TOCHNOG programmers manual - a free explicit/implicit FE program

Dennis Roddeman

February 11, 2001

Contents

1 Implementation of the convection-diffusion equation

In this section we will show how the convection-diffusion equation is implemented. We do this for the one-dimensional equation. See the users manual for the convection-diffusion (heat) equation:

$$\rho C(\dot{T} + \beta \frac{\partial T}{\partial x}) = k \frac{\partial^2 T}{\partial x^2} - aT + f$$

where we take here for simplicity a prescribed temperature T at $x = 0$ and $x = 1$. Multiplication with a weighting function (=test function) h and integration over the volume gives:

$$\int [\rho C(\dot{T} + \beta \frac{\partial T}{\partial x})h]dx = \int [k \frac{\partial^2 T}{\partial x^2} h - aTh + fh]dx$$

where h is 0 at $x = 0$ and $x = 1$ but arbitrary otherwise. Integration by parts of the conductive term, and moving everything to the right gives for the residue R in the equation:

$$R = \int [-\rho C(\dot{T} + \beta \frac{\partial T}{\partial x})h - k \frac{\partial T}{\partial x} \frac{\partial h}{\partial x} - aTh + fh]dx = 0$$

where the boundary terms resulting from the integration by parts disappear because h is 0 at $x = 0$ and $x = 1$. This form of the equilibrium equation is called the 'weak form'.

The weak form above is still perfectly equal to the heat equation. Now, however, we start approximating with the finite element method. The temperature field T is approximated with finite element interpolation functions $T_j(x)$ where j denotes the node number. In the standard galerkin approach, the same functions for the weighting field is used, so h is approximated with $h_i(x)$. Remark 1: we use indices i for the weighting function to prevent conflicting indices of the temperature and weighting functions. Remark 2:h can now be called a virtual temperature field, since it consists of the same set of functions as the temperature field itself. So:

$$T = T(x_j)T_j(x)$$

and:

$$h = h(x_i)h_i(x)$$

where x_j is the position of node j, and summation convention is used. For a two-noded linear element this would be:

$$T = T(0)(1 - x) + T(1)x$$

and:

$$h = h(0)(1 - x) + h(1)x$$

The weighting functions are substituted in the weak form of the equilibrium equation:

$$R = \int [-\rho C(\dot{T} + \beta \frac{\partial T}{\partial x}) h(x_i) h_i(x) - h(x_i) k \frac{\partial T}{\partial x} \frac{\partial h_i(x)}{\partial x} - aT h(x_i) h_i(x) + f h(x_i) h_i(x)] dx = 0$$

Now demanding that this is true for arbitrary nodal weighting function values $h(x_i)$ gives for each i:

$$R = \int [-\rho C(\dot{T} + \beta \frac{\partial T}{\partial x}) h_i(x) - k \frac{\partial T}{\partial x} \frac{\partial h_i(x)}{\partial x} - aT h_i(x) + f h_i(x)] dx = 0$$

The time derivative $\dot{T}$ will be approximated with the Euler backward scheme $\frac{T(t+dt)-T(t)}{dt}$. Integrals are approximated by numerically integrating over a set of integration points. The temperature T in the last expression is understood to be interpolated by the finite element interpolation functions, and so depends on the nodal temperatures $T(x_j)$. Thus the last expression gives a set of equations for the nodal temperatures $T(x_j)$.

We now will discuss where the seperate terms in this expression are implemented in the finite element program. First we link terms from the expression for R with the program source:

mathematical term	program
R	element_rhside
ρ	dens
C	condif_capacity
β	condif_flow
k	condif_conductivity
a	condif_absorption
f	element_volume_force
$T(t+dt)$	new_dof
$T(t)$	old_dof
dt	dtime
i	inol
$h_i(x)$	h[inol]
$\frac{\partial h_i(x)}{\partial x}$	new_d

Different parts of R are implemented in different routines. First, it will be collected in the variable element_rhside. Later, element_rhside will be used to fill node_rhside which contains the residue R for the nodes.

- The term $-\rho C \dot{T} h_i(x)$ is implemented in general.cc. The time derivative $\dot{T}$ is evaluated lumped. The implementation of this term is marked by the comment // inertia (lumped).
- The term $-\rho C \beta \frac{\partial T}{\partial x} h_i(x)$ is implemented in general.cc. The implementation of this term is marked by the comment // convection with respect to mesh.
- The term $-k \frac{\partial T}{\partial x} \frac{\partial h_i(x)}{\partial x}$ is implemented in general.cc. The implementation of this term is marked by the comment // diffusive terms (with partial integration).
- The term $-aT h_i(x)$ is implemented in condif.cc. The implementation of this term is marked by the comment // absorption.
- The term $+f h_i(x)$ is implemented in elem.cc. The implementation of this term is marked by the comment // element force.

2 Variables

Typically use of variables is as follows:

program variable	meaning
element	global element number.
max_element	highest global element number.
inod	global node number.
max_node	highest global node number.
inol	element node number.
nnol	the number of element nodes.
ipuknwn	primary unknown number.
npuknwn	the number of primary unknowns.
iuknwn	unknown number.
nuknwn	the number of unknowns.

