3 THERMAL ANALYSIS

3.1 Introduction

UDEC allows simulation of transient heat conduction in materials, and the development of thermally induced displacements and stresses. This includes the following specific features:

1. Heat transfer is modeled as conduction – either isotropic or anisotropic, depending on the user's choice of material properties.
2. Several different thermal boundary conditions may be imposed.
3. Any of the mechanical block models may be used with the thermal model.
4. Heat sources may be inserted into the material as volume sources. The sources may be made to decay exponentially with time.
5. Both implicit and explicit calculations schemes are available, and the user can switch from one to the other at any time during a run.
6. The thermal analysis provides one-way coupling to the mechanical stress calculation through the thermal expansion coefficient.
7. The thermal analysis provides one-way coupling to the calculation for fluid flow in joints through the temperature dependency of fluid density and joint permeability.

This section contains a description of the thermal formulation (Section 3.2). Recommendations for solving thermal and thermal-mechanical problems are also provided (Section 3.3). The *UDEC* input commands for thermal analysis (Section 3.4), and the system of units for thermal analysis (Section 3.5) are given. Finally, several verification problems (Section 3.6) are described.

*.

Refer to these examples as a guide for creating *UDEC* models for thermal analysis and coupled thermal-stress analysis. See Section 2 for a description of coupled thermal-fluid flow analysis.

---

* The data files in this section are stored in the folder "ITASCA\UDEC700\Datafiles\Thermal" with the extension ".DAT." A project file is also provided for each example. For the *GIIC*, open the project file by clicking on the FILE/OPEN PROJECT menu item and select the project file name (with ".PRJ" extension). Then click on the *Project Options* icon at the top of the *Project Tree Record*, select *Rebuild unsaved states*. For the GUI, open the project file by clicking on the FILE/OPEN PROJECT menu item and select the project file name (with ".UDPRJ" extension). Then click on the *Project* tab and select the "Master.dat" and run it

3.2 Formulation

3.2.1 Basic Equations

The basic equation of conductive heat transfer is Fourier's law, which can be written in one dimension as

$$Q_i = -k_{ij} \frac{\partial T}{\partial x_j} \tag{3.1}$$

where Q_i = flux in the i-direction (W/m^2);
k_{ij} = thermal conductivity tensor (W/m°C); and
T = temperature.

Also, for any mass, the change in temperature can be written as

$$\frac{\partial T}{\partial t} = \frac{Q_{\text{net}}}{C_p\, M} \tag{3.2}$$

where Q_{net} = net heat flow into mass (M);
C_p = specific heat (J/kg°C); and
M = mass (kg).

These two equations are the basis of the thermal version of *UDEC*.

For two-dimensional heat transfer, Eq. (3.2) can be written as

$$\frac{\partial T}{\partial t} = \frac{1}{C_p\, \rho} \left[\frac{\partial Q_x}{\partial x} + \frac{\partial Q_y}{\partial y} \right] \tag{3.3}$$

where ρ is the mass density.

Combining this with Eq. (3.1),

$$\frac{\partial T}{\partial t} = \frac{1}{C_p\, \rho} \frac{\partial}{\partial x} \left[k_x \frac{\partial T}{\partial x} \right] + \frac{\partial}{\partial y} \left[k_y \frac{\partial T}{\partial y} \right] \tag{3.4}$$

$$= \frac{1}{\rho\, C_p} \left[k_x \frac{\partial^2 T}{\partial x^2} + k_y \frac{\partial^2 T}{\partial y^2} \right]$$

if k_x and k_y are constant. This is called the diffusion equation.

Temperature changes cause stress changes according to the equation

$$\Delta\sigma_{ij} = -\delta_{ij}\ 3K^{*}\alpha\ \Delta T \tag{3.5}$$

where $\Delta\sigma_{ij}$ = change in stress ij;
δ_{ij} = Kronecker delta (δ_{ij} = 1 for $i = j$ and 0 for $i \neq j$);
K^{*} = K (for plane strain);
= $6KG/(3K + 4G)$ for plane stress, where K is bulk modulus and G is shear modulus;
α = linear thermal expansion coefficient; and
ΔT = temperature change.

The mechanical changes can also cause temperature changes as energy is dissipated in the system. This effect is neglected because it is usually negligible.

3.2.2 Diffusion Equation – Explicit Algorithm

UDEC discretizes fully deformable blocks into triangular zones which are also used for the thermal analysis.

At each timestep, Eqs. (3.1) and (3.2) are solved numerically, using the following scheme.

1. In each triangle, ($\partial T/\partial x$ and $\partial T/\partial y$) are approximated using the equation

$$\frac{\partial T}{\partial x_i} = \frac{1}{A}\int T n_i\ ds \tag{3.6}$$

$$\cong \frac{1}{A}\sum_{m=1}^{3}\bar{T}^{m}\ \epsilon_{ij}\ \Delta x_j^{m}$$

where A = area of the triangle;
n_i = i^{th} component of outward normal;
$\bar{T}^{m}$ = average temperature on side m;
Δx_j^{m} = difference in x_j between ends of side m; and
ϵ_{ij} = two-dimensional permutation tensor.

$$\epsilon_{ij} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

The heat flow into each gridpoint of the triangle is calculated from

$$F_i = A_j \; Q_i \tag{3.7}$$

where A_j is the width of the line perpendicular to the component Q_i, as shown in Figure 3.1.

$$\begin{aligned} F_{\text{total}} &= F_x + F_y \\ &= A_y \; Q_x + A_x \; Q_y \end{aligned} \tag{3.8}$$

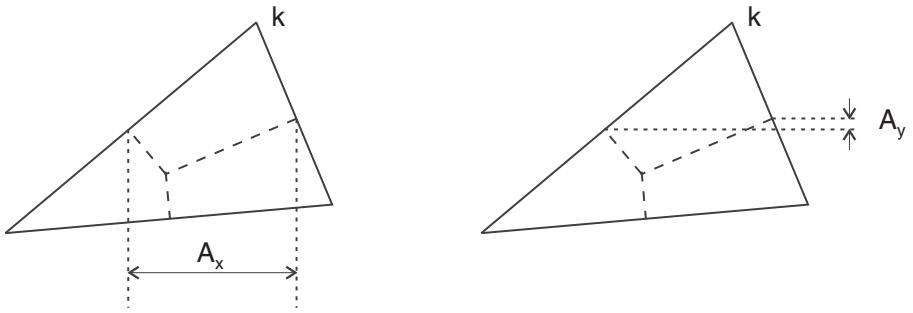

Figure 3.1 Heat flow into gridpoint k

2. For each gridpoint,

$$\Delta T = \frac{Q_{\text{net}}}{C_p \; M} \Delta t \tag{3.9}$$

where Q_{net} is the sum of F_{totals} from all zones affecting gridpoint i.

3.2.3 Stability and Accuracy of the Explicit Scheme

For the explicit scheme, Δt is limited by numerical stability considerations. The critical timestep for stability, assuming Δx = the smallest zone dimension in the model (see, for example, Karlekar and Desmond 1982), is

$$\Delta t \; \leq \; \frac{(\Delta x)^2}{4\kappa \left[1 + \frac{h \Delta x}{2k}\right]} \tag{3.10}$$

where h = the convective heat transfer coefficient; and
κ = the thermal diffusivity ($k/\rho C_p$ for $k_x = k_y = k$).

The accuracy of the explicit solution scheme is determined by the introduction of errors from several sources. A strict definition of error in the explicit formulation is not obtained, simply because error arises from the finite difference approximations used; the error also is affected by the zone discretization and timestep. The explicit solution introduces a mixed order of error in the diffusion equation. This is because a forward difference formulation is used in time (this is first-order accurate), and a central difference formulation is used in spatial coordinates (this is second-order accurate).

