cell type (e.g., simplex, cube).

2. A set of $p$-chains on $K$ that represent the physical state within $c_n$ and its subfaces.

3. A set of constraints on the $p$-chains of the element.

The cell $c_n$ (together with its subfaces) serves as a prototype; it is used for declaring the type of $p$-chains and allows definition of the structural and constitutive constraints. The relationships defined in the physical element can be applied to any cell of the appropriate type (e.g., 2-simplex).

In the sequel, we define a physical element for representing linear elasticity that implements a standard finite element solution using quadratic triangular elements.

3 Chain representations of piecewise polynomials

In this section we define some of the tools used to develop a chain model representation of a finite element discretization of a continuum problem.

The essence of the finite element method is the replacement of an (unknown) general continuous function $\phi$ with an approximation based on an (unknown) linear combination of simpler (known) polynomial functions. Thus we are interested in this section with chain representations of polynomials, which we might call polynomial (valued) chains.
Figure 8: An abstract 2-simplex $c_2 = \{V_0, V_1, V_2\}$, and a 0-chain, position, embedding $c_2$ in $\mathbb{R}^2$.

Figure 7 illustrates several well known finite elements defined over triangles. From the left, these are linear, quadratic, cubic, and quartic elements. Each of these elements defines a continuous function over the triangle by defining a polynomial which can be expressed as the inner product of a vector of (scalar, vector, or tensor valued) coefficients and a vector of “shape functions” – low degree polynomial functions. These mathematical objects (scalars, vectors, polynomials, etc.) will be represented with algebraic-topological chains.

Here we focus on the 2D quadratic triangular element, a finite element well suited for representing linear elasticity in complex planar regions[CME89]. It should be apparent from this derivation how one could use CHAINS to create other kinds of piecewise polynomial. The interested reader may wish to compare this derivation with that of [CME89].

We first define the prototype cell required in (1) of Definition 2.2. In this case, we choose a 2-simplex, i.e., a triangle, having abstract vertices $V_0$, $V_1$, and $V_2$.

$$K = \text{SimplicialComplex}[\{\{V_0, V_1, V_2\}\}]$$

Given the prototype complex $K$, we proceed to define “corner” functions. We first define a 0-chain pos that represents the position (in $\mathbb{R}^2$) of the 0-cells of $K$: $\text{pos} = \text{Ch}(K, \mathbb{R}^2, 0)$. Let $(x_i, y_i)$ be the coefficients of $V_i$, where $V_i$ the vertex of 0-cell $c_i$, as illustrated in Figure 8. The basis functions (which will be used below to construct the quadratic shape functions) are defined in CHAINS as follows:

- $e_0[x_-, y_-] := 1/(2 \, A) \, (x_1 \, y_2 - x_2 \, y_1 + x \, (y_1 - y_2) + y \, (x_2 - x_1))$
- $e_1[x_-, y_-] := 1/(2 \, A) \, (x_2 \, y_0 - x_0 \, y_2 + x \, (y_2 - y_0) + y \, (x_0 - x_2))$
- $e_2[x_-, y_-] := 1/(2 \, A) \, (x_0 \, y_1 - x_1 \, y_0 + x \, (y_0 - y_1) + y \, (x_1 - x_0))$

where $A$ is the area of the triangle, and is defined as:

$$A = ((x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0))/2$$

The basis functions $e_0, e_1, e_2$, which have value 1 at the 0-cell with which they are associated and 0 at the other 0-cells of the triangle, are now used to define the quadratic “shape functions.”

$$n_0 = e_0(2 \, e_0 - 1)$$
$$n_1 = e_1(2 \, e_1 - 1)$$
$$n_2 = e_2(2 \, e_2 - 1)$$
$$n_3 = 4 \, e_1 \, e_2$$
$$n_4 = 4 \, e_2 \, e_0$$
$$n_5 = 4 \, e_0 \, e_1$$
The piecewise polynomial \( u \) will be constructed from linear combinations of the shape functions:
\[
    u = u_0 \ n_0 + u_1 \ n_1 + u_2 \ n_2 + u_3 \ n_3 + u_4 \ n_4 + u_5 \ n_5
\]
To construct \( u \), we first define a vector of shape functions:
\[
    \mathbf{N}_v = \{n_0, n_1, n_2, n_3, n_4, n_5\}
\]
We now define a vector of coefficients:
\[
    \mathbf{U}_v = \{u_0, u_1, u_2, u_3, u_4, u_5\}
\]
and, using the vector of shape functions, \( \mathbf{N}_v \), define the quadratic polynomial \( u(x, y) \), which interpolates the nodal values:
\[
    U = \mathbf{U}_v \cdot \mathbf{N}_v
\]
Note that we have defined a relationship between an algebraic object (the vector \( \mathbf{N}_v \)) and a topological object (the relationship between the triangular element and its nodes). In order to have a well-defined algebraic expression that represents a polynomial over an element, we must have a means of consistently and systematically relating the nodes of an element to an algebraic vector. As we shall see in Section 3.2, CHAINS provides a means of doing this. Because current practice does not generally make use of such algebraic topological tools, mathematical derivations are not explicitly represented in computer programs, which is a great loss, since this precludes using the computer to perform a myriad of useful tasks in the creation of modeling software.

### 3.1 Consistency between cells

We have defined an algebraic means of specifying a chain \( U \) that defines a quadratic polynomial for each 2-cell in a complex \( K \). Note that we may represent this chain, as the inner product of a single vector \( \mathbf{N}_v \) (the vector of polynomial valued shape functions) and a 2-chain \( \mathbf{UvCh} \), where each chain coefficient is a vector \( \mathbf{Uv}^4 \):

\[
    \mathbf{UvCh} = \text{Chain}[K, \mathbb{R}^{[6,2]}, 2]
\]
\[
    U = \mathbf{N}_v \cdot \mathbf{UvCh}
\]

However, to define a piecewise polynomial over a region \( R \) that is decomposed by \( K \) (and \( \text{pos} \)), we must assure that these polynomials are single valued over \( R \). Figure 9 illustrates a chain \( U \) that does not define a single valued piecewise polynomial on \( K \) because nodal values are not consistent for elements meeting at a point.

