FEAP - A Finite Element Analysis Program

Version 8.5 Contact Programmer Manual

R.L. Taylor
Department of Civil and Environmental Engineering
University of California at Berkeley, USA
e-mail: rlt@ce.vulture.berkeley.edu

G. Zavarise
Department of Innovation Engineering
University of Lecce, Italy
e-mail: giorgio.zavarise@unile.it

October 2017
# Contents

1 Introduction
   1.1 Description of basic characteristics ................................... 2
      1.1.1 Surface definition .................................................. 3
      1.1.2 Restrictions on input data ....................................... 5
   1.2 Contact input commands .................................................. 6
      1.2.1 Command structure .................................................. 7
   1.3 Description of subprogram structure ................................... 15
      1.3.1 Sizing of arrays .................................................... 20
      1.3.2 Contact command control table .................................. 21

2 Contact driver: The CELMTnn subprogram .............................. 24
   2.1 Control data tables ..................................................... 26
   2.2 Pair data: Surface arrays .............................................. 27
   2.3 Material data ............................................................ 30
   2.4 History data management and assignment ........................... 30
   2.5 Options in driver program ............................................. 34
      2.5.1 Lagrange multiplier constraints ................................ 38
List of Figures

1.1 Mesh for indentor and platen for contact ......................... 4
2.1 Sequential search for LINE surface ............................... 28
2.2 Reverse search for LINE surface ................................. 29
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>COMMAND Options</td>
<td>12</td>
</tr>
<tr>
<td>1.2</td>
<td>Surface SUB-COMMANDS Options</td>
<td>12</td>
</tr>
<tr>
<td>1.3</td>
<td>Contact pair FEATURE options</td>
<td>15</td>
</tr>
<tr>
<td>1.4</td>
<td>Contact call actions based on \texttt{CSW} values</td>
<td>17</td>
</tr>
<tr>
<td>1.5</td>
<td>Contact call actions based on \texttt{CSW} values</td>
<td>18</td>
</tr>
<tr>
<td>1.6</td>
<td>Program set contact pair control array - \texttt{CP0}</td>
<td>22</td>
</tr>
<tr>
<td>1.7</td>
<td>User set contact pair control array - \texttt{CP0}</td>
<td>22</td>
</tr>
<tr>
<td>1.8</td>
<td>Variable names set in contact pair table</td>
<td>23</td>
</tr>
<tr>
<td>1.9</td>
<td>Contact surface control array - \texttt{CS0}</td>
<td>23</td>
</tr>
<tr>
<td>2.1</td>
<td>Definition of history variables</td>
<td>31</td>
</tr>
<tr>
<td>2.2</td>
<td>Activation of history variables</td>
<td>32</td>
</tr>
<tr>
<td>2.3</td>
<td>Parameters for use in contact driver programs</td>
<td>33</td>
</tr>
<tr>
<td>2.4</td>
<td>Existing calls to contact drivers (Part 1)</td>
<td>35</td>
</tr>
<tr>
<td>2.5</td>
<td>Existing calls to contact drivers (Part 2)</td>
<td>41</td>
</tr>
<tr>
<td>2.6</td>
<td>Existing calls to contact drivers (Part 3)</td>
<td>42</td>
</tr>
<tr>
<td>2.7</td>
<td>Indirect calls to contact drivers</td>
<td>42</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This manual is a short guide to describe the features of the FEAP contact algorithm. The contact algorithm comes with a small library of basic features. For the use of these existing features the algorithm can be treated as a black box. When implementing new contact formulations the algorithm may be treated partially as a black box. New contact formulations can be added similar to the way continuum elements are added; hence, the user is not directly involved in the management of arrays for history variables or in modifying some crucial data (e.g., the column height vector for the global stiffness matrix).

In the next paragraphs the basic input data organization is described. Moreover, the basic structure of the algorithm and the currently available features also are described. Finally, information is provided for users who are interested in implementing new features or their own contact formulation.

This manual is not intended to provide any detailed information about contact solution algorithms. However, it is assumed that the reader has some knowledge about how contact algorithms are solved using the finite element method. For example, some information on so called node to surface contacts may be found in reference [1] with additional information in references [2] to [58].
CHAPTER 1. INTRODUCTION

1.1 Description of basic characteristics

Independent modules are used in FEAP to define contact interactions between surfaces. The data input for a contact interaction is provided after the initial mesh is defined. Accordingly, contact data must follow the END mesh command and any TIE mesh manipulation commands. The description of the contact algorithm is initiated by a CONTact command and is terminated by an END command. Contact input data is divided into three main categories:

1. SURFace definitions.
   The SURFace definition is purely a geometrical description of any surfaces which may be considered in any analysis involving contact between bodies. A surface is defined as a group of element facets. A facet may be any geometric shape which the contact formulation can consider. Facets may be single nodes, edges of the finite elements defining each body, and/or faces of the finite elements.

2. MATEerial parameter definition.
   The MATErial parameter definition defines the constitutive characteristics of a contact surface. For analyses in which there is no constitutive equation for the normal direction but frictional behavior for sliding, the pseudo material model is called standard and defined by a STANdard command.

3. PAIR definitions.
   The PAIR definition defines two surfaces which can interact, as well as, the associated material constitution(s) and details for the solution algorithm to be employed.

FEAP uses the surface and material data sets to construct two independent control arrays which guide the overall solution process. As part of the control array construction, FEAP determines the total number of facets, number of material parameter sets, and the sets of pair data. A user need not specify the total number of pairs, facet or material sets (e.g., this is similar to FEAP’s ability to determine the total number of mesh nodes, elements, and element material sets in the problem). The pair data sets use the control array data sets to define and activate all contact elements which may then be assembled into the residual and tangent arrays during an analysis step. The use of the whole data structure is not mandatory. Consequently, a user may define contact surfaces or contact materials that are not used
within an analysis. This provides a flexibility to rapidly modify the characteristics of contact interactions. Moreover each contact pair may be enabled or disabled by specifying a feature option, without removing any data. Finally, the treatment of the contact part of an analysis can be deactivated simply by setting a flag. This feature permits a very efficient check on other features of the analysis without altering any contact data.

1.1.1 Surface definition

Each surface is defined as a group of facets. A facet is defined within the FEAP system by a sequence of global node numbers. For example, in a two-dimensional analysis involving surface interactions between solid elements modeled by three-node triangular finite elements (or four-node quadrilateral finite elements) a planar facet is defined by two nodes which are sequenced to traverse a boundary such that an outward normal points away from the body (i.e., the body lies to the left of the facet). This involves a counter clockwise traversing of the boundary curve.

A user has the option to use the FACEt command and define each facet by a its global node numbers (generation options are provided as described later) or to define a surface segment (similar to the BLOck or BLENd mesh commands) and let FEAP locate the facets which lie near the region defined by the surface segment.

As a simple example, consider the definition of a contact interaction between the indentor and the platen shown in Figure 1.1.

The FEAP input data for the contact part of the mesh shown in Figure 1.1 is given by:

```
FEAP * * Start of Problem
.......
END of mesh
CONTact
SURFace 1  ! Define first surface
  LINEar
  FACEt
     1 0 9 8
     2 0 8 7
```
Note that in the above example no MATERIAL parameters are specified. For the PAIR command a penalty method is requested and its parameters are associated with the solution algorithm, not material characteristics. On the other hand, if a frictional contact is necessary, a frictional constitutive model must be defined. For a Coulomb model where normal contacts are rigid the required data is:

MATERIAL 1
If a penalty method also is used to impose the friction, the solution strategy record would be modified to:

\[
\text{SOLUTION PENALty 1.E+05 1.E+04}
\]

where, now, the first value applies to interactions normal to surfaces and the second to tangential interactions (i.e., the frictional behavior).

The structure of the algorithm consists of a basic skeleton which can be treated as a \textit{black box} also from the programmers view point. This skeleton governs the whole data management and the data exchange within \textit{FEAP}. The user can program and add new subroutines for data input of particular geometry, or automatic geometry data generation. In the same way routines to read data for a user specified material model can be added, as well as the implementation of completely new contact algorithms. Data input is organized by keywords. A dictionary of keywords is defined is defined by the programmer and, in the case of new algorithms, every new keyword should be recorded within the subprogram \textit{CONTINIT}.