3.2.4 Diffusion Equation – Implicit Thermal Logic

The implicit thermal logic in *UDEC* uses the Crank-Nicholson method, and the set of equations is solved by an iterative scheme known as the Jacobi method. An implicit method is advantageous for solving linear problems, such as heat conduction with constant conductivity, because it allows the use of much larger timesteps than those permitted by an explicit method, particularly at later times in a problem, when temperatures are changing slowly.

The usual one-dimensional, explicit finite-difference scheme for heat conduction can be written as

$$\frac{\rho\, C_p}{k} \cdot \frac{T_i(t+\Delta T) - T_i(t)}{\Delta t} = \frac{T_{i+1}(t) - 2\,T_i(t) + T_{i-1}(t)}{(\Delta x)^2} \tag{3.11}$$

An implicit method can be derived by replacing the right-hand side of Eq. (3.11) with the expression

$$\frac{1}{2}\left[\frac{T_{i+1}(t+\Delta t) - 2\,T_i(t+\Delta t) + T_{i-1}(t+\Delta t)}{(\Delta x)^2} + \frac{T_{i-1}(t) - 2\,T_i(t) + T_{i-1}(t)}{(\Delta x)^2}\right]$$

This method, known as the Crank-Nicholson method, has the advantage that it is stable for all values of Δt, but it has the disadvantage of being implicit. This means that the temperature change at any point depends on the temperature change at other points. This can be seen by rewriting the implicit scheme as

$$\frac{\rho\, C_p}{k\ \Delta t}\ \Delta T_i = \left[\frac{T_{i+1} + \frac{1}{2}\,\Delta T_{i+1} - 2\,(T_i + \frac{1}{2}\,\Delta T_i) + T_{i-1} + \frac{1}{2}\,\Delta T_{i-1}}{(\Delta x)^2}\right] \tag{3.12}$$

since $T_k(t+\Delta t) = T_k(t) + \Delta T_k$.

The implicit method requires that a set of equations be solved at each timestep for the values of ΔT_i.

In matrix notation, the *explicit* method can be written as

$$\Delta \bar{T} = \mathbf{C}\ \bar{T} \tag{3.13}$$

where $\mathbf{C}$ is a coefficient matrix;
$\bar{T}$ is a vector of the temperatures; and
$\Delta\bar{T}$ is a vector of the temperature change.

The implicit scheme can be written as

$$\Delta\bar{T} = \mathbf{C}\left(\bar{T} + \frac{1}{2}\,\Delta\bar{T}\right) \tag{3.14}$$

which can be rewritten as

$$\left(\mathbf{I} - \frac{1}{2}\,\mathbf{C}\right)\Delta\bar{T} = \mathbf{C}\,\bar{T} \tag{3.15}$$

where we need to solve for $\Delta\bar{T}$ at each timestep.

The matrix

$$(\mathbf{I} - \frac{1}{2}\,\mathbf{C})$$

is diagonally dominant and sparse, because only neighboring points contribute nonzero values to $\mathbf{C}$.

Thus, this set of equations is efficiently solved by an iterative scheme. For ease of implementation as a simple extension of the explicit method, the Jacobi method is used. For the N × N system $\mathbf{Ax} = \mathbf{b}$, this can be written for the n^{th} iteration as

$$x_i(n+1) = \frac{b_i}{a_{ii}} - \sum_{\substack{j=1 \\ j\neq 1}}^{N}\left[\frac{a_{ij}}{a_{ii}}\,x_j(n)\right] \quad i = 1, 2, \ldots N \tag{3.16}$$

That is,

$$x_i(n+1) = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{N} a_{ij}\,x_j(n)\right] + x_i(n)$$

In our case, this becomes

$$\Delta T_i(n+1) = \frac{1}{1-\frac{1}{2}\,C_{ii}}\left[\sum_{j=1}^{N} C_{ij}\,T_j - \sum_{j=1}^{N}(\delta_{ij} - \frac{1}{2}\,C_{ij})\,\Delta T_j(n)\right] + \Delta T_i(n) \tag{3.17}$$

$$= \frac{1}{1-\frac{1}{2}\,C_{ii}}\left[\sum_{j=1}^{N} C_{ij}\,T_j + \frac{1}{2}\sum_{j=1}^{N} C_{ij}\,\Delta T_j(n) - \Delta T_i(n)\right] + \Delta T_i(n)$$

This equation shows the analogy between the implicit scheme and the explicit scheme, which can be written as

$$\Delta T_i = \sum_{j=1}^{N} C_{ij}\,T_j \tag{3.18}$$

The amount of calculation required for each timestep is approximately $n + 1$ times that required for one timestep in the explicit scheme, where n is the number of iterations per timestep. This extra calculation can be more than offset by the much larger timestep permitted by the implicit method, which makes the implicit scheme advantageous when the temperature change is linear in time.

3.2.5 Stability and Accuracy of the Implicit Scheme

As described previously, the implicit solution scheme has the advantage that it is unconditionally stable for all timesteps. However, the differencing scheme presented in Eq. (3.11) assumes that the temperature change is a *linear* function of time in a single timestep. Depending on the problem to be modeled, this assumption may lead to inaccurate results if temperature gradients are very high or are changing very rapidly (e.g., at early times in a simulation).

The code uses a Jacobi iteration method to solve the system of equations at every timestep. From a strictly numerical perspective, convergence of the iteration is achieved if

$$|a_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{N} |a_{ij}| \qquad i = 1, 2, ..., N \tag{3.19}$$

where a_{ij} are the previously described coefficients of the solution matrix $\boldsymbol{A}$.

The above condition simply means that it is possible to obtain a numerical solution to the system of equations but that the solution has no bearing on the accuracy with which the derived solution compares to the true solution.

There is no explicit method for determination of convergence to the true solution as a function of timestep, since the convergence depends on many factors (including properties, grid dimensions and grading, and boundary conditions). In most cases, the critical timestep (from Eq. (3.10)) provides a lower-bound estimate for the implicit timestep. A trial-and-error procedure is required to set the timestep above this value. Typically, a thermal problem is set up and initialized using the explicit procedure.

3.2.6 Thermal-Stress Coupling

The heat transfer may be coupled to thermal-stress calculations at any time during a transient simulation. The coupling occurs in one direction only (i.e., the temperature may result in stress changes, but mechanical changes in the body resulting from force application do not result in temperature change). This restriction is not believed to be of great significance here, since the energy changes for quasi-static mechanical problems is usually negligible. The stress change in a triangular zone is given by (from Eq. (3.5))

$$\Delta\sigma_{ij} = -\delta_{ij}\, 3K\, \alpha\, \Delta T \tag{3.20}$$

This assumes a constant temperature in each triangular zone, which is interpolated from the surrounding gridpoints. This stress is added to the zone stress state prior to application of the constitutive law.

3.3 Solving Thermal-Only and Coupled-Thermal Problems

UDEC has the ability to perform thermal analysis and coupled thermal-mechanical and thermal-fluid flow analysis. The form of the coupled thermal-mechanical interaction is described in Section 3.3.2. The coupling of thermal analysis with fluid flow in joints is described in Section 2.2.8. In all cases, the **block config** command must be given with the **thermal** keyword before any **block** command is specified.

The procedure and required commands to implement the thermal-only and thermal-mechanical analysis approach is described in the following sections. The application of the thermal analytic capability is illustrated by several verification problems in Section 3.6.

3.3.1 Thermal Analysis

UDEC can perform both transient and steady-state thermal analysis. The thermal calculation is performed with the **block thermal cycle** command. In order to perform a thermal-only analysis, the linking to the mechanical calculation must be suppressed with the command **block thermal substep-mechanical = 0**. This is the default state.