By Property 1, we know that if \( u \) is single valued on each cell of \( K \), it is single valued on \( R \), since every point of \( R \) maps to a unique lowest dimensional cell of \( K \). We will therefore construct a set of polynomial valued chains \( \{U_0, U_1, U_2\} \), which define polynomials over the 0, 1, and 2-cells of \( K \), respectively, and that satisfy the property that if \( c_j \) is a subface of \( c_i \), then the polynomials they define are equal when restricted to the subface \( c_j \):

**Condition 1** If \( c_j \in F(c_i) \), then \( U_j(x,y) = U_i(x,y) \), for \( (x,y) \in CvC(\text{Pos}(c_j)) \)

We begin by constructing such a set of polynomials, and determining what constraints they must satisfy in order to define a single valued continuous piecewise polynomial over \( K \).

\(^{4}\)In CHAINS, \( \mathbb{R}^{[6,2]} \) is the group \( \mathbb{R}^{6 \times 2} \). Also, note that the inner product operator "\( . \)" is applied to a vector \( \mathbf{N}_v \) and a (vector valued) 2-chain \( \mathbf{UvCh} \). The result is a 2-chain \( U \) such that \( U[\text{cell}] = \mathbf{N}_v \cdot \mathbf{UvCh}[\text{cell}] \), for \( \text{cell} \in \text{TwoCells}[K] \).
Figure 9: Piecewise polynomials defined over 2-cells that do not define a piecewise polynomial for the complex. The vector displayed at the center the 2-cells is the coefficient vector for the cell. The values displayed at the "nodes" of the 2-cells represent the coefficient associated with that node. The fact that coefficients do not match across 1-cells illustrates the inconsistency.

Figure 10: Chains representing coefficient vectors for a polynomial defined over the subfaces of a 2-simplex (top), and (bottom) schematic representation of nodal position of polynomial coefficients.
Figure 10 illustrates such a set of polynomial chains. In the top figure, the subfaces of the 2-simplex are labeled with a coefficient vector whose inner product with the shape function vector yields the desired polynomial. The lower figure illustrates the nodal positions at which these coefficients are defined. If Condition 1 is satisfied, it must be the case that: \( u_1 = u_7 = u_{10} = u_{13}, u_6 = u_9, u_2 = u_8 = u_{14} = u_{18}, u_4 = u_{16}, u_3 = u_{11} = u_{15} = u_{17}, \) and \( u_4 = u_{12} \). If constraints such as these are satisfied for all cells and their subfaces, we have a chain representation of continuous piecewise polynomial.

The ability to algebraically represent such constraints between a cell and an arbitrary subface requires a systematic or canonical way of ordering the subfaces of a cell. We may then use this to order the terms of a polynomial so that we may relate the coefficients of different dimensional cells. We now describe a canonical ordering of subfaces, which we will use to induce an ordering on the fem nodes, and hence on the polynomial coefficients.

### 3.2 A canonical ordering of the subfaces of a \( p \)-simplex

As stated in Representation 1, an \( n \)-simplex \( c_n \) is represented by a list of vertices \( V = \{v_0, \ldots, v_n\} \), and the subfaces of \( c_n \) are given by the subsets of \( V \) (Representation 2). If we assume an ordering on \( V \) (this statement may be applied to any ordering of the vertices, but let us assume that we use the order in which the \( v_i \) appear in \( V \), as described in Representation 3), we may extend this ordering to all the subfaces of \( c_n \) by a lexicographic (e.g., dictionary) ordering on the vertices of the subfaces. For instance, CHAINS can use a standard dictionary order\(^5\). Thus if we have a simplex \( c_2 \), which is represented by \( V = \{v_0, v_1, v_2\} \), its subfaces in dictionary order would be:

\[
\{\{v_0\}, \{v_1\}, \{v_2\}, \{v_0, v_1\}, \{v_0, v_2\}, \{v_1, v_2\}, \{v_0, v_1, v_2\}\}
\]

(2)

For reasons of implementation efficiency, the default canonical subface ordering for a 2-simplex in CHAINS is:

\[
\{\{v_0\}, \{v_1\}, \{v_2\}, \{v_1, v_2\}, \{v_0, v_2\}, \{v_0, v_1\}, \{v_0, v_1, v_2\}\}
\]

(3)

Figure 11 illustrates the canonical ordering defined above (3) for the subfaces of \( p \)-simplices, for \( p = 0, 1, 2 \). The canonical orderings depend only on the combinatorial properties of the cell representation, and hence there is a canonical ordering of the subfaces of a simplex or cube of arbitrary dimension. Again we stress that the simple ability to canonically order cells is the key ingredient required to formally map between algebra of a vector expression and the topology of a finite element. For the case at hand, that of assuring that we have defined a piecewise polynomial, the canonical ordering provides an means of specifying that the chain coefficients are consistent, as we shall see in the next section.

In CHAINS, the function \( \text{Subfaces} \) returns the set of subfaces in canonical order:

\[
\text{Sf} = \text{Subfaces}[\text{Cell}]
\]

An alternate syntax, when only the subfaces of certain dimensions (in the following example, 0 and 1) are required is:

\[
\text{Sf} = \text{Subfaces}[\text{Cell}, \{0, 1\}]
\]

\(^5\)CHAINS provides the means to define other canonical orderings.
3.3 A partitioned representation of a piecewise polynomial

Here we define a related set of chains, called a ChainSet in CHAINS, which generalizes the concept of a finite $p$-chain. If we view a $p$-chain as a map from cells to coefficients, a chain set can be viewed as a more general map, one that maps from $p_i$-cells of various dimensions $p_i$, into (possibly distinct) groups $G_i$, for $0 \leq i \leq k$.

**Representation 5 (ChainSet)** A chain set is set of $p$-chains

$$\{\text{Ch}_0 = \text{Ch}(K, G_0, p_0), \ldots, \text{Ch}_m = \text{Ch}(K, G_m, p_m)\}$$

for $i \neq j \rightarrow p_i \neq p_j$, and $0 \leq m \leq \text{dimension}(K)$ defined over a given complex $K$.