\subsection{Restrictions on input data}

For the currently implemented input data forms there are some restrictions on use. These are:

1. A contact surface must be defined with facets all of the same type and number of nodes.

2. The surface element definition is strictly related to the continuum discretization.

3. A surface should pertain to only one region.

4. The same material properties are attributed to the entire surface or to the whole pair. They may be nonlinear or involve history type variables to model such phenomena as wear.
1.2 Contact input commands

All the contact commands should be placed immediately after the END of mesh data and any mesh manipulation data (i.e., TIE or LINK commands, and should be included within the contact start command CONTact and the end command END.

Contact data are divided into three basic parts: (a) Definition of surfaces; (b) Definition of contact constitutive laws; and (c) Definition of contact pairs. There is complete independence of the data between the contact surfaces and the contact material sets. The coupling is carried out by a proper set of input data for the PAIR command.

The following is a second example of data file:

```
FEAP * * example input file
......
mesh data
......
END
CONT
SURF 123 First surface
  LINE 2
  FACE
    1 1 1 2
    10 0 100 101

SURF 27 Second surface
  LINE 2
  FACE
    1 1 700 701
    10 0 710 711
  BLOC SEGMENT
    1 0. 0.
    2 10. 2.
    3 5. 0.5

  FACE
    1 1 711 712
```
In the preceding example indentation has been used to clarify interdependence between data sets.

1.2.1 Command structure

All contact commands have a standard structure:

```
CONT,string,#,#
   COMMAND, #, Comment label
  type,  #1, ...., #15
  type data (optional)}
     ! blank record closes type data if they exist
   feature, option, #1, ...., #14
   feature, option, #1, ...., #14
   feature, option, #1, ...., #14
   sub-command, option, #1, ...., #14
   subcommand data
   subcommand data (optional)
     ! blank record closes subcommand data and command
```

```
COMMAND, #, Comment label
    ....
    ....
END
```
Every command set is terminated one or more blank records. A single blank record also terminates input for a type and/or sub-command data set. Notice that there is no need to duplicate the blank record which closes the last subcommand of a command. All the commands have a fixed input structure, which identifies the associated data set (i.e., Surface 100 Comment; Material 1 Comment; Pair 11 Comment). It is not necessary to adopt a progressive numbering of surfaces, materials or pair sets, numbering does not affect memory allocation, which is based only on the number of commands input. This implies that one can define a problem with two contact surfaces whose numbers are 100 and 500, and then define a contact pair with number 123 that uses these surface numbers. Internally, FEAP will define a sequential numbering and assign tag number 100 to the first set and tag number 500 to the second (assuming they are input in this order).

The **CONT** main command has an option string and two numeric data values which are not used for normal purposes, but are very useful in debugging. The command

```
CONTact OFF
```

causes all contact data to be skipped. This option is useful for a preliminary checks on mesh data without contact. It permits the user to avoid deleting the contact data when the example is tested without contact.\(^1\)

Use of

```
CONTact DEBUg #1 #2
```

causes contact to execute in a *debug* mode. Special debug routines perform output of various arrays and each contact routine write its name on a file each time they are called. The two related numbers define respectively the file unit for list of *call* outputs and for array outputs, respectively. Default unit numbers are 99 and 98. Output is found in files named *Cdebug* (unit #1) and *Cdebug0* (unit #2).

Use of

```
CONTact ON
```

\(^1\)Another option is to place a ! before the **CONTact** command.
results in normal execution mode (default mode when the option ON is not specified).

The currently available contact commands are the following:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SURF</td>
<td>Input of contact geometry</td>
</tr>
<tr>
<td>MATE</td>
<td>Input of contact materials</td>
</tr>
<tr>
<td>PAIR</td>
<td>Definition of the contact pairs</td>
</tr>
<tr>
<td>READ</td>
<td>Switch input to another file</td>
</tr>
<tr>
<td>SAVE</td>
<td>Save data on a file</td>
</tr>
<tr>
<td>END</td>
<td>End of contact data input</td>
</tr>
</tbody>
</table>

The structure of each contact command record is:

```
COMMAND # Comment_label
```

where # is a description number, and any comment label is simply placed in the output file.

Command data sets should terminate with a blank record. Each command can contain one or more additional data records. The first record after a command is a type declaration, which has the following structure:

```
type #1 #2 ..... #15
```

Such type declarations describe the main qualifying characteristic for each command, (i.e. the type of element used for the SURF command, the type of material used for the MATE command, the type of contact formulation for the PAIR command.

In particular, the type declaration of the MATE command permits one to define the subprogram to read material data, and the type declaration of the pair command permits one to describe the corresponding contact element. A type declaration does not have a second string description and accepts a maximum of 15 numerical input values. Each specified string description is converted in a numerical value which corresponds to the position within the control array. The values of all numerical parameters input are stored in the type column of the command control table (see section on control tables below). A type declaration can be followed by additional
CHAPTER 1. INTRODUCTION

data records. This data is input by user subroutines, hence the format may vary in each instance depending on how much is input; however a standard format consisting in two strings and up to 14 numerical data is strongly recommended. Such data are stored in suitably allocated arrays, e.g., the material vector. This is the case for the material data record \texttt{FRIC COUL 0.15} in the previous example where the value of the friction coefficient must be stored as a material parameter (i.e., there are 50 values possible for each material set in the \texttt{CM} array).

A feature record contain information that characterizes basic choices in more detail than a type declaration. A feature data permits one to specify certain options available within the same contact element, e.g., the solution method using penalty or Lagrangian multipliers. The structure of a feature record is the following:

\texttt{FEATURE, option, #1, #2, \ldots, \ldots, #14}

Which means every feature has a string variable which describes an option of the feature, and up to 14 numerical values. Also in this case a numerical translation of the feature and option string is performed, and the data stored in a feature column of the command control table (see section on control tables). The number of the column correspond to the number of the feature within the control table (These are set by the order given in the subprogram \texttt{CONTINIT}).

Finally a sub-command declaration can be used to input and store data in the same way that the type declaration does.

Subcommand data is terminated by a blank record. Contact surface input can be performed by using subcommands such as: \texttt{FACET}, \texttt{BLOCK} and/or \texttt{BLEND}. \texttt{FACET} is a subcommand which has no options and no numeric variables on the same records. It causes input of the subsequent data records (i.e., nodal connections for each element). \texttt{BLOCK} and \texttt{BLEND} are a sub-commands which generate automatically nodal connections along an edge whose characteristics are declared in subsequent data records.

Sub-command dependent data records are read in user subroutines hence the input format has no restrictions; however, in this case also we strongly suggest to keep the feature data structure, i.e., two string data items and up to 14 numeric data items.

A programmer has the possibility to list in the database new type declarations, new features and feature options, new sub-command and sub-commands options.
A programmer also has the possibility to add routines to input type declaration and sub-command data. Basic modifications proceed by making appropriate modifications to the subprogram \texttt{CONTINIT}.

However the current capability to input surface geometry can manage with most practical cases. Instead, it is more relevant for programmers to add new material input/computation routines.

\textbf{SURFace descriptions}

The \texttt{SURF} command record has the following type declarations listed:

\begin{verbatim}
  TYPE,element type, # of nodes per element
\end{verbatim}

Options available are shown in Table 1.1 and described as:

1. \texttt{LINE} - Two-dimensional contact element defined on the $x - y$ plane. The number of nodes (two or more) should be specified by the user.

2. \texttt{TRIA} - Three-dimensional triangular contact element with three or more nodes.

3. \texttt{QUAD} - Three-dimensional quadrilateral contact element with four or more nodes.

4. \texttt{BEAM} - Beam contact element with two or more nodes.

5. \texttt{POIN} - Point (nodal) contact element with one node.

6. \texttt{RIGI} - Rigid contact surface with functional form.

Note that the availability of the input routines for the various geometries does not imply the existence of any contact driver to solve a problem (In particular no use of the beam type is available). These options are simply provided to the user to input data in a standard manner and to build the control arrays. We emphasize again that construction of control tables does not imply one will use it! All the input commands simply generate and arrange the data in a suitable way for developing the compute
The possibility to solve a specific problem is checked by verifying what the available contact drivers (i.e., the subprograms \texttt{CELMT01} to \texttt{CELMT20}) can do.

No features are actually listed for the \texttt{SURF} command, instead it has the above cited sub-commands (i.e., \texttt{FACE BLOC} and \texttt{BLEN}), as well as, any additional ones listed in the \texttt{FEAP} user manual. Table 1.2 summarizes the available subcommand \textit{options} for each of the surface input sub-commands.