Both explicit and implicit solution methods are available for thermal analysis. By default, an explicit solution procedure is invoked with the **block thermal cycle** command. The thermal timestep is calculated from Eq. (3.10). A number of thermal steps can be specified with the **block thermal cycle step** command. Alternatively, a heating time limit, in seconds, can be specified with the **block thermal cycle age** command. The change in temperature during one thermal timestep is limited to 20°, by default. The thermal calculation will stop if this limit is exceeded. The limit can be changed with the **temperature-change** keyword following the **block thermal cycle** command, or the temperature change can be reduced by reducing the thermal timestep with the **block thermal cycle timestep** or **block thermal timestep** command. The thermal timestep is printed to the screen when **block thermal cycle** is given. The timestep can also be obtained with the **block list info** command.

The implicit solution algorithm described in Section 3.2.4 is implemented with the keyword **implicit** following the **block thermal cycle** command. The thermal timestep can then be adjusted with the keyword **timestep** following the **block thermal cycle** command, or with the **block thermal timestep** command. It is permissible to change between implicit and explicit solution methods at any time, using the **cycle** and **cycle implicit** keywords. The explicit scheme is always used unless the keyword **implicit** is given.

The advantage of an implicit method is that the timestep **block thermal timestep** is not restricted by numerical stability. There are three disadvantages:

(1) extra memory is required to use this method;

(2) a set of simultaneous equations must be solved at each timestep; and

(3) larger timesteps may introduce inaccuracy.

These disadvantages must be kept in mind when deciding which method to use. They are discussed below.

Memory Requirement – If an attempt is made to use the implicit method for a problem when the *UDEC* memory is almost full (typically, when the **block list mem** command reports at least 95% full), an error message may be generated. The only ways to avoid this are to run a smaller problem or use the explicit method.

Solving a Set of Equations – The set of equations to be solved at each timestep is solved iteratively. Each iteration of the solution takes about the same length of time as a single step of the explicit method. The number of iterations depends on the timestep chosen and the particular problem being solved, but is always at least 3. Thus, the implicit scheme only offers an advantage over the explicit scheme if the timestep is much larger than what the explicit scheme would use. On the other hand, the iterative scheme does introduce some restriction on the timestep. In general, a timestep between 100 and 10,000 times that used by the explicit scheme is satisfactory.

The program displays the iteration counter and a measure of convergence (the residual) to the left of the timestep counter while the implicit scheme is running. The user should check that the number of iterations being taken is such that the implicit scheme is indeed more efficient than the explicit scheme. If not, switch to the explicit scheme or change the timestep. This counter will also indicate whether the method is not converging. If the residual is increasing with successive iterations, the method is not converging, and a smaller timestep must be used.

Inaccuracy due to Large Timesteps – In the initial period of a solution, temperatures generally change much faster than later in the solution period. In addition, the implicit scheme uses more iterations when modeling rapid changes. It is appropriate, therefore, to use a smaller timestep or the explicit method, initially, and then to switch to the implicit method with a large timestep later in the solution period. Convergence of the solution generally occurs in fewer iterations at later timesteps.

Selecting the Implicit Method – From the preceding discussion, it can be seen that the implicit method is most efficient when used at late times in the solution, and only if the timestep can be increased significantly over the one used by the explicit scheme.

3.3.2 Thermal-Mechanical Analysis

The thermal calculation can be combined with the mechanical calculation to perform a thermal-mechanical analysis with *UDEC*. All the features of the thermal calculation (including transient and steady-state heat transfer, and thermal solution by either the explicit or implicit algorithm) are available in a thermal-mechanical calculation.

The thermal-mechanical coupling is provided by the influence of temperature change on the volumetric change of a zone (see Eq. (3.20)). The linear thermal expansion coefficient is assigned via the keyword **thermal-expansion** given with the **block property** command.

The thermal model applies to all zones in the *UDEC* model. If zones are made null mechanically, the thermal model automatically is made null as well.

The thermal-mechanical coupling can be invoked for any of the built-in mechanical constitutive models for plane-strain analysis. Plane-stress analysis can only be performed with the elastic isotropic and strain-hardening/softening models.

The most common way to use *UDEC* to solve thermomechanical problems is to come to initial mechanical equilibrium, and then take thermal steps to a time of interest. Remember that transient thermal problems involve time (e.g., the solution may be required after 10 years of heating). At this point, the mechanical problem has not been solved, although temperatures have been calculated. Mechanical steps are then taken until equilibrium is reached. This process is illustrated in Figure 3.2:

1. SETUP
   - configuration for thermal analysis **(CONFIG thermal)**
   - define problem geometry
   - define material models and properties
   - define thermal models and properties
   - set boundary conditions (thermal & mechanical)
   - set initial conditions (thermal & mechanical)
   - set any internal conditions, such as heat sources
2. STEP TO EQUILIBRATE MECHANICALLY **(STEP or SOLVE).**
3. PERFORM ANY DESIRED ALTERATIONS such as excavations.
4. STEP TO EQUILIBRATE MECHANICALLY.

   REPEAT steps 3 and 4 until "initial" mechanical state is reached for thermal analysis.
5. TAKE THERMAL TIMESTEPS until desired time is reached **(RUN).**
6. STEP TO EQUILIBRATE MECHANICALLY **(STEP or SOLVE).**

   REPEAT steps 5 and 6 until sufficient time has been simulated.

   REPEAT steps 3 to 6 as necessary.

Figure 3.2 General solution procedure for thermal-mechanical analysis

UDEC can be used in the usual way to model the excavation of material, change material properties and change boundary conditions. The mechanical logic (the standard *UDEC* program) is also used in the thermomechanical program to take “snapshots” of the mechanical state at appropriate intervals in the development of the transient thermal stresses.

The **block solve**, **block step** or **block cycle** command is used to control the mechanical steps. The **block thermal cycle** command controls the thermal process. For a problem in which the number of thermal steps is small before mechanical stepping is needed, analyses require a sequence of many **block thermal cycle** and **block step** commands, which can be cumbersome to create and run. Therefore, it is possible to use the **block thermal cycle** command to switch automatically to mechanical steps during a series of thermal steps, using the **block thermal substep-mechanical** command. **block thermal substep-thermal** specifies the increment of thermal steps at which mechanical steps are to

be taken. If **block thermal substep-thermal** is not zero, the calculation will switch to mechanical stepping every **substep-thermal** steps, *or* when the temperature change parameter (set with **block thermal cycle temperture-change**) is exceeded. **block thermal substep-mechanical** specifies the maximum number of mechanical steps executed between thermal steps. The mechanical calculation sub-stepping will stop either when the maximum number of mechanical steps is reached or the maximum unbalanced force ratio becomes smaller than 10^{-5}. The default is **substep-mechanical=500**.

A difficulty with thermal-mechanical analysis is that a large temperature increase may cause a large increase in unbalanced forces in the blocks. If the analysis being performed is linear-elastic, no temperature increase will be too great, and *UDEC* need only equilibrate when the simulation time is such that a solution is required. For nonlinear problems, it is necessary to experiment to obtain an acceptable temperature increase effect on unbalanced forces. This can be done with the following steps.

1. Save the mechanical equilibrium state reached by *UDEC*. This state may be restored later for additional analyses.

2. Plot the stresses and shear displacements. If the stresses are near yield, the thermal stresses caused by the temperature changes should not be large. If the stresses are far from yield, larger stresses can be tolerated.

3. Run thermal steps until a particular temperature increase is reported by the program (using a **block thermal cycle temperature-change** command).