3 Routines

The most important program structure is shown below:

tn (main routine)
    input (read input)
        db_initialize (initialize the database)
    top (time step loop)
        elem (element loop, integration point loop)
            general (generic element part)
            condif (convection_diffusion equation)
            materi (materi equation)
                stress (stress evaluation)
                    plasti (plasticity evaluation)
            wave (wave equation)
        solve (solve equations)
        refine_globally (h- and p-refinement)
        refine_locally (local h-refinement)
        print (print outpout)
    tn_exit (exit tochnog)

Below, all files are listed:

- adjust.cc

  Evaluate **control_mesh_adjust_geometry**.

- area.cc

  Evaluate area integrals like **condif_radiation**, **condif_convection**, and **force_element_edge**.

- bounda.cc

  Evaluate boundary conditions like **bounda_unknown**, **bounda_force**, **bounda_time** and **bounda_sine**.

- calcul.cc

  Evaluate **post_calcul**.

- check.cc

  Check validity of data in input file.

- clapack.c

  clapack routines from netlib.

- condif.cc

  Evaluate for type **-condif** whatever is not evaluated in general.cc.

- contact.cc

  Evaluate contact data like **contact_geometry**, etc.

- create.cc

  Create an element or a node.

- damage.cc

  Evaluate damage like **materi_damage_mazars**.

- data.cc

  Evaluate **control_data_delete** and **control_data_put**.

- database.cc

  Specification of properties of items in database and database routines.

- delete.cc

  Delete an element or a node or the elements and nodes in a geometry.

- dof.cc

  Routine for Degrees Of Freedom.

- elasti.cc

  Determine material stiffness matrix from **materi_elasti_young** and **materi_elasti_poisson**.

- elem.cc

  Element loop. Called from top.cc.

- equal.cc

  Evaluate **dof_equal**.

- error.cc

  Evaluate **post_error_item**.

- failure.cc

  Evaluate **materi_strain_plasti_failure**, **materi_strain_total_failure** and **materi_damage_failure**.

- filter.cc

  Evaluate **print_filter**.

- force.cc

  Evaluate **force_element_volume_factor** and other force info.

- general.cc

  Generic part of different types (**condif**, **wave**, **materi**, etc).

- geometry.cc

  Evaluate geometry data items (like **geometry_point**, etc.).

- groundfl.cc

  Evaluate for type **-groundflow** whatever is not evaluated in general.cc.

- group.cc

  Evaluate element group things like **area_element_group** and **dependency_item**.

- hyperela.cc

  Hyperelastic laws **materi_hyper_besseling** and **materi_hyper_mooney_rivlin**.

- initia.cc

  Initialise global variables.

- input.cc

  Read the input file.

- integra.cc

  Integration rules (gauss and lobatto).

- intersec.cc

  Calculate intersections (line with line, line with point, etc.).

- inverse.cc

  Evaluate inverse data like **inverse_parameter_set**, etc.

- locate.cc

  Evaluate **node_locate**, etc.

- macro.cc

  Evaluate **control_mesh_macro**.

- map.cc

  Map element and nodal data from one mesh to another mesh.

- materi.cc

  Evaluate for type **-materi** whatever is not evaluated in general.cc.

- math.cc

  Generic mathematical routines, like matric-vector operations, etc.

- membrane.cc

  Evaluate **materi_membrane**.

- merge.cc

  Evaluate **control_mesh_merge**.

- mesh.cc

  Mesh operation. Update element and node data if a mesh has changed. Add one mesh to another.

- miscel.cc

  Miscelaneous stuff.

- new_mesh.cc

  Evaluate **control_mesh_new_mesh_region**.

- node.cc

  Determine nodes in the direct neighbourhoud of a node.

- plasti.cc

  Materi stress plasticity evaluation.

- point_el.cc

  Determine of a point in which element it is located and what the corresponding weighting-factors of the element's nodes are.

- polynom.cc

  Determine an element interpolation functions, its derivatives, etc.

- post.cc

  Evaluate **post_point**, etc.

- pri.cc

  Print routines for test printing.

- print.cc

  Evaluate **control_print**.

- print_da.cc

  Evaluate **control_print_data_versus_data**.

- print_gm.cc

  Evaluate **control_print_gmv**.

- print_hi.cc

  Evaluate **control_print_history**.

- print_ma.cc

  Evaluate **control_print_matlab**.

- print_pl.cc

  Evaluate **control_print_plotmtv**.

- print_te.cc

  Evaluate **control_print_tecplot**.

- project.cc

  Projections (point on line, etc.).

- range.cc

  Evaluate a range **-ra ... -ra**.

- refine_g.cc

  Evaluate **control_mesh_refine_globally**.

- refine_l.cc

  Evaluate **control_mesh_refine_locally**.

- remesh.cc

  Evaluate **control_mesh_remesh**.

- renumber.cc

  Evaluate **control_mesh_renumber**.