The \texttt{FACE} subcommand performs input of data as the standard \texttt{ELEM} command in \textit{mesh}; however, there is no material or region associated for the contact case. If an increment different from zero is specified automatic generation of the missing
elements between the current and the next one is performed. Such generation is based on the node number of the first element and on the specified increment. Node numbers of the next element are not involved. The element input these as follows:

\[
\text{FACEt} \\
\text{El.\#}, \text{increment}, N_1, N_2, \ldots, N_N \\
\text{El.\#}, \text{increment}, N_1, N_2, \ldots, N_N
\]

The **BLOC** and **BLEN** commands perform generation of contact element for two dimensional elements of **LINE** type and three dimensional elements of **QUAD** type. The **BLOC** sub-command requires the following data records:

\[
\#\_\text{Block}\_\text{Node} \, x \, y \, z
\]

whereas the **BLEN** command requires a sequence of *super nodes* to describe the surface to be searched. The form of the data is given as

\[
S\_\text{node}\_1 \, S\_\text{node}\_2 \, \ldots
\]

**MATErial descriptions**

The **MATErial** command is used to input contact surface material characteristics. It should be recalled that for simple contact without friction the satisfaction of the non-penetration conditions can be performed *without* any material command defined. In this case contact is treated as a purely geometrical constraint (frictionless contact). In case of frictional contacts the material friction coefficient must be specified as a material parameter. We note that in the case of a penalty method one more parameter is necessary, (i.e., the penalty value). Due to the fact that this is not a material value, but a solution strategy value, it is specified as parameter in the a feature record of the **PAIR** command. The **MATE** commands should be followed by the **TYPE** record. The type declaration has the following structure

\[
\text{material}\_\text{type} \, \#\_\text{of}\_\text{surface}
\]

where the **#\_of\_surface** field take value 1 if the material model is specific to one surface, or 2 if the material model takes into account the characteristics of both the contacting surfaces.
CHAPTER 1. INTRODUCTION

**MATE**

erial types available are:

1. **STAN** - Standard rigid-with-friction material. Material data are specified in following feature-dependent data records. For the material currently available only a Coulomb friction model is available.

   \[ \text{FRIC,COUL, friction coefficient} \]

2. **NLFR** - Nonlinear friction model.

3. **USER** - User specified model.

   It should be noted that the choice to place the input for the friction coefficient on a separate record, declaring the friction model COUL, will permit one to easily add different friction models later.

**PAIR descriptions**

The **PAIR** command collects information from the **SURF** and **MATE** data to complete the data for each contact problem. Moreover some features that pertain to the solution strategy to be employed are specified. All options have a default value, except the solution method (**SOLM**), which requires specification of the method and any values needed (e.g., **PENA** and the value of the penalty parameters). The available features and options are the following:

1. **DETA**: Detection algorithm to check contact status
2. **MATE**: Mechanical Properties to be used for contact stiffness
3. **SOLM**: Solution method
4. **AUGM**: Augmentation
5. **SWIT**: Activate / deactivate a contact stiffness
6. **TOLE**: Specify contact tolerances

The available options for the cited features are given in Table 1.3. It has to be restated that the availability of the listed features does not imply the existence of any contact driver which uses all of them. It is the programmers responsibility to develop specific contact drivers which use specific combinations of the above features.
### Table 1.3: Contact pair FEATURE options

<table>
<thead>
<tr>
<th>Option Number</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OFF</td>
<td>PENA</td>
<td>BASI</td>
<td>OFF</td>
<td>NONE</td>
<td></td>
<td>INF</td>
</tr>
<tr>
<td>2</td>
<td>ON</td>
<td>LAGM</td>
<td>SEMI</td>
<td>BASI</td>
<td>PENE</td>
<td></td>
<td>STRE</td>
</tr>
<tr>
<td>3</td>
<td>TIMF</td>
<td>CROC</td>
<td>RIGI</td>
<td>HSET</td>
<td>OPEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>CONS</td>
<td></td>
<td>LISE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>SHAK</td>
<td></td>
<td>SMAU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>RATT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Other command descriptions

All other commands READ, SAVE, END are executed by calling existing subroutine of the MESH section, hence they are properly described in the FEAP Manual.

### 1.3 Description of subprogram structure

The contact algorithm structure is modular. The FEAP system connections to data are limited only to a contact switch (CSW). All the connections are performed by calling the same routine with a proper value of the switch. The main routine then performs a set of calls to a contact driver routine or to other FEAP subprograms in order to satisfy the input request. In case of data exchange with the rest of the program the contact driver routine retrieves the necessary arrays. There is no direct data exchange through the parameters of the call. Some data is exchanged by accessing FEAP common blocks. The main contact driver routine is called each time the element library is called. For a solution step there are two calls: (a) One just before the finite element array (residual and tangent) computations; and (b) the second just after. These entries are characterized by the contact switch CSW value which takes a value equal to the continuum element switch ISW for the second call (i.e., after the call to the finite element library), and the same value as ISW plus 100 (i.e., CSW = ISW + 100) for calls just before the finite element library call. Moreover there are direct and special calls identified by the switch values CSW = 200-299, 300-399, 400-499. Tables 1.4 and 1.5 show all currently defined values, and
the correspondent action performed.

The following list provides a brief description of the contact subroutines.

1. Data input - CSW=1

   (a) **SKIPCONT** Skip contact input data if contact is non-active.

   (b) **CONTINIT** Initialize input dictionary for commands, type definitions, features, feature options, sub-commands, sub-command options, set dimensions of command control tables.

   (c) **PNUMC** Determine the number of surfaces, materials, and pairs.

   (d) **COMCONTAB** Set up dimensions of contact command control tables and the length of the array requested to store them.

   (e) **PALLOC** Allocate memory for command control tables (C0). Allocate memory for the material data vector (CM). Allocate memory for the nodal connections data (ICS). At this stage the number of nodes to be stored is not known.

   (f) **PCONT** Main driver routine of the input phase. All the input commands are filtered here.

   (g) **PALLOC** Extend memory area for nodal connection vector, allocate memory for the history variable management correspondence array (HIC).

   (h) **DEFAULTP** Set default of all non-explicitly declared options for the contact pair.

   (i) **CONTLIB** Switch to the requested contact element to perform the initialization phase.

   (j) **STOHMAN** Store history management correspondence vector.

   (k) **PALLOC** Allocate memory for the contact history variables (vector CH). This vector is then fragmented in three vectors, CH1, CH2, CH3, which correspond to the continuum element vectors H1, H2 and H3, respectively.

The listed subroutines call the following second, third and fourth level routines:

The following call structure is the simplest one, because it requires a direct call to the contact driver with the appropriate contact switch value. The contact
<table>
<thead>
<tr>
<th>CSW</th>
<th>A</th>
<th>CCW</th>
<th>A</th>
<th>CSW</th>
<th>A</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>100</td>
<td>-</td>
<td>200</td>
<td>x</td>
<td>Show element information</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>101</td>
<td>-</td>
<td></td>
<td></td>
<td>Input of data</td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td>102</td>
<td>-</td>
<td></td>
<td></td>
<td>Check of data</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>103</td>
<td>x</td>
<td></td>
<td></td>
<td>Form stiffness / check geometry</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>104</td>
<td>-</td>
<td>204</td>
<td>2</td>
<td>Print contact status</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>105</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>106</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>107</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>108</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>109</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>110</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>111</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>112</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>113</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>114</td>
<td>-</td>
<td></td>
<td></td>
<td>Initialize history variables</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>115</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>116</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>117</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>118</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>119</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Profile maximization</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(obsolete)</td>
</tr>
<tr>
<td>301</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Time step update</td>
</tr>
<tr>
<td>302</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Back-up to the beginning of the step</td>
</tr>
<tr>
<td>403</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reset profile for active contacts</td>
</tr>
</tbody>
</table>

Table 1.4: Contact call actions based on CSW values
driver (user developed) can then perform the requested action locally or can call other routines (see also the description of the node-to-segment contact driver). This structure is used to satisfy request for data check (CSW=2); Compute stiffness and residuum (CSW=3); Initialize data at the start (CSW=14); print contact status (CSW=204); profile maximization (CSW=300).