4. Cycle mechanically to attain equilibrium.

5. Again, plot the stresses and shear displacements. If the area where the stresses are at or near yield is not much larger than at step 2, and the shear displacements are not very different, the allowed temperature increase was acceptable. If the changes are judged to be too great, the run must be repeated with a smaller allowed temperature change.

It is important to note that the same temperature increase is not necessarily acceptable for all times in a problem. While the system is far from yield, large temperature increases will be acceptable; near yield, only relatively small increases can be tolerated.

3.3.3 Heat Transfer across Joints

Heat transfers across the joints between blocks without resistance, provided that the blocks are in contact. Contacts are created along block edges for deformable blocks when the blocks are divided into triangular zones for mechanical calculations. For thermal calculations, the same zoning is used, with the exception that the triangles are further subdivided where the block is in contact with a corner on another block (Figure 3.3).

Rigid blocks are divided into triangles using the centroid as a common vertex of all the triangles, with the other vertices at the corner and at the contact with corners on other blocks (Figure 3.4).

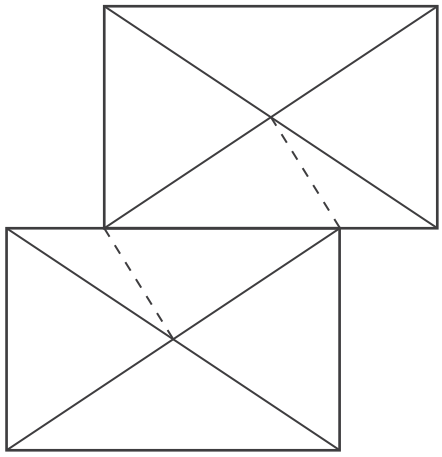

Figure 3.3 Subdivision of zones at contacts

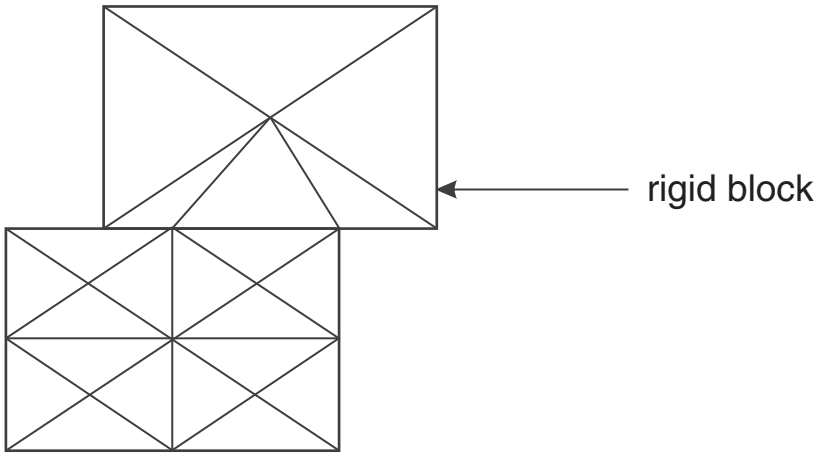

Figure 3.4 Typical zoning of rigid block

If the schemes outlined above are used without modification, it is possible for very narrow triangles (such as those shown in Figure 3.5) to be formed.

This causes inaccuracy, and may also lead to extremely small thermal timesteps. To avoid this, it is important to not have blocks with small zones neighboring blocks with large zones or large rigid blocks. The “thermal tolerance” option (keyword **tolerance-gridpoint**) on the **block thermal cycle** command should also be used to force points such as A and B in Figure 3.5 to be treated as one for thermal calculations.

The tolerance may need to be reduced for models that contain small blocks or fine zoning. The value for **tolerance-gridpoint** must be smaller than the smallest block or zone edge length if the gridpoints associated with the smallest edge are not to be combined for the thermal calculation. For this reason, the tolerance is reduced for the verification examples in Section 3.6.

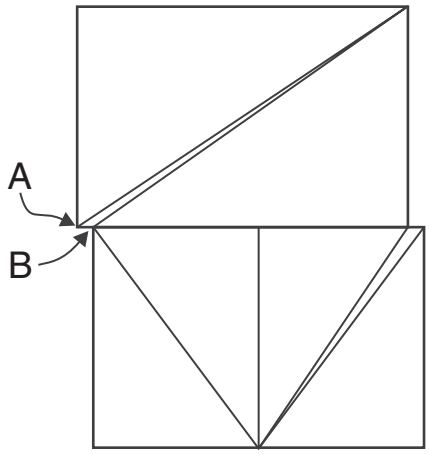

Figure 3.5 Zones that may cause inaccuracy

3.3.4 Thermal Boundary Locations

When modeling an infinite region, it is necessary to truncate the *UDEC* grid far enough away from the region of interest that the boundaries do not affect the solution. To determine whether the boundaries are far enough away, follow these steps:

1. Let the boundary representing infinity be insulated (the default boundary condition).
2. Solve the problem.
3. Examine the temperature changes on the boundary.
4. If the temperature changes are small, it is safe to assume the boundary has a negligible effect. If the temperature changes are not small, the boundary is probably too close. To confirm this, or disprove it, rerun the problem with the boundary temperatures fixed at their initial values. If the results are significantly different, the boundary was too close.

3.4 Input Instructions for Thermal Analysis

3.4.1 UDEC Commands

The following commands are provided to run thermal problems. Note that several thermal commands are invoked by new keywords used with existing commands in the standard mechanical code. The command **block config thermal** *must* be given before any **block create** command whenever a thermal analysis is to be performed. A summary of the thermal commands is given in Table 3.1:

Table 3.1 Summary of block thermal keywords

set thermal mode	**block config**	**thermal**
thermal properties	**property**	**conductivity**
		cond-x, cond-y
		specific-heat, thermal-expansion
	block fluid property	**permebility-table**
	block fluid property	**density-table**
initialize temperatures	**gridpoint init**	**temperature**
	gridpoint fix	**temperature**
	gridpoint free	**temperature**
specify thermal boundary conditions	**block edge apply**	**convection, flux**
		radiation, source
thermal-only solution		**substep-thermal = 0**
coupled thermal-mechanical solution	**cycle**	**substep-mechanical, substep-thermal, timestep, age, timestep, age-off, temp-change, step, tol-grid, implicit**
output options	**block gridpoint**	**list temp**
	block thermal	**list property**
	block thermal	**history temperature**
		history time-total

block **con**fig **th**ermal

This command specifies extra memory to be assigned to each zone or gridpoint for a thermal analysis. **block config thermal** *must* be given before any **block create** command. **block config thermal** can be combined with other calculation modes described in the UDEC help.

block **ed**ge **ap**ply keyword = ***value1, value2*** <**ran**ge. . . >

The **apply** command applies a thermal boundary condition to external boundaries. An optional **range** can be specified to limit the range of **apply**. (See Help in *UDEC*)

The following keywords are available.

convection ***value1*** is the convective heat transfer coefficient (w/m^2 °C).

value2 is the temperature of the medium to which convection occurs.

A convective boundary condition is applied between *corners* within the range.

flux ***value1*** is the initial flux (watts/m^2).

value2 is the decay constant (s^{-1}).

A flux boundary condition is applied between *corners* within the range. If a flux is applied between two blocks, the specified flux will be applied to both blocks.

radiation ***value1*** is the radiative heat transfer coefficient. (For black bodies, this is the Stefan-Boltzmann constant, 5.668×10^{-8} w/m^2 K^4.)

value2 is the temperature of the medium to which radiation occurs.

A radiation boundary condition is applied between corners within the range.

To remove a **convection** or **radiation** boundary condition, the same condition should be applied with the heat transfer coefficient of opposite sign.