- repeat.cc

  Evaluate **control_repeat**.

- restart.cc

  Evaluate **control_restart**.

- slide.cc

  Evaluate **slide_geometry**, etc.

- solve.cc

  Matrix solver and also eigenvalue calculations.

- split.cc

  Evaluate **control_mesh_split**.

- stress.cc

  Materi stress evaluation. Called from materi.cc.

- sysother.cc

  Routine if not sysposix.cc, syssgi.cc.

- sysposix.cc

  Specially for posix systems.

- syssgi.cc

  Specially for sgi systems.

- tendon.cc

  Evaluate **tendon**.

- tn.cc

  Highest level routine.

- top.cc

  Main loops like inverse iterations, time integration, equilibrium iterations, etc.

- tyings.cc

  Setup and apply tyings.

- umat.c

  umat routine to enable usage of abaqus umat.f routines.

- user.cc

  User supplied routines.

- viscoela.cc

  Evaluate **materi_maxwell_chain**.

- viscosit.cc

  Evaluate **materi_viscosity**.

- volume.cc

  Evaluate **volume_factor** and **volume_element_factor**.

- wave.cc

  Evaluate for type **-wave** whatever is not evaluated in general.cc.

4 Database

The file database.cc contains:

- Arrays with actual data and information arrays.
- Acces routines.

The actual data and information about it is stored in a number of arrays:

- dbl_data This array contains pointers to the actual double data. It has so for all possible data (MDAT), and for all possible versions of the data (MVERSION). (Some data, like element data and nodal data, can exist in more than one version; that is, it can exist on more than one mesh).
- int_data The same as dbl_data but now for integers however.
- max_index The maximum index allocated for the data item.
- no_index Set to 1 if the data item doesn't have an index, like NUMBER_OF_PROCESSORS.
- type INTEGER for integer data, and DOUBLE for double data.
- name The name, e.g. **element** for data item ELEMENT.
- external Set to 1 if it is user data, set to 0 if it is internally for tn only.
- data_length Set to the length of the data item (or the maximum length in the data item doesn't have a fixed length).
- fixed_length Set to 1 if the length of the data item is fixed, set 0 0 if the length of the data item is not a priori fixed.
- data_class The class a data item belongs to. E.g. ELEMENT_GROUP belongs to the class ELEMENT. This allows for operations on a class, like deleting an element and all its properties.
- print_only Set to 1 if the data item is meant for printing after the calculation only, and thus is not allowed to be input in the data file.
- version_all Set to 1 if all versions (meshes) of a data item can exist. E.g. of ELEMENT all versions can exist, but of NUMBER_OF_PROCESSORS only one version can exist.

There are a number of routines to acces the data. The most important routines are listed below:

- db(idat,index,*ival,*dval,&length,version,task) This is the main routine to acces data on the database. The data can either be read or written with this routine.
  - The data item accessed is idat, e.g. idat is ELEMENT, or idat is NODE, etc.
  - The index, for the data item, accessed is index
  - Integer data is accessed via the ival array.
  - Double data is accessed via the dval array.
  - The length of the data item is contained in length. If the data is read, the length is returned. If the data is written, the length should be set.
  - The version of the data should be set in version. This is either VERSION_NORMAL, or VERSION_NEW, etc. See the discussion of versions in a seperate section below.
  - The task to be performed is et in task. If the data is written then set task to PUT. If the data is read then set taks to GET. Some other task options exist.
- ...

5 Versions

Of some data more than one version can exist in the database. This gives the possibility to keep more values for the data items in the database. This is used to store e.g. nodal unknowns at the old time point t, while the new nodal unknowns are being calculated in the current time step. Moreover, it comes handy at mesh refinements, where data for the old coarse mesh and the refined new mesh are avaiable at the same time, etc. Some of the allowed versions are listed below:

version	purpose
VERSION_NORMAL	Normal working version. Keeps unknowns at old time point t.
VERSION_NEW	New mesh. Keeps unknowns at time t+dt.
VERSION_START	Keeps the mesh at the start of the calculation; for restarts.
VERSION_TMP	Temporarily working version (scratch version).
VERSION_PRINT	Temporarily version for printing (some plotters require sequential numbering).
VERSION_MACRO	Temporarily version for control_macro option.
...	...

6 Debugging

Activate debug printing by setting the environment symbol TOCHNOG_DEBUG to **yes**.

Printing can be limited by setting the following environment symbols:

environment symbol	purpose
TOCHNOG_ELEMENT *element_number*	Printing for specific element only.
TOCHNOG_ICONTROL *control_number*	Printing for specific control index only.
TOCHNOG_ITERATION *iteration_number*	Printing for specific iteration only.
TOCHNOG_NODE *node_number*	Printing for specific node only.
TOCHNOG_ROUTINE *routine_name*	Printing for specific routine only.

You also can set the environment variable TOCHNOG_REPORT_ROUTINE. Then tn reports the routines it passes.

After a calculation, the entire database can be printed by using **control_print_database** *index* **-everything**. This prints, besides the normal user variables, also internal variables.