1. For the values: CSW=2, 3, 14, 204, 300 The listed subroutines call the following second level routines:

(a) CDRIVLIB Contact driver library.
   i. SETCOMP Load on commons contact pair data for the current pair
   ii. CDRIV# Contact driver required by the problem described in the PAIR features

(b) The following call structure is used to check active contact and, compute geometrical variables and determine the new shape of the stiffness matrix.
   i. For CSW=103:
      A. CDRIVLIB Contact driver library for geometry check
      B. RSTPRF Reset profile for continuum discretization

<table>
<thead>
<tr>
<th>CSW Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0&lt;=CSW&lt;=20</td>
<td>Call from FORMFE after continuum elements to perform an equivalent action</td>
</tr>
<tr>
<td>100&lt;=CSW&lt;=120</td>
<td>Call from FORMFE before continuum element call to perform special action</td>
</tr>
<tr>
<td>200&lt;=CSW</td>
<td>Direct call outside FORMFE to perform an equivalent action</td>
</tr>
<tr>
<td>300&lt;=CSW</td>
<td>Call for element non–standard calls</td>
</tr>
<tr>
<td>400&lt;=CSW</td>
<td>Special internal calls</td>
</tr>
<tr>
<td>x</td>
<td>Action performed in a proper section</td>
</tr>
<tr>
<td>#</td>
<td>Action performed in section #</td>
</tr>
<tr>
<td>-</td>
<td>Not allowed —return with no warning</td>
</tr>
<tr>
<td>I</td>
<td>Internal call not from CONTLIB</td>
</tr>
<tr>
<td></td>
<td>Action still not defined—return with no warning</td>
</tr>
</tbody>
</table>

Table 1.5: Contact call actions based on CSW values
### CHAPTER 1. INTRODUCTION

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKIPCDAT</td>
<td>Skip input data between contact commands.</td>
</tr>
<tr>
<td>CRSURF</td>
<td>Driver to read and print surfaces data.</td>
</tr>
<tr>
<td>CRMATE</td>
<td>Driver to read and print material data.</td>
</tr>
<tr>
<td>CRPAIR</td>
<td>Driver to read and print pair data.</td>
</tr>
<tr>
<td>READFL</td>
<td>Switch input data reading to another file.</td>
</tr>
<tr>
<td>SAVEFL</td>
<td>Save data on a file.</td>
</tr>
<tr>
<td>CUNU1</td>
<td>Unused contact command # 1.</td>
</tr>
<tr>
<td>CUNU2</td>
<td>Unused contact command # 2.</td>
</tr>
<tr>
<td>CUNU3</td>
<td>Unused contact command # 3.</td>
</tr>
<tr>
<td>CRTYPE</td>
<td>Input subroutine for reading type declarations.</td>
</tr>
<tr>
<td>CRDATA</td>
<td>Input subroutine to read features and sub-commands.</td>
</tr>
<tr>
<td>CRELO1</td>
<td>Read surface element connections generated by the FACE sub-command</td>
</tr>
<tr>
<td>CRELO2</td>
<td>Read surface element connections generated by the BLOC sub-command</td>
</tr>
<tr>
<td>CRBLOK</td>
<td>Perform automatic generation of the BLOC command</td>
</tr>
<tr>
<td>CRMAT01</td>
<td>Read material data requested by the type declaration, for simple no-material with Coulomb friction</td>
</tr>
<tr>
<td>CRMAT02</td>
<td>Unused subroutine available for a new material</td>
</tr>
<tr>
<td>CUMATER</td>
<td>Unused subroutine available for a new material</td>
</tr>
<tr>
<td>CRSURF</td>
<td>Unused routine to read type declaration data or sub-command data.</td>
</tr>
<tr>
<td>SETCOMP</td>
<td>Load on commons contact pair data for the current pair</td>
</tr>
<tr>
<td>CELMT#</td>
<td>Contact driver routine. Equivalent to the standard element routine ELMT#</td>
</tr>
<tr>
<td>ACTIVE</td>
<td>Function that performs variable activation and definition of the number of sets required for the pair.</td>
</tr>
</tbody>
</table>

C. **CDRIVLIB**: Internal call (CSW=403) to reset profile for contact

D. **NWPROF**: Set new pointers for the profile

(c) The following structure is used to show element information. In this case all the available element are scanned to check their properties.

i. For: **CSW=200**

   A. **CDRIVLIB**: Contact driver library for geometry check
CHAPTER 1. INTRODUCTION

B. RSTPRF Reset profile for continuum discretization
Also in this case the listed subroutines call the same second level routines of the previous case.

(d) The next structure is called to perform time step updates
i. For CSW=301
   A. CRESHIS Perform dump of the history vector CH2 on to CH1
      No higher level subroutines are called.

(e) The next structure is called to perform time step update
i. For CSW=302
   A. CRESHIS Perform copy back of the history vector CH1 on to CH2.
      No higher level subroutines are called.

All the other still undefined or not allowed entries are processed in silent mode.

1.3.1 Sizing of arrays

The limits on storage of various data arrays in the contact elements is set in the include file C_0.H. The file is given as

```plaintext
! CONTACT PARAMETERS

integer c_ncc,c_ncs,c_ncel,c_lp1,c_lp3,c_lmv

parameter (c_ncc=10) ! # available contact commands
parameter (c_ncs=200) ! # available command strings
parameter (c_ncel=22) ! # available contact elements
parameter (c_lp1 = 200) ! # available history variables
                   !   for vectors CH1 & CH2
parameter (c_lp3 = 100) ! # available history variables
                   !   for vectors CH3
parameter (c_lmv = 50) ! # available material parameters
```

Generally, this file must be included in any file which contains contact common files (i.e., any include file which has name C_xxxx.H.)
1.3.2 Contact command control table

For each contact command used in the input file a control table is built up. Such table permits to store all the options associated to the command. It permits also to deposit memory offsets or other values specifically related to the command itself. In case some options are not specified in input, default values are assigned.

This control table is a matrix here all the descriptions for input or default data are stored. All the control tables have the same number of rows, currently set to 16. This corresponds to the maximum number of variables which may be assigned to data record. The number of columns depends on the number of *features* defined for the command, plus the number of *user* defined columns, plus the number of *system* defined columns, plus a *type* declaration column. The number of rows is the same for all the table, and the size of each control table is hence defined by:

1. **Feature columns**: There is one columns for each assigned feature. The number of features is assigned in subprogram CONTINIT.

2. **User extra columns**: These columns are available for the user to store user values related to that specific command. The number of user extra-columns should be set in the initialization routine CONTINIT, the default value is zero.

3. **System extra columns**: These columns are used by the contact skeleton to store pointers or other global values. They have been set for each table and should not be changed. Generally, the system columns are assigned to negative column indices in each control table and are not passed to the contact driver routine.

4. **Type declaration column**: This column is similar to the feature columns, and stores the type data. This data is assigned to column zero in each table.

All the values which define the size of each control table are grouped in the subroutine CONTINIT, and can be easily modified.

The number of control tables depends on the number of commands input to describe the contact problem. One pair control table is defined for each command PAIR appearing in the input data and one surface control table is constructed for each SURFace command appearing. All control tables are assigned to the array C0 allocated by the subprogram PALLOC. Tables are stored by contact command order, and then tables related to the same commands are sorted by number.
### Table 1.6: Program set contact pair control array - CP0

<table>
<thead>
<tr>
<th>Command Number</th>
<th>System Type -1</th>
<th>Type 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pair No.</td>
<td>ELMT No.</td>
</tr>
<tr>
<td>1</td>
<td>$h_{1_{offset}}$</td>
<td>$S_1$</td>
</tr>
<tr>
<td>2</td>
<td>$h_{3_{offset}}$</td>
<td>$S_2$</td>
</tr>
<tr>
<td>3</td>
<td>$l_{h_1}$</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>$l_{h_3}$</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>nset</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>nsurf1</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>nsurf2</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>nmat1</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>nmat2</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>nacte</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>genf</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>ncdim</td>
<td>-</td>
</tr>
</tbody>
</table>