CAUTION: It is not physically realistic to use negative heat transfer coefficients in any other circumstances.

To remove a **flux** condition, the condition should be applied with the strength replaced by S_{rep}, where

$$S_{rep} = - S_{ini} * \exp[c_d (t_{curr} - t_{ini})]$$

Note that unless otherwise specified by the **block edge apply** command, all boundaries are adiabatic (i.e., insulated).

block **fl**uid **dens**ity-**t**able ***n***

Fluid density is a function of temperature. Table number ***n*** is used to look up variations of fluid density as a function of temperature.

block **gr**idpoint keyword

history keyword

temperature ***x, y***

history of the temperature at a gridpoint

initialize **temp**erature ***value***

The temperature is set to ***value*** at all corners and gridpoints in the range. Thermal stresses are not induced by this method of setting the temperature.

list keyword

temperature gridpoint temperature

The column headings are:

(1) gridpoint address;

(2) x-coordinate of gridpoint;

(3) y-coordinate of gridpoint; and

(4) temperature at gridpoint.

fix ***keyword*** <**ran**ge. . . >

temperature ***value***

The temperatures at all corners and gridpoints are held fixed at ***value*** during the simulation. If ***value*** is not the current temperature, stresses are induced by the difference between ***value*** and the current temperature. An optional **range** can be given to limit the range of **fix**. (See Help in *UDEC*)

source ***value1 value2***

The **source** keyword results in a heat source of the stated strength applied to all gridpoints in the range. The initial strength is ***value1***. The decay constant is ***value2*** (s−1). The correct units for **source** are Watts, (cal/s) or the British equivalents.

volume-**so**urce ***value1 value2***

The **volume-source** applies heat of the stated strength in all blocks that have *centroids* in the specified range. The initial strength is ***value1***. The decay constant is ***value2*** (s−1). The user is responsible for determining the strength of the source for different size blocks. The initial strength to be given for each block is the intended power/volume ratio multiplied by the area of the block. The correct units for **volume-source** are Watts/m, (cal/s)/cm or the British equivalents.

The decay constant in the **source** and **flux** options is defined by the equation

$$S_{curr} = S_{ini} * \exp[c_d\ (t_{curr} - t_{ini})]$$

where S_{curr} = current strength;
S_{ini} = initial strength;
c_d = decay constant;
t_{curr} = current time; and
t_{ini} = initial time

To remove a **source**, the source should be applied with the strength replaced by S_{rep}, where

$$S_{rep} = - S_{ini} * \exp[c_d\ (t_{curr} - t_{ini})]$$

NOTE: By default, all temperatures are free to change initially.

free <**ran**ge. . . >

The temperatures at all corners and gridpoints are allowed to change during the simulation. An optional **range** can be specified to limit the range of **free**. (See Help in *UDEC* .)

NOTE: By default, all temperatures are free to change initially.

block **th**ermal keyword

cycle <<keyword ***value***>. . . >

This command executes thermal timesteps. Calculation is performed until some limiting condition is reached. The limiting condition may be the temperature increase at any point, the number of steps, or the simulated age. The limits are changed by the optional keywords listed below. Once a particular limit is specified, it is used for future **cycle** commands.

age ***t***

thermal "heating time" limit (in consistent units with input properties)

timestep ***dt***

The thermal timestep, ***dt***, is calculated automatically by the program. This parameter allows the user to change the timestep. If the program determines that this value is too large when the explicit scheme is used, it will automatically reduce the timestep to a suitable value when it begins the analysis. The value determined by the program is usually one-half the critical value for numerical stability. If the program selects a value that causes instability, this option can be used to further reduce the timestep.

age-off

turns off the previously requested test for exceeding age ***t***. The default for the age parameter is that the age is not tested until an age has been explicitly requested via an "**age** = ***value***" following a **cycle** command.

step ***s***

thermal step limit (default = 100,000)

temperature**-ch**ange ***dtp***

maximum total temperature change, ***dtp*** since the previous mechanical cycles (default is ***dtp*** = 20)

Two other keywords are available:

implicit

uses the implicit scheme rather than the default explicit scheme.

tolerance-**gr**idpoint ***tol***

Points in this tolerance are merged for thermal calculations (default = 0.1).

Old limits apply when set or restarted. When a **cycle** command has been completed, the program will indicate which parameter has caused it to terminate. To ensure that it stops for the correct one, the values of the others should be set very high.

The explicit scheme is always used unless the keyword **implicit** follows the **cycle** command.

history keyword

tiime-**to**tal

history of real time for heat transfer problems

list

property

The properties relevant to the thermal model are printed.

boundary

Thermal boundary conditions and sources are printed.

property **m**aterial ***n*** keyword ***v*** <keyword ***v***>

The following keywords are allowed.

conductivity thermal conductivity

specific-**h**eat specific heat

thermal-**ex**pansion linear thermal expansion coefficient

conductivity-**x** thermal conductivity in x-direction

conductivity-**y** thermal conductivity in y-direction

The actual properties used by the program are the thermal conductivities in the x- and y-directions. The **cond** keyword simply sets the conductivities in both directions equal to the set value.

For **contact model** = 2 or 5, joint permeability can be specified as a function of temperature by the following keyword.

permeability **-t**able ***n***

Table ***n*** (see the **table** command) contains a list of pairs (e.g., permeability and temperature) defining the temperature dependency. The table applies to *all* joint permeabilities, regardless of joint material number. Note that for a "parallel-plate" joint, **permeability-factor** = (1/12) μ, in which μ is the fluid dynamic viscosity.

substep-**me**chanical ***n***

maximum number of mechanical steps executed between thermal steps, when **substep-thermal** is nonzero (see below). The mechanical stepping will stop when either the maximum step number defined by **substep-mechanical** is reached or the maximum unbalanced force ratio becomes smaller than 10^{-5}. The default value = 500.

substep-**th**ermal ***n***

number of thermal steps to do before switching to mechanical steps

NOTE: The default value of **substep-thermal** is zero, in which case no interlinking occurs. If **substep-thermal** is not zero, the program will switch to mechanical steps every **substep-thermal** steps *or* when the temperature change parameter (**cycle temp-change** = ***value***) is violated. If the temperature change parameter is violated when **substep-thermal = 0**, thermal cycling stops, and further thermal or mechanical cycling is controlled by the user.

CAUTION: Geometry changes are ignored by the thermal model until a **cycle** command is given. This means that when the mechanical models are accessed automatically, the geometry changes are ignored on return to thermal steps. If large geometry changes occur, it is better to divide the run into several **cycle** commands instead of only one.

timestep The thermal timestep is set to ***value***.

NOTE: The program calculates the thermal timestep automatically. This keyword allows the user to choose a different timestep. For the explicit method, if the program determines that the chosen step is too large, it will automatically reduce it to a suitable value when thermal steps are taken. It will not revert to a user-selected value until another **block thermal timestep** command is issued. The program selects a value that is usually one-half the critical value for numerical stability. This command has the same effect as a **cycle timestep** command.

3.4.2 FISH Variables

The following scalar variables are available in a *FISH* function to assist with thermal analysis.

block.**th**ermal.**timestep** for the thermal calculation.

block.**th**ermal.**ti**me.**total** thermal time

block.**gr**idpoint.**temperature** gridpoint temperature.

Also, thermal property values may be accessed (changed, as well as tested) in a *FISH* function. See the **block thermal property** command in Section 3.4.1 for a list of the thermal properties.

3.5 Systems of Units for Thermal Analysis

All thermal quantities must be given in an equivalent set of units. No conversions are performed by the program. Tables 3.2 and 3.3 present examples of consistent sets of units for thermal parameters.