In CHAINS, a chain set can be defined as follows

$$\text{Chs} = \text{ChainSet}[K, \{\{0, R2\}, \{1, R2\}\}]$$

This chain set maps every 0-cell and 1-cell to an element of $\mathbb{R}^2$, and will be used to define a virtual chain for representing a piecewise polynomial. CHAINS provides other means of creating chain sets, for instance if $\text{Ch}0$ and $\text{Ch}1$ are chains defined over $K$, we may use them to define a chain set as follows:

$$\text{ChS} = \text{ChainSet}[K, \{\text{Ch}0, \text{Ch}1\}]$$

3.4 Virtual chains

A virtual chain is a $p$-chain that is defined as a function of some other chains. In terms of is formal algebraic definition, a virtual $p$-chain is indistinguishable from any other $p$-chain – it differs only in it's implementation, and in the fact that the implementation assures that the defining functional constraints are satisfied.

We demonstrate the use of chain sets and virtual chains by using ChS as defined above to define a virtual chain $\mathcal{U}$ that satisfies Condition 1, i.e, is a single valued continuous piecewise polynomial.
Figure 12: Chain representations of piecewise polynomials defined over a 2-cell and its subfaces. The top figure represents the virtual p-chains $\mathcal{U}_0$, $\mathcal{U}_1$ and $\mathcal{U}_2$, while the bottom illustrates the chain set $\mathcal{C}_{\mathcal{S}}$ that forms the “basis” for the virtual chains.

Suppose $\mathcal{C}_0$ is a 0-chain representing the value of the polynomial $U$ at the 0-cells, and that $\mathcal{C}_1$ is a 1-chain representing the value of the polynomial $U$ at the centers of the 1-cells. Together these two chains represent $U$ at all the “nodes” of $K$. We define $\mathcal{C}_{\mathcal{S}} = \text{ChainSet}[K, \mathcal{C}_0, \mathcal{C}_1]$, and then define a virtual 2-chain $\mathcal{U}_2$ as follows:

$$\mathcal{U}_2 = \text{VirtualChain}[K, 2, R12, \mathcal{C}_{\mathcal{S}}, \text{Flatten}]$$

A 1-chain satisfying Condition 1 is defined by:

$$\mathcal{U}_1 = \text{VirtualChain}[K, 1, R6, \mathcal{C}_{\mathcal{S}}, \text{Flatten}]$$

The effect of the $\text{VirtualChain}$ statement is to apply the function (in this case Flatten) to the coefficients of the subfaces $p$-cells of appropriate dimension in $K$.

Figure 12 illustrates the relationship between the chain set $\mathcal{C}_{\mathcal{S}}$ and the virtual chains $\mathcal{U}_0$, $\mathcal{U}_1$ and $\mathcal{U}_2$ derived from it. In this case, the coefficients of $\mathcal{U}_2$ representing the polynomial defined over the triangle becomes completely “virtual” – all coefficients are functionally derived form subface chains. In contrast, $\mathcal{U}_1$ uses two coefficients associated with its faces and one of its own.

4 Chain representation of finite elements

As an example of how cell complexes, chains, and constraints can be used to define a chain model that is equivalent to an FEM solution, we develop a chain model for 2D linear elasticity based on triangular elements with quadratic shape functions.
4.1 Representing elasticity

In this section we construct a physical element for linear elasticity, using the chain representation of piecewise polynomials described in the previous sections.

The continuum model of elasticity[LL86] expresses physical behavior in terms of four basic quantities: displacement \( u \), strain \( \xi \), force \( f \), and stress \( \sigma \). The strain tensor defines the local deformation properties of a body at a point, while the stress tensor represents forces acting through infinitesimal planes passing through a point. These four quantities are related by three basic physical laws: kinematics conditions relating displacement and strain, equilibrium between body force and stress, and a constitutive relation (generalized Hooke's law) relating strain and stress.

Using the piecewise polynomial construction described in Section 3 we may define a displacement field by defining the coefficients at the nodes, that is, by defining a 0-chain that defines the displacement \( u \) at the 0-cells, and a 1-chain that defines \( u \) at the centers of the 1-cells. In the case of two dimensional elasticity, \( u \) is a function from \( \mathbb{R}^2 \rightarrow \mathbb{R}^2 \), the definition of the vector \( u \) becomes:

\[
\{u_{0x}, u_{0y}\}, \{u_{1x}, u_{1y}\}, \{u_{2x}, u_{2y}\}, \\
\{u_{3x}, u_{3y}\}, \{u_{4x}, u_{4y}\}, \{u_{5x}, u_{5y}\}
\]

From this construction we may then define the strain \( \xi \), which is the symmetric part of the displacement gradient: \( \xi = 1/2(\nabla u + (\nabla u)^\top) \). For two dimensional problems, the strain is a symmetric second order tensor, which may be represented as either a \( 2 \times 2 \) symmetric matrix, or as a vector \( \{\xi_x, \xi_y, \gamma_{xy}\} \). We choose the latter.

A finite element solution to a linear elasticity problem has the form \( Ku = f \), where \( u \) is a vector of nodal displacement coefficients, \( f \) is a vector of applied nodal forces, and \( K \) is the \( n \times n \) global "stiffness matrix" relating \( u \) and \( f \), where \( n \) is the number of nodes. In FEM terminology, \( K \) is "assembled" from individual stiffness matrixes \( K_e \), where \( e \) ranges over the elements.

In CHAINS, we may express \( K_e \) as a 2-chain, while \( u \) and \( f \) are chain sets implemented as described in Representation 5.

4.2 Symbolic computation of the stiffness matrix \( K_e \)

In this section we describe how the CHAINS environment was used to create the stiffness matrix used in this elasticity example. The symbolic operations described here are performed once to create a a function \( KeFun \) that, given the coordinates of a 2-simplex \( \{x_0, y_0, x_1, y_1, x_2, y_2\} \), creates a stiffness matrix \( Ke \) of the appropriate form.