### Table 1.7: User set contact pair control array - CP0

<table>
<thead>
<tr>
<th>Command Number</th>
<th>Features 1</th>
<th>Features 2</th>
<th>Features 3</th>
<th>Features 4</th>
<th>Features 5</th>
<th>Features 6</th>
<th>Features 7</th>
<th>User 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SWIT</td>
<td>SOLM</td>
<td>DETA</td>
<td>MATE</td>
<td>AUGM</td>
<td>TOLE</td>
<td>ADHE</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Norm.</td>
<td>$K_n$</td>
<td>-</td>
<td>$M_1$</td>
<td>-</td>
<td>tlpen</td>
<td>$\sigma_{ad}$</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>Tang.</td>
<td>$K_t$</td>
<td>-</td>
<td>$M_2$</td>
<td>-</td>
<td>tlopn</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>Ther.</td>
<td>$K_h$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>tlopt</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
### Table 1.8: Variable names set in contact pair table

<table>
<thead>
<tr>
<th>Option Number</th>
<th>Features</th>
<th>Number</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>npair</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ofh1</td>
<td>0</td>
<td></td>
<td>ndrv</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ofh2</td>
<td>1</td>
<td></td>
<td></td>
<td>ifsolm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>lh1</td>
<td>2</td>
<td></td>
<td></td>
<td>ifdeta</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>lh2</td>
<td>3</td>
<td></td>
<td></td>
<td>ifaugm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>nset</td>
<td>4</td>
<td></td>
<td></td>
<td>tlipen</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>nsurf1</td>
<td>5</td>
<td></td>
<td></td>
<td>tlopen</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>nsurf2</td>
<td>6</td>
<td></td>
<td></td>
<td>tlouts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>nmat1</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>nmat2</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>nacte</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>cndm</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 1.9: Contact surface control array - CS0

<table>
<thead>
<tr>
<th>System Number</th>
<th>Type 0</th>
<th>Feature 1</th>
<th>User 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. Surf.</td>
<td></td>
<td>TYPE</td>
<td></td>
</tr>
<tr>
<td>s_{offset}</td>
<td></td>
<td>nope</td>
<td></td>
</tr>
<tr>
<td>e_{max}</td>
<td></td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>dnope</td>
<td></td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Table 1.9: Contact surface control array - CS0
Chapter 2

Contact driver: The CELMTnn subprogram

All the connection with the FEAP program takes place through the main contact subroutine CONTACT. The subroutine receives only the contact switch value for CSW, and then performs the requested activity switching to the input routines, or to the contact elements library routine CONTLIB which calls the appropriate contact driver routine (e.g., a routine between CELMT01 and CELMT20). A typical structure for a contact driver routine is given by:

```fortran
subroutine celmt01 (ndm,ndf,x,u,
& csw,npair,cs01,cs02,cm01,cm02,cp0,
& ix1,ix2,cm1,cm2,ch1,ch2,ch3,ww1,ww3)
```

```plaintext
c-----[--.--.----+----.----+----.----+----.----+----.----+----.----]
c Inputs :
c ndm - Space dimension of mesh
c ndf - Number dof/node
c x(*) - Nodal coordinates
c u(*) - Current nodal solution vectors
c csw - Contact switch
c npair - # of current pair
c cs01(*) - Contact surface control data for surface 1
c cs02(*) - Contact surface control data for surface 2
c cm01(*) - Contact material control data for surface 1
```
CHAPTER 2. CONTACT DRIVER: THE CELMTNN SUBPROGRAM

cm02(*) - Contact material control data for surface 2
cp0(*) - Contact pair control data
ix1(*) - Element nodal connection list for surface 1
ix2(*) - Element nodal connection list for surface 2
cm1(*) - Contact materials data storage for surface 1
cm2(*) - Contact materials data storage for surface 2

Outputs:
ch1(*) - Contact history variables (old)
ch2(*) - Contact history variables (current)
ch3(*) - Contact history variables (static)
ww1(*) - Dictionary of variables for CH1 & CH2
ww3(*) - Dictionary of variables for CH3

implicit none

include 'c_0.h', 'c_comnd.h', 'c_contac.h', 'c_geom.h',
include 'c_keyh.h', 'c_mate.h', 'c_pair.h', 'c_tole.h'
include 'iofile.h', 'print.h'

character ww1(*)*(*), ww3(*)*(*)
integer ndm, ndf, csw, npair, ix1(dnope1,*), ix2(dnope2,*)
real*8 cs01(nr0,n0c1:*), cs02(nr0,n0c1:*), cm01(nr0,n0c2:*)
real*8 cm02(nr0,n0c2:*), cp0(nr0,n0c3:*), cm1(*), cm2(*)
real*8 ch1(lh1,*), ch2(lh1,*), ch3(lh3,*), x(ndm,*), u(ndf,*)

call cdebug0 (', celmt01', csw) ! Outputs debug data

if ((csw.eq. 1) then ! Initialize assign history
elseif ((csw.eq. 3) then ! Compute tangent and residual
elseif ((csw.eq. 103) then ! Compute contact geometry
elseif ((csw.eq. 14) then ! Initialize history data
elseif ((csw.eq. 400) then ! Start new problem
  once = .true.
endif

end
CHAPTER 2. CONTACT DRIVER: THE CELMTNN SUBPROGRAM

The first few arguments in the driver subprogram CELMTnn are values associated with the finite element model. Thus, NDM and NDF are the space dimension of the mesh and the (maximum) number of degrees of freedom associated with each node, respectively; X is the array of nodal coordinates (type REAL*8) dimensioned X(NDM,*); and U is the array of solution parameters (REAL*8) dimensioned U(NDF,*). The second and third columns contain incremental values; however, the geometric aspects of contact normally require only the solution parameters to construct current coordinates. Using these two arrays, the current position of a node NN may be computed as

\[
\text{DO } I = 1, \text{NDM} \\
\quad x_{\text{cur}}(I) = X(I,NN) + U(I,NN) \\
\text{END DO} \quad \text{! I}
\]

where it is assumed that \( u_i, i = 1, ndm \) contains displacements in the direction of \( x_i \).

The next two arguments on CELMTnn are the contact switch parameter CSW and the pair number being processed, NPAIR, both are of type INTEGER. The NPAIR parameter is used only for output and thus is not usually needed during any computation phase.

2.1 Control data tables

The arguments CS01 and CS02 provide the values in the surface control data tables for surface number 1 (the first surface number on the PAIR command) and surface number 2 (the second surface number on the PAIR command, respectively. These tables are dimensioned

\[
\text{REAL*8} \quad \text{CS01(NR0,N0C1:*),CS02(NR0,N0C1:*)}
\]

The number of rows in each array is NR0 and is currently set to 16. The column numbers define the feature and user columns and N0C1 is currently set to 1. Based on the problem input records the data in these arrays is assigned as described in Tables 1.2 and 1.9.
Similarly, the material control data is passed through arguments CM01 and CM02. These are dimensioned

\[
\text{REAL}\ast 8 \quad \text{CM01}(NR0,NOC2:*) , \text{CM02}(NR0,NOC2:*)
\]

These also describe the feature and user data starting with NOC2 (currently set to 1).

Finally, the pair control data is passed as the argument CP0 and is dimensioned

\[
\text{REAL}\ast 8 \quad \text{CP0}(NR0,NOC3:*)
\]

in which NOC3 is set to 1. The data is stored as described in Tables 1.3, 1.6 and 1.7.

### 2.2 Pair data: Surface arrays

The nodal connection lists for surface 1 and surface 2 are passed through the arguments IX1 and IX2, respectively. These are dimensioned

\[
\text{INTEGER} \quad \text{IX1}(DNOPE1,*), \text{IX2}(DNOPE2,*)
\]

in which DNOPE1 and DNOPE2 are defined in common block C\_GEOM as described in Table 2.3. The actual number of nodes attached to each connection array may differ from the dimension and are given by the parameters NOPE1 and NOPE2, also passed through common C\_GEOM. The main difference is for the LINE option where there are two added columns to assist locating the geometric point of contact. A typical array (e.g., IX1) then has the form

\[
\begin{array}{ccccc}
\text{NODE}_1 & \text{NODE}_2 & \text{ELMT}_1 & \text{ELMT}_2 \\
\text{IX1}(1,* ) & \text{IX1}(2,* ) & \text{IX1}(3,* ) & \text{IX1}(4,* )
\end{array}
\]

in which \text{NODE}_1 and \text{NODE}_2 are the facet global node numbers and \text{ELMT}_1 is the facet number adjacent to the current facet (before) and \text{ELMT}_2 is the facet number which is adjacent (after). A zero \text{ELMT}_1 or \text{ELMT}_2 define the ends of the surface (note there can be only one \text{ELMT}_1 and one \text{ELMT}_2 defining end points on any one
surface – i.e., the surface must be connected). Using this scheme it is easy to locate an adjacent element when the contact node slides from one facet to an adjacent one. A simple example for a search which starts at the first segment and continues to the last is given in the code fragment of Fig. 2.1.

```fortran
  c Locate start segment

  do nel1 = 1, neps1
    if(ix1(dnope1-1,nel1).eq.0) then
      ns1 = nel1
      go to 100
    endif
  end do ! nel1

  100 continue
  c Loop over segments (Forward sequence)

  check1 = .true.
  do while (check1)
    ixl(1) = ix1(1,ns1)
    ... Insert rest of code
    ns1 = ix1(dnope1,ns1)
    if(ns1.le.0) then
      check1 = .false.
    endif
  end do ! while check1
```

Figure 2.1: Sequential search for LINE surface

When matching with another surface to find a possible contact pair one may wish to traverse in a reverse sequence. A code fragment for this is given in Fig. 2.2.