Table 3.2 System of SI units for thermal problems

Length	m	m	m	cm
Density	kg/m^3	10^3 kg/m^3	10^6 kg/m^3	10^6 g/cm^3
Stress	Pa	kPa	MPa	bar
Temperature	K	K	K	K
Time	s	s	s	s
Specific Heat	J/(kg K)	10^{-3} J/(kg K)	10^{-6} J/(kg K)	10^{-6} cal/(g K)
Thermal Conductivity	W/(mK)	W/(mK)	W/(mK)	$(cal/s)/cm^2$ K^4
Convective Heat Transfer Coefficient	$W/(m^2$ K)	$(W/m^2$ K)	$W/(m^2$ K)	$(cal/s)/(cm^2$ K)
Radiative Heat Transfer Coefficient	$W/(m^2$ K^4)	$W/(m^2$ K^4)	$W/(m^2$ K^4)	$(cal/s)/cm^2$ K^4
Flux Strength	W/m^2	W/m^2	W/m^2	$(cal/s)/cm^2$
Source Strength	W/m^3	W/m^3	W/m^3	$(cal/s)/cm^3$
Decay Constant	s^{-1}	s^{-1}	s^{-1}	s^{-1}
Stefan-Boltzmann Constant	5.67×10^{-8} W/m^2 K^4	5.67×10^{-8} W/m^2 K^4	5.67×10^{-8} W/m^2 K^4	1.356×10^{-12} $cal/(cm^2$ s K^4)

Table 3.3 System of Imperial units for thermal problems

Length	ft	in
Density	$slugs/ft^3$	$snails/in^3$
Stress	lb_f	psi
Temperature	R	R
Time	hr	hr
Specific Heat	$(32.17)^{-1}$ Btu/(1b R)	$(32.17)^{-1}$ Btu/(1b R)
Thermal Conductivity	(Btu/hr)/(in R)	(Btu/hr)/(in R)
Convective Heat Transfer Coefficient	$(Btu/hr)/(ft^2$ R)	$(Btu/hr)/(in^2$ R)
Radiative Heat Transfer Coefficient	$(Btu/hr)/(ft^2$ R^4)	$(Btu/hr)/(in^2$ R^4)
Flux Strength	$(Btu/hr)/ft^2$	$(Btu/hr)/in^2$
Source Strength	$(Btu/hr)/ft^3$	$(Btu/hr)/in^3$
Decay Constant	hr^{-1}	hr^{-1}
Stefan-Boltzmann Constant	1.713×10^{-9} $Btu/(ft^2$ hr R^4)	1.19×10^{-11} $Btu/(in^2$ hr R^4)

where 1K = 1.8 R;

1J = 0.239 cal = 9.48×10^{-4} Btu;

1J/kg K = 2.39×10^{-4} btu/1b R;

1W = 1 J/s = 0.239 cal/s = 3.412 Btu/hr;

1W/m K = 0.578 Btu/(ft/hr R); and

1W/m^2 K = 0.176 Btu/ft^2 hr R.

Note that, unless radiation is being used, temperatures may be quoted in the more common units of $^\circ C$ (instead of K) or $^\circ F$ (instead of R), where

$$\text{Temp}(^\circ C) = \tfrac{5}{9} * (\text{Temp}(^\circ F) - 32);$$

$$\text{Temp}(^\circ F) = (1.8\ \text{Temp}(^\circ C)) + 32;$$

$$\text{Temp}(^\circ C) = \text{Temp}(K) - 273; \text{ and}$$

$$\text{Temp}(^\circ F) = \text{Temp}(R) - 460.$$

3.6 Verification Examples

Several verification examples are presented to demonstrate the thermal model in *UDEC*. The data files for these examples are located in the "Datafiles\Thermal" directory.

All of the models contain joints. This allows the evaluation of joint behavior on the thermal and thermal-mechanical response of the models. The joint stiffnesses can influence the results for thermal-mechanical analyses; stiffnesses are at least two to three orders of magnitude higher than the block stiffnesses for the thermal-mechanical examples.

3.6.1 Conduction through a Composite Wall

An infinite wall consisting of two distinct layers is exposed to an atmosphere at a high temperature on one side and a low temperature on the other. The wall eventually reaches an equilibrium at a constant heat flux and unchanging temperature distribution.

The two layers of the wall have the specifications presented in Table 3.4. Figure 3.6 shows the wall geometry and boundary conditions.

The wall is of infinite height and thickness, and the temperatures of the atmosphere on either side are constant. The two layers, individually, are homogeneous and isotropic, and the conductive contact between them is perfect.

Table 3.4 Problem specifications

temperature of outside	T_i = 3000°C	T_o = 25°C
convection coefficient	h_i = 100 w/m^2 °C	h_o = 15 w/m^2 °C
thermal conductivity	k_1 = 1.6 w/m°C	k_2 = 0.2 w/m°C
thickness	d_1 = 25 cm	d_2 = 15 cm

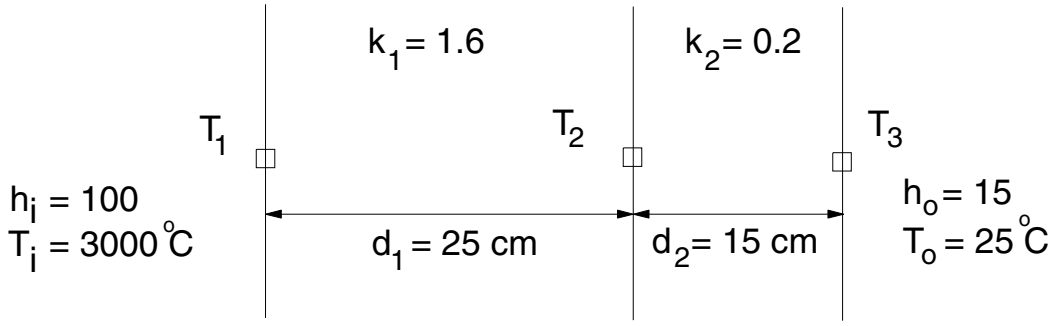

Figure 3.6 Composite wall

The analytical steady-state solution to this problem is quite simple and common. The total equilibrium heat flux is

$$\frac{q}{A} = \frac{T_i - T_o}{R_T} \tag{3.21}$$

where R_T is the sum of the four thermal resistances:

$$R_1 = 1/h_i$$
$$R_2 = d_1/k_1$$
$$R_3 = d_2/k_2$$
$$R_4 = 1/h_o$$

This heat flux is constant across the three interfaces. Hence, after setting this flux equal to the temperature difference divided by the interface resistance and solving for the unknown, we arrive at

$$T_1 = T_i - \frac{q}{A} \cdot \frac{1}{h_i}$$
$$T_2 = T_1 - \frac{q}{A} \cdot \frac{d_1}{k_1} \tag{3.22}$$
$$T_3 = T_2 - \frac{q}{A} \cdot \frac{d_2}{k_2}$$

The temperature will vary linearly among the three.

Example 3.1 contains the input commands necessary to solve this problem with *UDEC*. Commands for both the explicit and implicit solutions are given. The data file as shown runs the explicit solution. If the semicolon is removed from the **block thermal timestep=6000** command, and the **implicit** keyword is added to the **block thermal cycle** command, then the implicit solution will be performed. The model is run to a thermal age of 600,000 seconds to reach steady-state for both solutions.