The local stiffness matrix \( K_e \) is, in this case, of the form

\[
\int_{c_e} B^T DBdV
\]

where \( B \) is a \( 3 \times 12 \) matrix that transforms the nodal displacements \( u_e \), which are the coefficients of the piecewise polynomial \( u_e.Nv \), to the strain \( \xi \) within the 2-cell \( e \). \( D \) is the constitutive relation, for linear elasticity, the generalized Hooke's relation for plane stress:

\[
D = \frac{E}{(1-\nu^2)} \begin{bmatrix}
1 & \nu & 0 \\
\nu & 1 & 0 \\
0 & 0 & 2(1-\nu)
\end{bmatrix}
\]

(5)

The CHAINS definition of \( D \) is:

\[
D = E/(1-\nu^2) \ \{\{1,\nu,0\},\{\nu,1,0\},\{0,0,2(1-\nu)\}\}
\]
while the stiffness matrix $K_e$ is defined as follows:

\[
\text{dude} = \text{Jac}[u, \{e0, e1, e2\}]
\]
\[
\text{dedx} = \text{Jac}[\{e0[x,y],e1[x,y],e2[x,y]\},\{x,y\}]
\]
\[
\text{dudx} = \text{dude} \cdot \text{dedx}
\]
\[
\{(xix, gammaxy),\{gammaxy,xiy\}\} = \text{Sym}[\text{dudx}]
\]
\[
\text{strain} = \{xix, xiy, gammaxy\}
\]
\[
B = \text{MatForm}[\text{strain}, \text{Flatten}[Uv]]
\]

The first five lines define the strain:

1. Define `dude`, which is a symbolic representation of $\frac{\partial u}{\partial x}$, the matrix of partial derivatives of $u$ with respect to $x = \{x, y\}$.

2. Define `dedx` as $\frac{\partial x}{\partial x} = 1$.

3. Use the chain rule to compute `dudx` as $\frac{\partial u}{\partial x} \cdot \frac{\partial x}{\partial x}$.

4. Compute a symbolic expression representing the strain ($\xi$), which is the symmetric part of `dudx`.  \(^6\)

5. Put `strain` in the standard vector form.

6. Finally, we construct the matrix form of the strain, i.e., we construct a matrix $B$ such that $\text{strain} = Bu$. The function `Flatten` takes a list of nested lists and returns a single depth list. In this case it takes $\{(u0x, u0y), \{u1x, u1y\}, \{u2x, u2y\}, \{u3x, u3y\}, \{u4x, u4y\}, \{u5x, u5y\}\}$ to $\{u0x, u0y, u1x, u1y, u2x, u2y, u3x, u3y, u4x, u4y, u5x, u5y\}$. 

Finally, we construct a symbolic form of the stiffness matrix $K_e$, by symbolically integrating over the area of the simplex defined by the barycentric coordinates $\{e0, e1, e2\}$.

\[
K_e = \text{IntegrateSimplex}[\text{Transpose}[B] \cdot \text{D} \cdot B, \{e0, e1, e2\}]
\]

We may then take this symbolic form and construct a function which, given the coordinates of the vertices of a 2-simplex, returns the appropriate stiffness matrix:

\[
\text{DefineFunction}[\text{KeFun}, \{x0, y0, x1, y1, x2, y2\}, Ke]
\]

This form defines a function called `KeFun` having parameters $x0, y0, x1, y1, x2, y2$, and having as its body the value of $Ke$.

In the current implementation, the symbolic form describing function `KeFun` is optimized to remove common subexpressions, and a function generated in either FORTRAN or an optimized subset of COMMON LISP (with the type declarations the COMMON LISP compiler needs for code optimization). For the example described in this article, COMMON LISP was used for the stiffness matrix function, since `KeFun` is invoked once per triangle to create a stiffness matrix, while FORTRAN was used to solve the linear system, which is repeatedly solved for under a variety of loading conditions.

\(^6\)Note that this expression assigns values to the scalar (polynomial valued) variables $xix, gammaxy, gammaxy$ and $xiy$, rather than assigning the complete tensor to a single tensor valued variable

\(^7\)This transformation (Flatten) would not be necessary, except that we have formulated the constitutive relation as a $12 \times 12$ matrix, to be consistent with FEM practice. In CHAINS it would have been simpler to represent the constitutive relation as a $6 \times 6$ matrix with $2 \times 2$ block entries, i.e., the matrix relating (the unflattened) $Fv$ and $Uv$.  

23
4.2.1 Discussion

The methods described here could be used to create constitutive relations for other systems, and do not directly depend upon or use chains, per se. Another consequence is that stiffness matrices from other systems could as easily be used with CHAINS, given appropriate definitions of the canonical surface ordering\(^8\). This makes it possible to incorporate much of the vast engineering knowledge of specialized problems that has been encoded as finite elements into a CHAINS based system, enabling these models to interact with models from other physical domains.

It is important to notice that the kind of symbolic expression described in this section is a very powerful and convenient means of expressing certain classes of computation. Computations expressed in such a domain specific, high level, symbolic language are easier to create, understand, modify, and debug than those expressed in general purpose programming languages. For instance, CHAINS code described here constitutes the complete definition of a quadratic stiffness matrix. Much of this derivation may be reused to define matrices for other FEM models. For instance, a linear triangular element requires changing only the definition of \( Nv \) and \( Uv \):

\[
Nv = \{e0, e1, e2\} \\
Uv = \{\{u0x, u0y\}, \{u1x, u1y\}, \{u2x, u2y\}\}
\]

4.3 Assembly

The previous section described the symbolic definition of a stiffness matrix based on a quadratic piecewise polynomial defined over a triangle. From the chain models perspective, this is an example of a discrete model of elasticity: a set of rules for constructing an effectively computable model of the behavior of some physical system. Specifically, we view the previous section as defining a "generalized constitutive relation" between two sets of measurements – \( K_e \) defines a linear equality constraint between displacement measured at six nodes on the triangle, and "nodal force" at those same six nodes. Thus if we are given six values of nodal displacement, we may directly compute nodal force, and vice versa\(^9\).

In order to use this element to solve elasticity problems in geometric shapes that are not triangular or in triangular areas that are too large for a single polynomial to accurately represent elasticity, however, we partition the space with a simplicial complex \( K \), and apply the triangular construction to each 2-simplex. In FEM terminology, this is called assembly, and is viewed as defining a "global" stiffness matrix from the local ones defined above. The derivation we present is theoretically equivalent to the FEM description (as it must be, in order to be correct), yet is presented from perspective of the algebraic-topological structure of the problem and solution scheme. Cell complexes and chains provide a language that has certain advantages over matrix algebra. In particular, complexes more naturally encode the sparse interaction structure between the elements of an FEM formulation than do matrices. In fact, the implementation of chains in the CHAINS language is a kind of sparse matrix data structure.