Other command options than LINE do not have additional columns (thus, DNOPE1 = NOPE1), and thus, all columns denote potential contact node numbers.

The geometry of the facets described by the surface node connection numbers in arrays IX1 and IX2 are used to find which parts of surface pairs are in contact and which are not. Thus, when CSW = 103 it is necessary to check all the facets on surface IX1 against those on surface IX2 and determine, using whatever contact strategy
C Locate start segment

   do nel2 = 1, neps2
       if(ix2(dnope2,nel2).eq.0) then
           ns2 = nel2
           go to 200
       endif
   end do ! nel2
   200   no2 = ns2

C Loop over segments (Reverse order)

   ns2   = no2 ! Can use to restart at 'no2'
   check2 = .true.
   do while (check2)
       ixl(...) = ix2(1,ns2)
       ... Insert rest of code
       ns2 = ix2(dnope2-1,ns2)
       if(ns2.le.0) then
           check2 = .false.
       endif
   end do ! while check2

Figure 2.2: Reverse search for LINE surface

is being considered, whether a contact state exists or not. This aspect is quite different from coding of standard finite elements for FEAP and why it is necessary to have a special module to carry out contact. Currently implement algorithms use either a node to node algorithm (NtoN or PtoP option on the PAIR command) or a node to surface algorithm (NtoS option on the PAIR command. Thus, for any other strategy it is necessary for users to construct their own module (i.e., the contact driver routine CELMTnn). Indeed, users may find better strategies for even the node to node or node to surface algorithms currently in the program.
2.3 Material data

For material models which have parameters to describe their behavior the contact driver passes the arrays CM1 and CM2 for the surface pairs 1 and 2, respectively. These arrays are dimensioned

\[
\text{REAL*8} \quad \text{CM1}(*) , \text{CM2}(*)
\]

and, thus, it is evident they apply to all facet pairs of the current contact surfaces.

2.4 History data management and assignment

The arrays which contain any history data are passed through the arguments CH1 (for data defined at \( t_n \)), CH2 (for data described at \( t_{n+1} \)) and CH3 for data independent of time. In addition two character arrays W1 and W3 are passed to facilitate the assignment of specific data items to each of the history arrays. The W1 and W2 arrays are dimensioned as

\[
\text{CHARACTER} \quad \text{W1}(*)(*) , \text{W3}(*)(*)
\]

To understand how the CHi data is used, it is necessary to describe in more detail the method used within the contact module to manage this data.

To manage the assignment of the history data depending on what data in actually input, two routines are written which describe all the types of history variables possible and those which are actually active. One subprogram is the define routine and the other the activate routine which will look at the data and make appropriate choices. A typical definition routine, called DEFHV01 here, is given in Table 2.1.

During contact definition (generally when \( \text{CSW} = 1 \) the necessary parameters to perform a contact analysis are activated using a set of calls to ACTIVE. These may be placed in an activation routine, called ACTHV01 here, as shown in Table 2.2. With this structure it is possible to have just the number of history variables needed to solve each specific problem. There are a number of parameters which are set automatically depending on the contact data provided as input. A list of these parameters is given in Table 2.3.
subroutine defhv01 (ww1,ww3)

c----[-----------------+-----------------+-----------------+-----------------]--------------------
c Outputs:
c   ww1(*) - Dictionary of variables for CH1 & CH2
c   ww3(*) - Dictionary of variables for CH3

c----[-----------------+-----------------+-----------------+-----------------]--------------------
implicit none

classer ww1(*)*8,ww3(*)*8

call cdebug0 (' defhv01',0)

c CH1 & CH2 VARIABLES (dynamic, CH2 copied in CH1)

   ww1(1) = 'masts' ! Master segment number
   ww1(2) = 'istgn' ! Contact normal state indicator
   ww1(3) = 'istgt' ! Contact friction state indicator
   ww1(4) = 'gapn' ! Contact normal gap
   ww1(5) = 'gapt' ! Contact tangential slip
   ww1(6) = 'fn'  ! Normal contact force
   ww1(7) = 'ft'  ! Tangential contact force

c CH3 VARIABLES (static, never automatically modified)

   ww3(1) = 'area'  ! Area of contact surface

end

Table 2.1: Definition of history variables

As noted above, the history variables for each contact pair are passed through the argument list of the contact driver subprogram (CELMTnn) as CH1 (data at time $t_n$), CH2 (data at time $t_{n+1}$) and CH3 (data not changing with time). The arrays CH1, CH2 and CH3 are dimensioned in the driver as:

REAL*8 CH1(LH1,*), CH2(LH1,*), CH3(LH3,*)
subroutine acthv01 (nset)

c-----[---.-------.--------.--------.--------.--------.--------.-----]

c Inputs :

c nset - # of history set required for contact pair

Table 2.2: Activation of history variables

where LH1 is the number of variables assigned to each contact element for the CH1 and CH2 arrays (this is the number allocated by the subprogram ACTHVnn as W1(j) items) and LH3 is the same for the items named W3(j). Note that all history variables
Table 2.3: Parameters for use in contact driver programs

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>COMMON BLOCK</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iffric</td>
<td>Int</td>
<td>C_PAIR</td>
<td>0 = Frictionless; 1 = Friction</td>
</tr>
<tr>
<td>ifsolm</td>
<td>Int</td>
<td>C_PAIR</td>
<td>1 = PENA; 2 = LAGM; 3 = CROC; 4 = CONS</td>
</tr>
<tr>
<td>ifdeta</td>
<td>Int</td>
<td>C_PAIR</td>
<td>1 = BASI; 2 = SEMI; 3 = RIGI</td>
</tr>
<tr>
<td>ifaugm</td>
<td>Int</td>
<td>C_PAIR</td>
<td>0, 1 = OFF; 2 = BASI; 3 = HSET; 4 = LISE; 5 = SMAU</td>
</tr>
<tr>
<td>ifadhe</td>
<td>Int</td>
<td>C_PAIR</td>
<td>1 = INFI; 2 = STRE</td>
</tr>
<tr>
<td>tlipen</td>
<td>Real</td>
<td>C_TOLE</td>
<td>Tolerance for initial penetration</td>
</tr>
<tr>
<td>tlopen</td>
<td>Real</td>
<td>C_TOLE</td>
<td>Tolerance for opening gap</td>
</tr>
<tr>
<td>tlouts</td>
<td>Real</td>
<td>C_TOLE</td>
<td>Tolerance for out of segment</td>
</tr>
<tr>
<td>neps1</td>
<td>Int</td>
<td>C_GEOM</td>
<td>Number of elements on surface 1</td>
</tr>
<tr>
<td>neps2</td>
<td>Int</td>
<td>C_GEOM</td>
<td>Number of elements on surface 2</td>
</tr>
<tr>
<td>dnope1</td>
<td>Int</td>
<td>C_GEOM</td>
<td>Dimension for IX1 array</td>
</tr>
<tr>
<td>dnope2</td>
<td>Int</td>
<td>C_GEOM</td>
<td>Dimension for IX2 array</td>
</tr>
<tr>
<td>nope1</td>
<td>Int</td>
<td>C_GEOM</td>
<td>Number nodes/element on surface 1</td>
</tr>
<tr>
<td>nope2</td>
<td>Int</td>
<td>C_GEOM</td>
<td>Number nodes/element on surface 2</td>
</tr>
<tr>
<td>ifsty1</td>
<td>Int</td>
<td>C_GEOM</td>
<td>Surface 1 type: 1 = LINE; 2 = TRIA; 3 = QUAD; 4 = BEAM; 5 = POIN; 6 = RIGI</td>
</tr>
<tr>
<td>ifsty2</td>
<td>Int</td>
<td>C_GEOM</td>
<td>Surface 2 type: 1 = LINE; 2 = TRIA; 3 = QUAD; 4 = BEAM; 5 = POIN; 6 = RIGI</td>
</tr>
<tr>
<td>ifmty1</td>
<td>Int</td>
<td>C_MATE</td>
<td>Surface 1 material type: 1 = STAN; 2 = NLFR; 3 = USER</td>
</tr>
<tr>
<td>ifmty2</td>
<td>Int</td>
<td>C_MATE</td>
<td>Surface 2 material type: 1 = STAN; 2 = NLFR; 3 = USER</td>
</tr>
</tbody>
</table>

are stored as REAL*8 values, thus, as in treatment of history variables in the finite elements, it is necessary to recast any integer values using a statement

\[
II = NINT(CH1(...)\])

