Interface Between FEAP and INVERSE

(DURING MARIE CURIE PROJECT)

Igor Grešovnik

Ljubljana, January 2004

Contents:

1. SPECIFICATIONS

1.1 Interface Functions

1.1.1 FEAP Functions Used by Inverse

1.1.1.1 void feapanalyse_(int *numparam,double *param, int *numres,double *res,int *numgradres,double *gradres);

Performs a numerical analysis at parameters *param* and writes the results of the analysis to the arrays *res* and *gradres*. Arrays are zero offset (note however that in FORTRAN the index of the first element is 1 rather than 0). On return, *gradres* must eventually contain the gradients of the quantities stored in *res* as a matrix arrangement where each line (due to FORTRAN convention) contain gradients of each quantity stored in *res* , i.e. derivatives of this quantity with respect to each parameter in *param*.

Drawbacks:
Currently things on FEPA side are hard-coded. See the section at the end of this document.

1.1.2 Inverse Functions Used By FEAP

Currently none.

1.2 Overview

1.3 Specification of Shell Interpreter Interface Functions

1.3.1 feapanalyse *{ param specres specgradres }*

Runs the feap analysis at given parameter and retrieves results from FEAP. Vector argument *param* defines the parameters that are be transferred to FEAP. Vector element specification *specres* specifies the vector into which results are stores, and matrix variable element specification *specgradres* specifies the matrix into which gradients of results are stored (each row corresponding to a given result and each column to derivative with respect to a given parameter). In general, *param* must have at least one component (it is not necessarily a copy of global *parammom*) and *specres* must specify an existent vector element with at least one element.

1.3.2 setmicromin *{ minx <miny> <minz> }*

Sets the required minimal co-ordinates of the micro mesh to *minx*, *miny* and *minz*. Arguments are scalar values. The function **setmicromax** is used in combination with this function to set the maximum co-ordinates of the micro mesh. After setting minimum and maximum co-ordinates for the micro mesh, the basic periodic cell is translated and stretched before forming the grid of cells for the micro mesh in such a way that the complete micro mesh fits exactly within the frame defined by minimal and maximal co-ordinates. If *minx* and *maxx* are set to identical values (which is the case if the requested minimal and maximal co-ordinates are not set) then the position and size of the micro mesh are not adjusted and follow directly from the original size of the periodic cells and the number of cells in a grid.

By default, all minimal and maximal co-ordinates of the shell are set to 0, which implies that the micro mesh size and position are not adapted and are determined by the original cell mesh.

1.3.3 setmicromax *{ maxx <maxy> <maxz> }*

Sets the prescribed maximal co-ordinates of the cell; This function is a pair to **setmicromin**.

1.3.4 setrin *{ rin }*

Sets the initial radius o the inclusion or hole within a periodic cell to *rin*. Normally, the radius of the inclusion (which is important for mesh transform) is calculated from the cell, but in the case of a hole, for example, this can not be currently done. By calling **setrin** this radius is set manually. To cancel the current setting and switch back to automatic calculation of the radius (which is default), this function must be called with the value 0 for *rin*.

Notes:

The initial radius of the inclusion can only be done if there are two element groups for the cell, one for the inclusion and one for the surrounding material.

In the case that the micro mesh size is adjusted to given pre-scribed limits (by functions **setmicromin** and **setmicromax**) then the radius of the inclusion in the adjusted micro mesh (which is usually a grid of basic cells) must be specified.

1.3.5 setcellparlim *{ <min> <max> <d> <max12> }*

Sets the limits on shape parameter range, which will be imposed by parameter transform. All arguments are optional. All parameters are optional.

Vector value argument *min* determine the lower limits for parameters, *max* determines the upper limits and *d* determines the relative transition intervals of the transform. Components of *d* must be greater than 0 and less than 1/2. The dimensions of all these vectors must be the same, and can be less than the number of parameters (the remaining parameters will just not have limited range imposed). If for any component the lower limit is greater or equal to the upper one then no range limit is imposed on this parameter. If any corresponding component of d is less or equal to 0 then it is automatically set to 0.05 of the allowed range for the corresponding parameter component.

In addition, *max12* can specify the maximum ratio between the first and second parameter and vice versa. It must be greater than 1 in order to take effect. It will only take effect if *min* and *max* are specified and have at least one component (but their components can be equal so the have no effect). The transition interval is set automatically for this transform bound.

Limits on parameter ranges can be switched on by simply calling the function without arguments (by defaults, these limits are not imposed). When this function is called, only those limits and transition intervals will be set that are specified as arguments (i.e. the limits that were set before but are not specified with arguments will be annulled).

When the limits on parameter range are imposed, the transformed parameters are stored in vector variable *transf* immediately after the transform if performed. It is usually useful to print out the transformed parameters because these are actually the parameters with more direct geometrical interpretation.