Thus, from the chain models perspective, we have defined a generalized constitutive element that defines the physical behavior of (and is parameterized by) any triangular region in the plane\(^{10}\). The assembly process may be viewed as defining a set of coupled boundary value problems, as illustrated

---

\(^8\) CHAINS provides a means of defining alternate canonical surface orderings.

\(^9\) Also, we may compute mixed combinations of \( u \) and \( n_f \), given a consistent set of "applied" displacements and forces. That is, at each node we may define either the displacement or the applied force, and compute the other.

\(^{10}\) Any triangular region that satisfies the "conditions of validity" associated with this element. We do not discuss conditions of validity here, other than to say that they are expressions that can be tested to determine whether a physical element may be applied to a particular cell.
Figure 13: A finite element approximation viewed as a set of coupled boundary value problems. In this example, the triangles represent a finite elements — their (local) nodes are displayed as smaller dots located around each triangle. The larger dots represent the (global) nodes of the assembled problem, i.e., the 0-cells and the center points of the 1-cells. The global nodes, and the lines connecting them to the elements represent a displacement/force constraint — for all local nodes that have line to a global node, local forces must sum to zero, and displacements must be equal.

in Figure 13. Once the individual elements are defined, we complete the model by defining how they are coupled, i.e., we describe constraints the individual elements must satisfy in order to represent the desired model of physics (in the case a finite element approximation to linear elasticity) when applied to the cells of a complex $K$.

In this case, the topological basis of the chain models methodology provides a clear view of the physical nature of the problem — a view that is somewhat obscured by the usual finite element "matrix assembly" presentation. The topological view also brings us closer to other physical modeling languages and methodologies based on the underlying topology of physical problems[Ton75], such as electrical circuit theory[BS90], and the bond graph methodology[Pay61]. Rather than view the assembly process as that of assembling a matrix, we view a physical element as a definition of the set of quantities that define the state of a given "physics" (such as linear elasticity) for a single cell, together with the set of constraints these quantities must satisfy. These constraints fall into two categories: internal (constitutive) and external (structural)[PS93]. The internal constraints have been described in Section 4.2, and we proceed to the structural constraints.

Consider the constraints that must be satisfied in order to define a model of elasticity using the element described in Section 4.2. The interaction between two adjacent elements takes place through their surface (or point) of intersection. By the properties of a cell complex (Definition A.7), this intersection is either a 0-cell or a 1-cell in $K$. The first constraint, which we have previously discussed (Section 3.1), is that the displacement $u$ be well defined and continuous, and we have already described how this condition can be satisfied (Section 3.3): we simply represent the displacement field $u$ with a set of basis chains, which guarantees that $u$ is well defined. In terms of the analogy between elasticity and electrical circuits, this is equivalent enforcing Kirchoff's voltage law. Just as the voltage drop for any cycle of the circuit must be zero, the sum of displacement $u$ around any path must sum to zero, which has the consequence that, just as the voltage must be
identical for all conductors connected to a given point, so the displacement must be the same for all elements meeting at a point.

In Chains this constraint is expressed simply by stating that $U$ is represented by chain set, which, as we have seen in Representation 5, may be used to define a set of interrelated vector valued virtual chains (with coefficients $Uv$, as described in Section 3) for dimensions $p = 0, 1, 2$, which, together with the vectors of shape functions $Nv$ (Section 3), define a set of interrelated polynomial chains that represent $u$ over their respective $p$-cells.

\[
U = \text{ChainSet}[K, \{0, R2\}, \{1, R2\}]
\]
\[
U^2 = \text{VirtualChain}[K, 2, R12, \text{ChS}, \text{Flatten}]
\]

The second structural constraint we must satisfy is analogous to Kirchoff's current law: just as the sum of all currents passing through a point must be zero, so the resultant of all forces on any given region (of a system in equilibrium) must be zero.

In the formulation of this second class of constraints (the equilibrium constraints) we ensure that equilibrium is satisfied at every 0-cell of the complex $K$, which by the basis construction (Section 3.3), ensures that equilibrium is satisfied for every cell in $K$, and therefore for every region of $K$. (Note that, given any $U$, equilibrium is always satisfied for an individual elastic triangle, because the symmetry operator in line 4 in the definition of $B$ in Section 4.2. That is, an arbitrary displacement vector $Uv$ gives rise of a set of forces $Ke$. $Uv$ that satisfy the equilibrium conditions.) The forces $f$ on a given 0-cell originate from two sources: from the nodes associated with 2-cells, and from applied nodal forces $Af$, each of which we may represent as a chain set.

\[
Af = \text{ChainSet}[K, \{0, R2\}, \{1, R2\}]
\]
\[
F = \text{ChainSet}[K, \{0, P[R2]\}, \{1, P[R2]\}]
\]

In this example the chain set $F$ has polynomial coefficients. In Chains, the notation $P[R2]$ indicates a polynomial with coefficients in $\mathbb{R}^2$.

The specifying the applied forces is straightforward, as they are input to (or output from) the problem. We represent the forces due to interaction between elements (2-cells) as a chain set with the Sum operator, which defines a chain as a sum of (other) chains\(^1\):

\[
F = \text{Sum}[	ext{Cell, TwoCells}[K], \text{Pairs}[Ke \cdot Uv] \cdot \text{SubFaces}[	ext{Cell, \{0, 1\}]]
\]

Figure 14 illustrates this expression. The Pairs operator maps a vector of size $2n$,

\[
\{v_0, \ldots, v_{2n-1}\}
\]

into a vector of size $n$:

\[
\{\{v_0, v_1\}, \ldots, \{v_{2n-2}, v_{2n-1}\}\}
\]

Subfaces, as described in Section 3.2, returns a vector of the subfaces in canonical order. It is standard to identify a cell $c$ and the formal value $1 \times c$, where $1$ is the appropriate (multiplicative) identity element[Mun84], and in doing so we find that expression above defines a chain set $F$, with coefficients in $\mathbb{R}^2$, which represents the sum of the surfaces forces (resolved to the nodes by the basis construction, Representation 5) on the nodes (i.e., virtual 0-cells) of $K$. Thus the equilibrium constraint may be expressed as:

---

\(^{11}\) Actually the Sum operator is defined over any indexed abelian group $G$. In this case, the elements of $G$ are chain sets, which are indexed by the 2-cells of $K$. 