Specific data items are found using two pointer arrays named P1(*) for those associated with W1(*) assignments and P3(*) for that of W3(*). For example, to extract
the value of the gap at time $t_n$ for the element number $\text{NELM}$ for the assignment order given in Table 2.1, one uses the statement:

$$\text{NGAP} = \text{CH1}(P1(4), \text{NELM})$$

since the normal gap is defined by $W(4)$. Note that it is not necessary to use the same name as given for the definition, only the same position. Similarly if one wanted to extract the area to be used for the same element one uses the statement

$$\text{AREA} = \text{CH3}(P3(1), \text{NELM})$$

Care must be taken to ensure that the specific variable was activated for the problem at hand (i.e., checks such as given in the activation subprogram described in Table 2.2 should be included). For example to extract the friction force one should use

$$\begin{align*}
\text{IF}(\text{IFFRIC}.\text{EQ}.1) & \text{ THEN} \\
& \text{FT} = \text{CH2}(P3(7), \text{NELM}) \\
\text{ELSE} & \\
& \text{FT} = 0.0d0 \\
\text{ENDIF}
\end{align*}$$

to ensure that correct extraction is made (of course the above may need to be modified if other friction models are described for the $\text{IFFRIC}$ variable).

### 2.5 Options in driver program

Tables 2.4 to 2.6 describe all the direct calls to CELMTnn which currently exist. A user will not need to code all of the options to get a working element (see below for more information on what MUST always be implemented).

The other indirect calls to the contact elements are defined by the CSW values shown in Table 2.7

Remarks:
## Calling Routine

<table>
<thead>
<tr>
<th>Calling Routine</th>
<th>CSW Value</th>
<th>Description of action to be performed in contact driver routine.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMFE</td>
<td>X isw</td>
<td>Perform operation equivalent to isw in FE’s Not called when ‘isw’ is 1 (MATE); 4 (STRE); 5 (MASS); or 7 (Obsolete surface load treatment). N.B. FORMFE is the routine which does all finite element array calculations – it always calls CONTACT (ISW).</td>
</tr>
<tr>
<td>CONTACT</td>
<td>X 0</td>
<td>Called after initialization to allow users to change default pair name ‘celn’ to any character name.</td>
</tr>
<tr>
<td>PCONTR</td>
<td>X 1</td>
<td>Called immediately after ‘CONT’act data input. Perform any sets on ‘input’ parameters; must define all history variables which can ever exist. Must be processed only once, thus it requires a ‘once’ flag to test against. N.B. In fact, like normal input data, FEAP reads the contact data twice: Once to determine how many surfaces, pairs, materials exist (used to define the tables) and Second to store the data in the appropriate allocated arrays.</td>
</tr>
<tr>
<td>PMACR1</td>
<td>X 103</td>
<td>Determine which elements are in contact to adjust matrix storage. First pass: CSW = 103 Second pass: CSW = 403 : ICCOM = 1; NCEN = 0; NUMELC = 0. If NUMELC &lt; 0 after second pass then: Third pass: CSW = 403 : ICCOM = 2;NUMELC = 0. If optimization of profile or Lagrange multiplier then: Fourth pass: CSW = 403 : ICCOM = 3; For sparse solver another pass is required: Last pass: CSW = 403 : ICCOM = 4.</td>
</tr>
<tr>
<td>PMACR5</td>
<td>X 200</td>
<td>Called when command ‘SHOW CONT” given.</td>
</tr>
</tbody>
</table>

Table 2.4: Existing calls to contact drivers (Part 1)
• N.B. The following values of CSW are not processed: CSW = 0, 4, 5, 7. Any others not given above let the element decide: IFCHIS = F; call sets CSW = ISW.

• Any omissions may be checked in file ‘contact.f’ in the /contact/main directory.

• An ‘X’ in the second column indicates that the contact driver will be called with the CSW set to the value indicated. If not, ‘description’ gives CSW the ‘value’ in the ‘call contact (value)’ indicated.

The CSW values which MUST be in the CELMTnn driver:

• CSW = 0 : Change default names ‘cell’ to ‘ce20’ to user defined name. Inset statements below:

```fortran
  include 'umac1'
  logical pcomp
  integer typ
  ...
  if(csw.eq.0) then
    if(pcomp(uct,'celn',4)) then
      uct = 'user_name' ! 4-characters
    endif
  endif
```

where \( n \) is the number of the contact element (i.e., the number after CELMT without the zero) and ‘celn’ is given as ‘cell’ to ‘ce20’ depending on the \( n \) value.

• CSW = 1 : Set variable once to \texttt{.false.} (Note it can be set true at CSW = 400). Define all possible history variables (DEFHVAR).

• CSW = 14 : Initialize any non-zero history variables.

• CSW = 3 : Compute tangent and residual – must finish with call to routine CONSTASS.

• CSW = 6 : Compute residual – must finish with call to routine CONSTASS. (same for CSW = 206).
• CSW = 103 : Do search for active contact elements. For some cases it may be best to do little here and do most in 403. I think you should be able to use the statement structure in file CNTS2D.F to do the search (except for GEOPAR’s). In particular, the routine GLOSCLN and MASTSEG should work for the 3-d case (Note you must then have the same values for the ww1(1), ww1(2) and ww1(3) in your DEFHVAR routine) ch.(p1(1)) is the number of the master facet for the current slave node; ch.(p1(2)) is the number of the closest local node; and ch.(p1(3)) is an indicator on what may be happening near an intersection between two facets (when the search cannot make up its mind which should be used). Generally if the output is not 1 (unity) one should use both facets and do a corner condition (I think!).

• CSW = 304 : Do search to find active contact elements. Same details as for CSW = 103 this form is used when the command sequence is

```plaintext
LOOP, check, no_ck
    CONTACT CHECK
    LOOP, newton, no_nt
        TANG, 1 (or UTAN, 1)
    NEXT, newton
NEXT, check
```

(call contact(304))

instead of just

```plaintext
LOOP, newton, no_nt
    TANG, 1 (or UTAN, 1)  [call contact(103)]
NEXT, newton
```

(which should check state contact each iteration) and generally leads to more robust performance.

When the form for CSW = 304 is coded the flag IFISTGN (located in common C_CONTAC.H must be checked in the CSW = 103 portion. If it is false no contact search should be performed (however, the location of the contact position on the currently active master should be recomputed); if the value is true then the full check should be made. Careful attention to the details in coding these two values of CSW must be taken to ensure good performance overall. (As a side note, the standard features in the three types of contact elements currently in the program do not perform correctly for both algorithms.)
• CSW = 313 : Activate the history variables which INPUT says are needed (ACTHVAR).

• CSW = 403 : Set list of elements which will be active in next solution. Called when CSW = 103, but to work for all solution options (e.g., profile or sparse) the call to MODPROF must be given when CSW = 403.

Optional CSW which may be good to implement:

• 10 : Do augmented update. Check flag: IFAUGM ≠ 0. When true do augmented update on the contact force: $F_n|_{aug} < - F_n|_{aug} + k_n \cdot gap_n$ N.B. When augmenting is done one MUST check on the sign of $F_n$ to determine when contact is made. The value of the $gap_n$ can only be used to check if the gap is really open and no contact has ever occurred. Also, it will be necessary to monitor the $gap_n$ to detect an initial contact (i.e., when $F_n$ is zero the solution will run until the $gap_n$ penetrates and then one introduces the ‘penalty’ solution to prevent further penetration. Once this has happened (i.e., the value of $F_n|_{aug}$ will be zero) the computation of the force $F_n$ will be done as $F_n = F_n|_{aug} + k_n \cdot gap_n$ and then check conditions on the state of contact. One does this because at convergence $gap_n - > 0$ (and may change sign due to roundoff in computing the zero!).