1.3.6 **readmicrocell** *{ filename <eldefdir> }*

String argument *filename* must define the Elfen data file from which the mesh for periodic cell is read. String argument *elfdir* defines the directory in which the element topology specification file is looked for. If it is omitted then the directory containing the mesh file is taken.

1.3.7 **createmicromesh** *{ which nx ny param }*

Creates a micro mesh by setting up a grid of periodic cell derived from the basic cell. The mesh in the basic cell is first transformed according to the parameters *param* (a vector value argument). A counter value argument *which* specify which transform is to be used (currently only *1* is available). *nx* and *ny* (counter value arguments) are the number of basic cells in both directions.

1.3.8 **feapexportcell** *{ filename }*

Exports the basic cell into a FEAP mesh file named filename (a character value argument). Used mainly for control.

1.3.9 **feapexportmicromesh** *{ filename }*

Exports the micro mesh to a FEAP mesh file named *filename* (a character value argument). This file is read by FEAP and used as the definition of the mesh for each micro element.

1.3.10 **fileplotcell** *{ filename fi theta str param }*

Plots the basic cell into a Tcl file named *filename*. Scalar arguent *fi* and *theta* (defaunlt 30) determine the viewpoint of the plot in degrees, but don’t have any effect for 2D meshes. If string value argument *str* is specified then this string is printed on the top of the plot. Vector value *param* can be specified, in which case the components of the vector are printed on a plot (useful to see which mesh corresponds to which parameters). If both *str* and *param* are not specified then the vector *parammom* is printed on the top of the plot.

All arguments are optional, but must follow in the specified order. If *filename* is an empty string ("") then plot will be exported to a file named *SGS_meshplot#*.tcl* in the current directory, where * is rplaced by a number string. In this case file names are generated uniquely within the same program run so that the files do not overwrite one another. This possibility should be used by care since files are usually quite large (over 100 kB).

The plots generated by this command can be visualized by interpreting the file by the *wish* shell (just type "wish <filename>" in the terminal window). *wish* is a part of the Tcl/Tk package.

1.3.11 fileplotmicromesh *{ filename fi theta str param }*

Similar to **fileplotcell**, except that the micro mesh (i.e. a grid of cells) is plotted.

1.3.12 fileplotcellps *{ filename fi theta str param }*

Similar to **fileplotcell**, except that it plots in *Encapsulated PostSctipt* format (eps).

1.3.13 fileplotmicromeshps *{ filename fi theta str param }*

Similar to **fileplotmicromesh**, except that it plots in *Encapsulated PostSctipt* format (eps).

1.3.14 plotcell *{ fi theta str param winid }*

Similar to **fileplotcell**, except that it plots on the screen. The filename argument therefore falls away. In addition, the *winid* can be used to specify an internal ID of the window used to plot in. If *winid* is not specified or it is 0 or negative then a new window is used everytimie this function is called. If it is greater than 0 then all calls with the same *winid* will plot in the same window.

1.3.15 plotmicromesh *{ fi theta str param winid }*

Similar to **fileplotmicromesh**, except that it plots on the screen. If *winid* is not specified or it is 0 or negative then a new window is used everytimie this function is called. If it is greater than 0 then all calls with the same *winid* will plot in the same window.

1.4 Questions about improvement of the interface

Instead of a hard-coded interface it would be better to have the whole interface controllable by the shell – would this be possible?

Basically only three functions should be implemented for such an interface, which would allow much more flexibility than the current one, and essentially nothing would be hard-coded except the positions and names of interrupts. An existing Elfen interface can serve as a model. It is therefore described below what was done on Elfen’s side of the interface. Actually the interrupt function could be hard-coded on the shell side without l any lost of elegance. The advantage would be that in the future shell could be used in combination by FEAP for solving optimization problems without any intervention to either program’s structure.

1.4.1 Elfen Functions Used by Inverse

Inverse needs utilities for accessing the database (i.e. reading results and changing input) and for running a certain part of analysis. In the Elfen-Inverse interface, Inverse can run the whole analysis at once, but can also interact with Elfen at certain points during the analysis using interrupts.

1.4.1.1 void define_elfendyn_user_interrupt_function (int (*func) (const char * message));

Installs user interrupt function (*func*), i.e. sets the function that is executed at every interrupt in the analysis. In Elfen, interrupt function is run with a string parameter that enables identification of location where the function was called.

1.4.1.2 void * elfendyn_database_entry_new (const char * recname,int recnum,const char *arrayname,char *datatype,int *arraydim);

A general function for accessing analysis database.

Returns a pointer to the requested array (or NULL if pointer is not available).

Recname is record name, *recnum* is record number, and *arrayname* is array name. Function writes specification of the array type in **datatype* and dimensions of the array in *arraydim*. a*rraydim* must point to space allocated for at least 5 integers at function call. After the function call, the first zero element of *arraydim* indicates the end of dimensions.

1.4.1.3 void ELFRUN(void);

Runs Elfen analysis, i.e. reading input data and evaluation part.