26
Figure 14: The assembly process represented algebraically as a sum of chains. In this example, the 2-cells are assigned letters of the alphabet from left to right, and the nodes of each 2-cell are numbered canonically. The vector drawn at the center of each 2-cell represents the coefficient of $Fv$ associated with that cell. The values drawn at the nodal positions of the 2-cells represent the nodal forces arising from the displacement $U$, and are algebraically related to $Uf$ by the canonical ordering. Each 2-cell in the complex $K$ gives rise to a chain set representing the nodal force representation due to displacement. (That is, “$Fv . \text{Surfaces[cell, 0, 1]}$” is chain set.) When these chain sets (one for each 2-cell in $K$) are summed, the result is a 0-chain $F$ representing the nodal force – the resultant of all forces due to displacement on the 0-cells of $K$. This sum is represented by the expressions located at the nodal points of $K$, e.g., the lower-center 0-cell is labeled “$A0 + B2 + C2$”: the sum of the coefficients of the three chain sets defined by the three incident 2-cells.
\[ 0 = F + Af \]

Appendix B gives the complete definition of a function \texttt{KeFun} that, given the coordinates of a 2-simplex, returns a local stiffness matrix \(Ke\), which is a \(12 \times 12\) symmetric matrix defining the constitutive relation between the vector of displacements \(u\) and the vector of nodal forces \(F\), as derived in Section 4.2.

Assuming that we have a function \texttt{KeFun} as described above, a physical element for linear elasticity, which is implements a finite element approximation to linear elasticity (plane stress), using quadratic shape functions defined over 2-simplices is defined as follows:

```plaintext
ElasticPE =
PhysicalElement[
  CellType == TwoSimplex,
  ZeroChains == {Pos[R2]},
  TwoChains == {Uv[R[6, 2]], Fv[R[6, 2]], Ke[R12xR12]},
  ChainSets[{0, 1}] == {U[R2], Af[R2], F[fP[R2]]},
  Constraints ==
    {Ke = KeFun[Pos],
     Uv = Flatten[U . Subfaces[{0, 1}]],
     Fv = Ke . Uv,
     F = Sum[Cell, TwoCells[K],
                Pairs[Ke . Uv] . SubFaces[Cell, {0, 1}]],
     0 == F + Af}]
```

Thus we see that to create the physical element \texttt{ElasticPE}, as described in Definition 2.2, we collect together the type definitions of the chains defined above, together with \texttt{Chains} expressions described in this section.

### 4.4 Discussion

We discussed the potential value of a language for representing physical systems computations in Section 1.1. In particular, we believe that the ability to express computations in a mathematically well defined, high level, concise, and expressive language is essential realizing the potential of computational physical systems modeling, and perhaps, scientific computing in general.

In the preceding section, we see the reduction of the "assembly process" of the finite element method expressed algebraically – a single, mathematically well defined algebraic-topological expression that: 1) is easy for a human being to understand, verify, and adapt to other uses, and 2) may be symbolically operated on by a computer to create numerical codes correctly implement the algebraic relationships that are specified.

### 4.5 Creating an executable model of elasticity

One of the benefits of a high level symbolic description of a physical problem is that it may be used to create a variety of computations. In the example above, we have specified a linear system which is an approximation to the continuum model of linear elasticity.

We may then choose among solution methods to fit the nature of the problem instance (e.g., condition number, sparsity structure, etc.), available computational resources (e.g., computers, numerical libraries, etc.) Currently, the \texttt{Chains} environment contains a few linear systems solvers. For the sample at hand, which is \texttt{SPD} (sparse positive definite) we have used two classes of methods.
The first is the direct sparse solver SPARSPAK [GL81]. The CHAINS environment contains a SPARSPAK interface module, which allows a chain constraint to be solved using SPARSPAK.

\[
\text{SolutionScheme[\text{ElasticPE, Unknowns == U, Sparspak]}
\]

This expression causes the CHAINS environment to create code that, when given a cell complex \( K \), which has a 0-chain \( \text{pos} \) defined, will create a mathematical model of linear elasticity, as described in this article, which may then be (once or repeatedly) given boundary conditions, and solved.

The second class of solvers includes iterative methods, such as Jacobi, Gauss-Seidel, and SOR (successive over-relaxation). To use one of these methods, the SolutionScheme specification above is changed appropriately. For instance,

\[
\text{SolutionScheme[\text{ElasticPE, Unknowns == U, SOR]}
\]

This creates an executable solver based on the SOR method. Thus, while the specification (the physical element) is identical, the resulting implementation is vastly (and structurally) different. In the case of the SPARSPAK solver, the symbolic information contained in the physical element is operated on to create code that fills in the SPARSPAK data structures and calls the appropriate FORTRAN routines to solve the system. On the other hand, the direct methods, which are much simpler use the CHAINS data structures themselves to perform the computation.

When a model of physics is represented as a physical element the semantics of the problem is symbolically encoded at a high level. This enables the computer to specialize the specification to suit individual problems, resources, etc., as illustrated above.

The symbolic representation enables properties of a solver to be encoded at a higher level, which in turn allows implementation decisions to be postponed. This partitioning of the problem of creating physical systems codes (i.e., the software architecture of the CHAINS environment) is possible because CHAINS represents (and operates on, to create code) the semantics of the physical problem in terms of algebraic-topological chains.

5 Conclusion

In the introduction, we posed several questions about chain models, physical elements, and the relationship between chain models and finite element analysis. Here we re-visit those questions.

We also discussed the concept of a “domain specific” computer language for representing physical objects and systems, and for specifying computations involving physical systems. We have demonstrated this approach by developing the chain models methodology to represent physical systems, the CHAINS computer language to express algebraic-topological properties, and the CHAINS computing environment as a first implementation of, and testbed for, the methodology.