Example 3.1 Conduction through a composite wall

```
model new
model title 'Thermal conduction'
block config thermal
block tolerance corner-round-length 0.001
block tolerance minimum-edge-length 0.002
block create polygon 0 0 0 2.5E-2 0.4 2.5E-2 0.4 0
block cut crack 0.25 -1 0.25 1 join
block zone gen edge 0.05 range pos-x 0 0.25 pos-y 0 2.5E-2
block zone gen edge 0.025 range pos-x 0.25 0.4 pos-y 0 2.5E-2
block zone group 'wall:high density' range pos-x 0.0 0.25 pos-y 0 2.5E-2
block zone group 'wall:low density' range pos-x 0.25 0.4 pos-y 0 2.5E-2
block zone cmodel assign elastic density 1E4 cond 1.6 specheat 300 ...
  range group 'wall:high density'
block zone cmodel assign elastic density 1.25E3 cond 0.2 specheat 300 ...
  range group 'wall:low density'
block edge apply convection 100.0 3000.0 ...
  range pos-x -0.001 0.001 pos-y -0.001 2.6E-2
block edge apply convection 15.0 25.0 ...
  range pos-x 0.399 0.401 pos-y -0.001 2.6E-2
block gridpoint init temperature 2500.0 ...
  range pos-x -0.001 0.251 pos-y -0.001 2.6E-2
block gridpoint init temperature 1000.0 ...
  range pos-x 0.249 0.401 pos-y -0.001 2.51E-2
block thermal cycle age 600000.0 temperature-change 100000.0 ...
  tolerance-gridpoint 1.0E-4
model save 'cond1.sav'
;
;
fish define constants
   d_1  = 0.25
   d_2  = 0.15
   t_i  = 3000.0
   t_o  = 25.0
   h_i  = 100.0
   h_o  = 15.0
   k_1  = 1.6
   k_2  = 0.2
   r_1  = 1.0/h_i
   r_2  = d_1/k_1
   r_3  = d_2/k_2
   r_4  = 1.0/h_o
   r_t  = r_1 + r_2 + r_3 + r_4
   q_a  = (t_i-t_o) / r_t
```

```
   t_1  = t_i - q_a * r_1
   t_2  = t_1 - q_a * r_2
   t_3  = t_2 - q_a * r_3
end
@constants
; store numerical results in table 1
fish define num_sol
   ib = block.head
   loop while ib # 0
      ig = block.gp(ib)
      loop while ig # 0
         x   = block.gp.pos.x(ig)
         tmp = block.gp.temp(ig)
         table(1,x) = tmp
         ig = block.gp.next(ig)
      endloop
      ib = block.next(ib)
   endloop
end
;
; store analytical results in table 2
fish define ana_sol
   ib = block.head
   loop while ib # 0
      ig = block.gp(ib)
      loop while ig # 0
         x = block.gp.pos.x(ig)
         if x < d_1 then
            tmp = x * ((t_2-t_1)/d_1) + t_1
         else
            x_2   = x - d_1
            tmp = x_2 * ((t_3-t_2)/d_2) + t_2
         endif
         table(2,x) = tmp
         ig = block.gp.next(ig)
      endloop
      ib = block.next(ib)
   endloop
end
@num_sol
@ana_sol
model save 'cond2.sav'
return
```

The wall is idealized by the geometry shown in Figure 3.7. Since the model is infinitely long in one direction, the model is essentially one-dimensional, and horizontal boundaries may be represented as adiabatic boundaries.

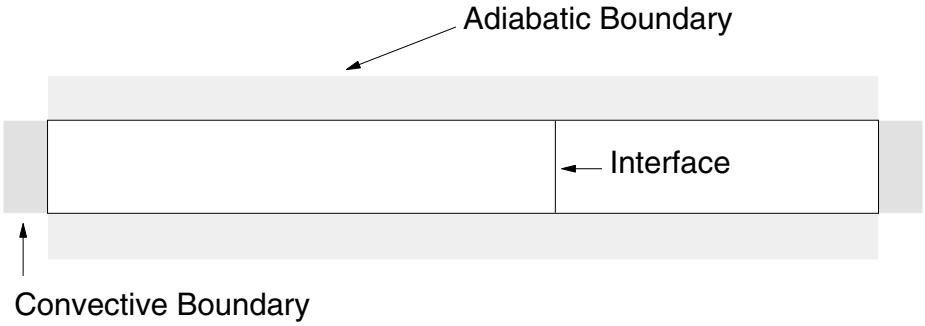

Figure 3.7 Idealization of the wall for the UDEC model

In the *UDEC* analysis, the wall is defined by two deformable blocks; each block corresponds to an individual layer of the wall. The zoning for the two blocks is shown in Figure 3.8. An adiabatic boundary condition (zero heat flux across boundary) is applied to the top and bottom of this model to simulate the infinite dimensions of the wall. (Adiabatic boundaries are the default condition.) The appropriate convective boundary conditions are applied to the ends of the grid, and the different sets of thermal properties are applied to the two blocks to model the composite material.

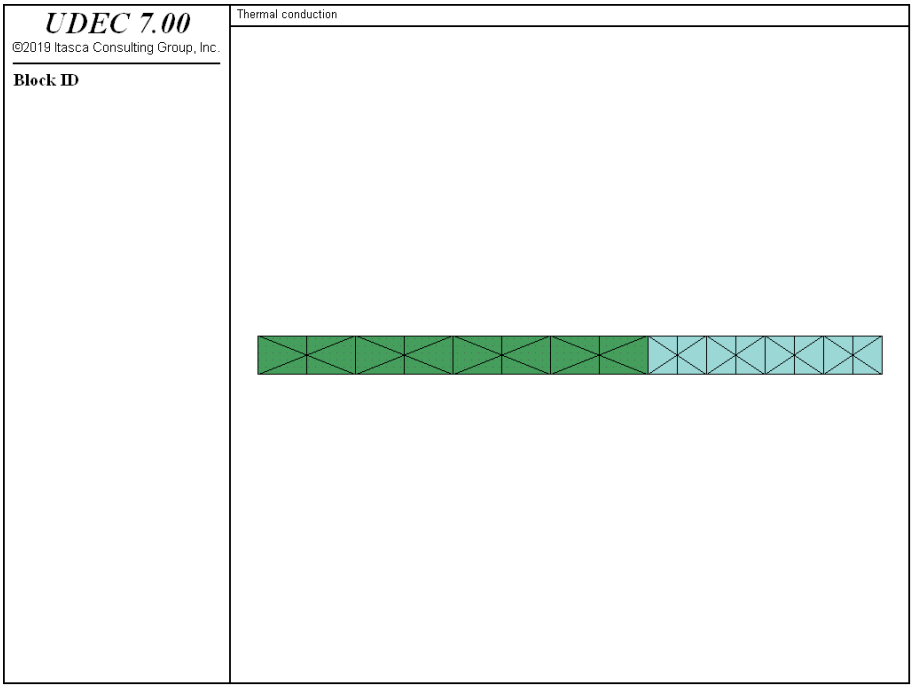

Figure 3.8 Zone distribution

Figure 3.9 shows a contour plot of the steady-state temperature distribution using the explicit solution.