• 204 : May want to output some values for history variables which can be useful for a ‘user’ to know. (or maybe for debugging).

• 305 : Plot of the slideline surfaces. This helps to ensure date has been input. You should be able to use the statements below:

```plaintext
    elseif(csw.eq.305) then
        call c2geoplt(ix1,ix2,2,6) ! 2 = ix1 , 6 = ix2 colors
    elseif(csw.eq.....
```

2.5.1 Lagrange multiplier constraints

One solution option within the PAIR command if LAGM. This option permits the imposition of constraints using a Lagrange multiplier method. For this option to function correctly, users must check the solution flag IFSOLM. Values of the flag for
a penalty solution are set to unity (1) and for a Lagrange multiplier method to two (2).

For a Lagrange multiplier to be properly handled users should have the following options (in addition to or modified from those above):

- In the definition of history variables provisions must be made to store the values of all the Lagrange multipliers in each element. These should be activated when IFSOLM is two (2).

- For CSW = 3: Assembly should be performed according to:

  ```
  if (ifsolm.eq.1) then
      call constass(ixl,ida,nnod,ndof,ilm, 0, 0,size,s,r)
  elseif(ifsolm.eq.2) then
      call constass(ixl,ida,nnod,ndof,ilm,lnod,nlag,size,s,r)
  endif
  ```

  where **IXL** is an array storing the **NNOD** nodal values which are active in the current element; **IAD** is an **NDOF** array defining the degrees of freedom to be assembled; **ILM** is a list of **LNOD** nodes to which Lagrange multipliers are associated (N.B. There is no scheme to associate them to a contact element); and **NLAG** is the number of multipliers at each node (all nodes are assumed to have the same number within the driver elements); **SIZE** is the first dimension of the tangent stiffness array **S**; and **R** is the residual vector.

  A similar assembly scheme must be included for residual calculations computed when CSW = 6 or 206.

- For CSW = 403: The program must include a call to the routines which perform calculation of the profile. These are given by:

  ```
  if (ifsolm.eq.1) then
      call modprof(ixl,ida,nnod,ndof)
  elseif(ifsolm.eq.2) then
      call modprof(ixl,ida,nnod,ndof,ilm,lnod,nlag)
  endif
  ```

  where the parameters are identical to those described for the call to the **CONSTASS** subprogram.
• For $CSW = 314$: Updates of the Lagrange multipliers should be performed using the following call

```fortran
if(contact_active) then
    call getlagm(iln,lnod,nlag,ch2(p1(.),kset))
else
    ch2(p1(.),kset) <-- zero
endif
```

Here the $CH2(p1(.),kset)$ are the history variables for the current time (there must be $LNOD*NLAG$ values available. They should be set to zero whenever the element is inactive.

The actual calculations for all the operations necessary to insert the multiplier equations into the profile are carried out by the main program CONTACT and the subprograms called above. Operations performed by each user are merely the building of the node lists $IXL$ and $ILM$ together with their sizes.
### Table 2.5: Existing calls to contact drivers (Part 2)

<table>
<thead>
<tr>
<th>Calling Routine</th>
<th>CSW Value</th>
<th>Description of action to be performed in contact driver routine.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMACR3</td>
<td>X 203</td>
<td>Executed on command ‘OPTT’ profile. CSW = 103</td>
</tr>
<tr>
<td></td>
<td></td>
<td>First pass: CSW = 103</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Second pass: CSW = 403 : ICCOM = 1; NCEN = 0; NUMELC = 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Third pass: CSW = 403 : ICCOM = 2; NUMELC = 0.</td>
</tr>
<tr>
<td>PMACR1</td>
<td>X 204</td>
<td>Called when ‘STRE,CONT’ given to output any contact ”stress” values.</td>
</tr>
<tr>
<td>PTIMPL</td>
<td>X 206</td>
<td>Called to set ‘cpl(*)’ array values for any contact time history values. Activated by ‘CONT,N1,N2’ after a ‘TPLO’ command.</td>
</tr>
<tr>
<td>PCONTR</td>
<td>300</td>
<td>Start of new problem. Program sets flags IFCT = F; IFDB = F; LAGRM = F.</td>
</tr>
<tr>
<td>AUTBAC</td>
<td>X 301</td>
<td>Set values to start a time step (‘TIME’) This call occurs during an auto time step. Generally not necessary to do anything.</td>
</tr>
<tr>
<td>PMACR2</td>
<td>X 301</td>
<td>Set values to start a time step (‘TIME’) Generally not necessary to do anything.</td>
</tr>
<tr>
<td>AUTBAC</td>
<td>X 302</td>
<td>Reset history data to start values (‘BACK’). Generally not necessary to do anything.</td>
</tr>
<tr>
<td>OUTARY</td>
<td>303</td>
<td>Called to ‘dump’ values to screen on a ‘SHOW XX: XX = C0, CH, CM, ICS, HIC’.</td>
</tr>
<tr>
<td>PMACR3</td>
<td>X 304</td>
<td>Called on ‘CONT,CHEC’k or ‘CONT,NOCH’eck solution command. IF ‘CHEC’ flag set to IFCHIST = T; if ‘NOCH’ set to F before call to contact element driver.</td>
</tr>
<tr>
<td>PPLTF</td>
<td>X 305</td>
<td>Called on: PLOT,PAIR,k1,k2,k3 command.</td>
</tr>
<tr>
<td>RESTRT</td>
<td>306</td>
<td>Called on ‘REST’art solution command after all data has been read.</td>
</tr>
<tr>
<td>RESTRT</td>
<td>307</td>
<td>Called on ‘SAVE’ solution command after all data has been written.</td>
</tr>
<tr>
<td>PPLTF</td>
<td>X 308</td>
<td>Called on: PLOT,CVAR,k1,k2,k3 command. First pass: CSW = 308: Determine min/max. Second pass CSW = 408. Do plot.</td>
</tr>
</tbody>
</table>
### Table 2.6: Existing calls to contact drivers (Part 3)

<table>
<thead>
<tr>
<th>Calling Routine</th>
<th>CSW Value</th>
<th>Description of action to be performed in contact driver routine.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMACR3</td>
<td>309</td>
<td>Called on ‘CONT,ON’ or ‘CONT,OFF’ solution command. For ‘ON’ set flag IFCT = T; if ‘OFF’ set flag F.</td>
</tr>
<tr>
<td>PMACR3</td>
<td>X 310</td>
<td>Called on ‘CONT,PENA’ or ‘CONT,FRIC’ solution command. Command is: CONT XXXX N1 V1 V2. IF XXXX = PENA, N1 is pair number, CVALUE(1) = V1; CVALUE(2) = V2. IF XXXX = FRIC: Flags set: IFFRON = T; and IFCHIST = F. IF XXXX = NOFR: Flags set: IFFRON = F; and IFCHIST = F.</td>
</tr>
<tr>
<td>PCONTR</td>
<td>312</td>
<td>Called after a ‘TIE’ to reset any eliminated node numbers on the contact facet data.</td>
</tr>
<tr>
<td>UPDATE</td>
<td>X 314</td>
<td>Perform any updates on history data. Called after a ‘SOLV’, ‘TANG,,1’ or ‘UTAN,,1’. Use for updates on any contact solution variables. For example, Lagrange multipliers.</td>
</tr>
</tbody>
</table>

### Table 2.7: Indirect calls to contact drivers

<table>
<thead>
<tr>
<th>Calling Routine</th>
<th>CSW Value</th>
<th>Description of action to be performed in contact driver routine.</th>
</tr>
</thead>
<tbody>
<tr>
<td>X 400</td>
<td>From CSW = 313. Used to avoid mult calls.</td>
<td></td>
</tr>
<tr>
<td>X 403</td>
<td>From CSW = 103. Users to determine active equations and make a call to ‘modprof’ or ‘modprofl’.</td>
<td></td>
</tr>
<tr>
<td>408</td>
<td>From CSW = 308. Do actual plotting for hist variables.</td>
<td></td>
</tr>
</tbody>
</table>
Bibliography