5.0.1 The relationship between chain models and FEM

As we have seen, the chain models approach provides an algebraic-topological view of physical systems, and formally defines a relationship between physical systems and the computations used to approximate them. The CHAINS programming language is a computer implementation of the functionality required for defining and computing with algebraic-topological representations of physical systems (i.e., chain models). On the other hand, the finite element method is a particular class of approximation schemes for defining computer models of physical systems. Thus chain models are a framework for understanding the finite element method, and CHAINS is a language that may be used to express finite element derivations, as described in this article.
5.0.2 The advantages of using CHAINS

The primary advantage of using CHAINS is in raising the semantic level at which the computer is programmed. Each time we are able to raise this level, we shift the part of burden of encoding a problem from the programmer, scientist or engineer on to the computer itself. There are several computational advantages to be gained by this shift. First, more of the meaning of a computation is represented, and can thus be used (by the computer) to make decisions. For instance, depending on the computational resources available (e.g., PC, workstation or supercomputer), the knowledge that a particular linear system is SPD (Sparse Positive Definite) can used to map a single CHAINS specification into completely different implementations. Second, because more of the meaning is represented, it is possible to operate on the (physical element representations of the) physics domains themselves – to combine two domains to get a multiple domain problem (as described below), as well as to simplify (e.g., lower the dimension of) or modify an existing physical element to create a new one (for example, to symbolically create a specialized elastic element, such as a beam or plate, from a more general 3D elasticity element). Finally, because CHAINS is a concise and simple language, it is easier to encode physical phenomena in the first place, as well as to understand and debug, etc.

5.0.3 Performance

We have not really addressed the performance issues in this article. However, it is fair to say that the issues discussed in the Section 5.0.2 suggest that there is very little to be lost and much to be gained in raising the semantic level of physical systems specification and computation. In particular, experience with the example of this paper has shown the value of being able to switch between (very different, e.g., direct and iterative) solution methods for the large sparse positive definite systems generated by the physical element developed in this article.

5.1 Other applications

This article has illustrated how the CHAINS, an algebraic-topological programming language, has been used to create a computer code implementing a standard finite element formulation of linear elasticity, and how the chain models approach to representing physical systems relates to finite element analysis.

Current and future work related to CHAINS, chain models, and physical elements:

Other domains We have also been using CHAINS to write computer models of (i.e., physical elements for) other physical domains, such as fluid flow. We have recently been experimenting with a physical element for two dimensional, inviscid, compressible fluid flow, based on a Lax-Wendroff discretization.

Multiple domain problems A primary advantage of representing a physics model in CHAINS is that symbolic information about the physical domain itself is represented, which may be used in concert with other chain model representations to create multiple domain models. For instance, we are currently working to create a combined fluid/solid model that is a simplified representation of a deflector in the exhaust nozzle of a jet engine. The deflector will be modeled with the elastic physical element defined in this article, while the fluid will be modeled with the fluid element described above.

Other domains The algebraic-topological notions of cell, complex and chain may be applied to variety of other domains. In particular, we are interested in applications in general systems
theory and distributed and parallel computer systems. We are just beginning to consider application of CHAINS to these domains. Great synergy will be gained if it becomes possible to use the same computer language and system for expressing such a variety of systems.

5.2 Recap

This article has introduced CHAINS, a computer language whose primitive types are based on the algebraic-topological concepts of cells, cell complexes, and chains. We have seen that CHAINS provides a high level means of "programming" physical systems (that is, e.g., programming the desired behavior of an elastic solid), and that such a program may be compiled into executable computer code. In particular, we developed a standard quadratic triangular finite element for linear elastic plane stress. In the process we demonstrated the extent to which CHAINS is a high level language suitable for symbolically expressing the complete derivation of a finite element approximation.

When expressing physical computations in CHAINS, an engineer or scientist "programs" in terms of geometric regions, fields, and physical properties instead of individual floating point numbers, DO loops and GOTO statements. This has two effects. The first is to allow the scientist to focus on the problem at hand rather than on the mundane, time consuming, and error prone process of "hand translating" scientific ideas into a general purpose computer language. The second is to allow techniques from symbolic processing, language translation and compiler optimization to be used (on the CHAINS code) to create executable code, which in turn may use the latest standard libraries and packages for symbolic, numerical, user interface functionality.

Acknowledgements

Vadim Shapiro of General Motors Research and Palghat Ramesh of Xerox Corporation read early drafts of this article, and provided valuable comments. Bharat Bagepali, Mark Kolb, Dipen Moitra, Will Schroeder and other researchers at GE Corporate Research provided the jet engine nozzle problem that was the genesis of the example used in this article. The author would like to thank these people, and the corporations they work for, as well as the Design Research Institute (DRI) at Cornell University.

A Definitions

A.1 Balls and cells

Definition A.1 \( B^n \) is the closed unit \( n \)-ball, a subset of \( \mathbb{R}^n \): \( B^n = \{ x \in \mathbb{R}^n \mid \|x\|_2 \leq 1 \} \).

Definition A.2 The (point set) boundary of \( B^n \), \( \partial(B^n) \) is the closed unit \( n \)-sphere \( S^n \subseteq B^n \): \( S^n = \{ x \in \mathbb{R}^n \mid \|x\|_2 = 1 \} \).

Definition A.3 Two sets \( S_1 \) and \( S_2 \) are homeomorphic if there is a continuous, invertible, 1-1 and onto function \( h \) mapping \( S_1 \) to \( S_2 \): \( h : S_1 \rightarrow S_2 \)[Mun84].

Homeomorphism defines the concept of "topologically equivalent," e.g., the class of all sets that are homeomorphic to some "reference set," such as the \( n \)-ball \( B^n \):

Definition A.4 An \( n \)-cell \( c \) is a set that is homeomorphic to a \( n \)-ball \( B^n \).

Definition A.5 The (point set) boundary of an \( n \)-cell \( c = h^{-1}(B^n) \) is defined to be \( \partial(c) = \{ x \in c \mid \|h(x)\|_2 = 1 \} \).
Figure 15: An example of a simplicial decomposition of a region of space that is not a complex. The set of simplices fails to satisfy the complex property because the intersection of A and C is not a unique subface of both. The same statement is also true of B and C. On the other hand, if we were to remove C from K, the set K becomes a simplicial complex.

A.2 Convex combinations

Definition A.6 The predicate CvC defines the set of convex combinations of a discrete set of points:

\[
\text{CvC}(b_0, \ldots, b_n) = \begin{cases} 
  b_0 + \ldots + b_n = 1 \\
  \text{AND} \\
  0 \leq b_i \leq 1, \text{ for } 0 \leq i \leq n
\end{cases}
\] (6)

A.3 Cell complexes

Definition A.7 A cell complex K is a set of cells that satisfies the following properties:

1. The boundary of each p-cell c ∈ K is a finite union of (p−1)-cells c_j ∈ K:

   \[ \partial(c) = \bigcup_{c_j \in K} c_j \]

2. The intersection of any two cells c_i, c_j in K is either empty, or is a unique cell in K that is a subface of both c_i and c_j. (The definition of the subfaces of a cell follows.)

Definition A.8 The faces of an n-cell c ∈ K are the (n−1)-cells in K comprising its boundary. If f is a face of c, then c is a coface of f. Thus, the cofaces of an n-cell c are the (n+1)-cells in K that have c as a face.

Figure 15 shows an example of a simplicial decomposition that does not satisfy property (2) of Definition A.7, which requires that if there is any intersection between two cells in K, that intersection is represented as a unique cell in K. In this case, the intersection of 2-cells A and C is not a unique cell in K. This is also true of B and C. Thus K is not a simplicial complex.

Definition A.9 A subface of c ∈ K is an element of the transitive closure of the face relation in K. Thus:

1. c is a subface of itself
2. if $c_j$ face of $c_i$ and $c_i$ is a subface of $c$, then $c_j$ is a subface of $c$.

**Definition A.10** An *oriented* cell is a pair $c = (u, o)$, where $u$ is an (unoriented) cell, and $o \in \{1, -1\}$.

**A.4 Chains**

**Definition A.11** A *p-chain* $ch$ defined over a complex $K$, and an abelian group $G$ is a formal sum $\sum_{c_i \in p\text{-}cells(K)} g_ic_i$ of *p-cells* of $K$ with coefficients $g_i \in G$.

**A.5 Canonical ordering of subfaces**

Suppose that our representation for a $p$-simplex $c_p$ is a list of vertices $v_0, \ldots, v_p$, as described in Representation 1. Let us impose a well-ordering on the subfaces of of $c_p$ by first imposing a a well-ordering on the vertices:

$$v_i < v_j \iff i < j$$

We may then extend this ordering to the subfaces as follows: Suppose we have two subfaces $c_i$ and $c_j$, where $c_i = (v_0, \ldots, v_{p_i})$, $c_j = (w_0, \ldots, w_{p_j})$, and that the vertices of $c_i$ and $c_j$ are in sorted order (e.g., $v_l < v_m$ for $0 \leq l < m \leq p_i$). We may order the subfaces with the following ordering:

1. $c_i < c_j$ if $p_i < p_j$. (Lower dimensional simplices are less than high ones.)

2. If there is some $k \leq p_i = p_j$ such that $v_i = w_i$, for $0 \leq i < k$, and $v_k < w_k$, then $c_i < c_j$, (That is when two simplices have the same dimension, the ordering is determined by the least vertex in which they differ.)

Thus we have a form of lexicographical ordering of the subfaces of a $p$-simplex. (Actually, this ordering applies to any set of simplices defined over a well-ordered set of vertices.) It is straightforward to verify that the above definition defines a well-ordering.

**B Chains code for FEM model of elasticity**

We begin with the complete definition of a function KeFun that, given the coordinates of a 2-simplex, returns a local stiffness matrix $Ke$, which is a $12 \times 12$ symmetric matrix defining the constitutive relation between the vector of displacements $\mathbf{u}$ and the vector of nodal forces $\mathbf{f}_v$, as derived in Section 4.2.

```plaintext
(* barycentric coordinate functions *)
e0[x_,y_] := 1/(2 A) (x1 y2 - x2 y1 + x (y1 - y2) + y (x2 - x1))
e1[x_,y_] := 1/(2 A) (x2 y0 - x0 y2 + x (y2 - y0) + y (x0 - x2))
e2[x_,y_] := 1/(2 A) (x0 y1 - x1 y0 + x (y0 - y1) + y (x1 - x0))

(* quadratic shape functions *)
n0 = e0(2 e0 - 1)
n1 = e1(2 e1 - 1)
n2 = e2(2 e2 - 1)
n3 = 4 e1 e2
n4 = 4 e2 e0
```

33
n5 = 4 e0 e1

(* vector of shape functions *)
Nv = {n0, n1, n2, n3, n4, n5}

(* vector of displacements *)
Uv = {{u0x, u0y}, {u1x, u1y}, {u2x, u2y}, {u3x, u3y},
     {u4x, u4y}, {u5x, u5y}}

(* function u[x,y] *)
u = Nv . Uv

(* strain computed from gradient of u *)
dude = Jac[u, {e0, e1, e2}]
dedx = Jac[{{e0[x,y], e1[x,y], e2[x,y]},{x,y}}]
dudx = dude.dedx

{{xix, gammaxy}, {gammaxy, xi0}} = Sym[dudx]
strain = {xix, xi0, gammaxy}

(* compute matrix mapping u[x,y] to strain[x,y] *)
B = MatForm[strain, Flatten[Uv]]

(* constitutive relation for plane stress *)
D = E/(1 - nu*2) {{1, nu, 0}, {nu, 1, 0}, {0, 0, 2 (1 - nu)}}

(* integrate over 2-simplex *)
Ke = IntBcent[Transpose[B] . D . B, {e0, e1, e2}]

(* Define function KeFun that returns stiffness matrix *)
DefineFunction[KeFun, {x0, y0, x1, y1, x2, y2}, Ke]

KeFun is then used to define a physical element for linear elasticity (plane stress), based on a finite element approximation using quadratic shape functions defined over triangles.

ElasticPE =
  PhysicalElement[
    CellType == TwoSimplex,
    ZeroChains == {Pos[R2]},
    TwoChains == {Uv[R[6, 2]], Fv[R[6, 2]], Ke[R12xR12]},
    ChainSets[{0, 1}] == {U[R2], Af[R2], F[fP[R2]]},
    Constraints ==
      {Ke = KeFun[Pos],
       Uv = Flatten[U . Subfaces[{0, 1}]],
       Fv = Ke . Uv,
       F = Sum[Cell, TwoCells[K],
                Pairs[Ke . Uv] . SubFaces[Cell,{0,1}],
       O == F + Af}]}
Finally, the physical element ElasticPE may be used to create a code for solving plane stress problems by defining a solution scheme:

SolutionScheme[ElasticPE, Unknowns == U, Sparspak]

References