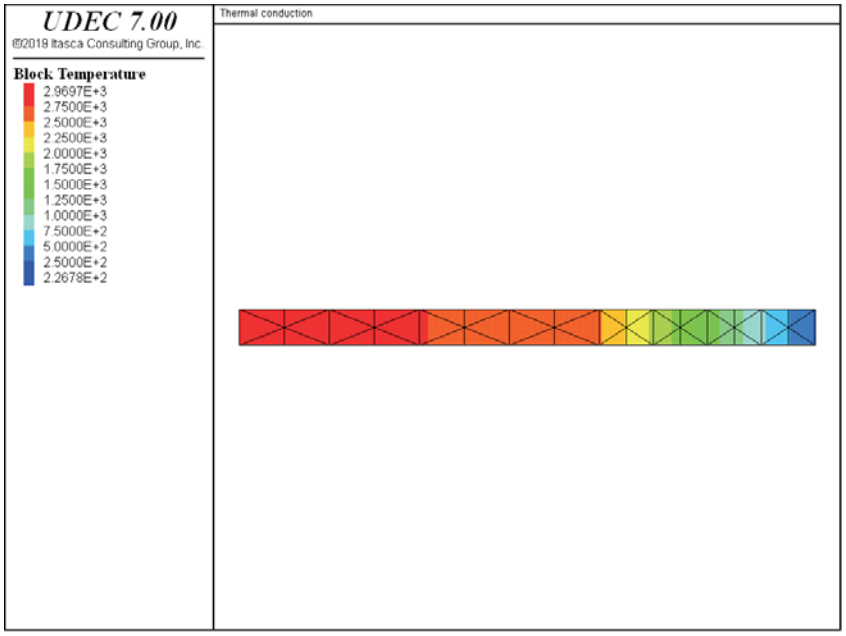

Figure 3.9 Steady-state temperature distribution

Figure 3.10 compares *UDEC*'s temperature distribution with the analytical solution. The numerical calculations for steady-state temperatures are stored in table 1, and the analytical values are stored in table 2 for comparison.

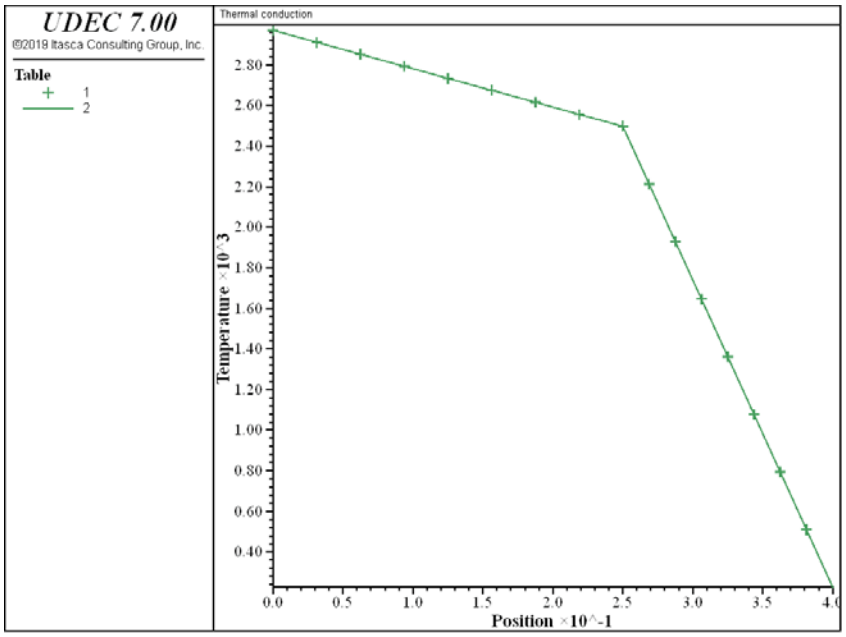

Figure 3.10 Temperature vs distance comparison between UDEC and analytical solution

Table 3.5 displays a more precise comparison for five points along the wall, including the three interface points ("interface" refers to thermal properties, not a mechanical interface). The results based on the implicit solution are identical.

Table 3.5 Comparison of UDEC results and the analytical solution

	Position	Analytical (°C)	*UDEC* (°C)	% Error
T_1	0	2970	2970	$< 0.01\%$
	0.125	2733	2733	$< 0.01\%$
T_2	0.250	2497	2496	-0.04
	0.325	1362	1361	-0.07
T_3	0.400	226.7	226.7	$< 0.01\%$

The comparison between *UDEC* and the analytical solution shows that, for this simple one-dimensional problem, *UDEC* produces excellent agreement. The errors on both the boundaries and the interface are negligible ($<0.1\%$).

3.6.2 Thermal Response of a Heat-Generating Slab

An infinite plate of thickness $2L$ = 1 m generates heat internally. This problem determines the transient temperature distribution after application of a constant temperature boundary condition.

The physical properties of the plate in question are

density (ρ)	500 kg/m^3
specific heat (C_p)	0.2 J/kg°C
thermal conductivity (k)	20 w/m°C

The plate is initially at a uniform temperature of 60°C, the surface is then fixed at 32°C, and the plate itself has internal heat generation of 40 kW/m^3 (as shown in Figure 3.11).

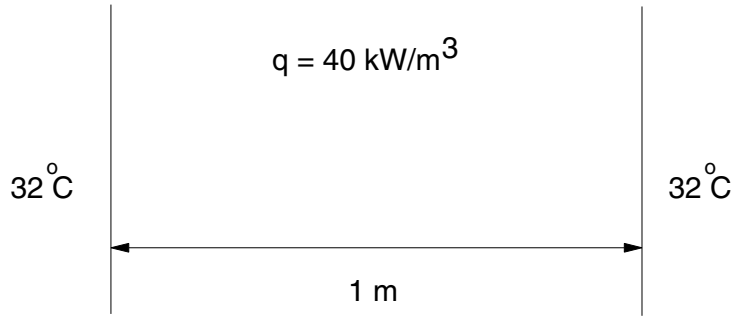

Figure 3.11 Heat-generating slab showing initial and boundary conditions

Assuming that the slab is infinitely long and that the material is homogeneous, isotropic and continuous, with temperature-independent thermal properties, the governing equation for this problem is

$$\frac{\partial^2 T}{\partial x^2} + \frac{Q}{k} = \frac{1}{\kappa} \cdot \frac{\partial T}{\partial t} \tag{3.23}$$

where T = temperature;

x = distance from slab centerline;

Q = constant volumetric heat generation rate;

k = thermal conductivity;

t = time; and

κ = diffusivity = $\frac{k}{\rho \cdot C_p}$.

By symmetry, only half of the plate is modeled. The applied initial and boundary conditions are

$$\begin{aligned} \frac{\partial T}{\partial x} &= 0 \quad x = 0;\ t > 0 \\ T &= T_s \quad x = L;\ t > 0 \\ T &= T_i \quad t \le 0 \end{aligned}$$

where T_i = initial uniform temperature;

T_s = constant temperature at the slab faces; and

L = slab half-width.

The integration of Eq. (3.23) is presented by Ozisik (1980):

$$T(x,t) = T_s + \frac{Q}{2k}(L^2 - x^2) + \frac{2}{L}(T_i - T_s)\sum_{m=0}^{\infty}(-1)^m e^{-\kappa\beta_m^2 t}\left(\frac{\cos(\beta_m x)}{\beta_m}\right) \tag{3.24}$$

$$-\frac{2Q}{Lk}\sum_{m=0}^{\infty}(-1)^m e^{-\kappa\beta_m^2 t}\left(\frac{\cos(\beta_m \cdot x)}{\beta_m^3}\right)$$

where β_m are the positive roots of the transcendental equation

$$\cos(\beta_m \cdot L) = 0 \quad \text{or} \quad \beta_m = \frac{(2m+1)\pi}{2L}, m = 0, 1, 2... \tag{3.25}$$

For steady-state condition ($t \to \infty$), the two last terms of Eq. (3.24) tend to zero, so

$$T_{steady}(x) = T_s + \frac{Q}{2k}(L^2 - x^2) \tag{3.26}$$

The conditions imposed to model this problem are shown in Figure 3.12; the corresponding *UDEC* model is given in Figure 3.13. Because the plate is infinitely long, and because the heat generation is uniform, symmetry conditions exist for any plane perpendicular to the long axis of the plate. These are represented by adiabatic boundaries. The right boundary of the model has a fixed temperature of 32°C. The left boundary is a symmetry line.

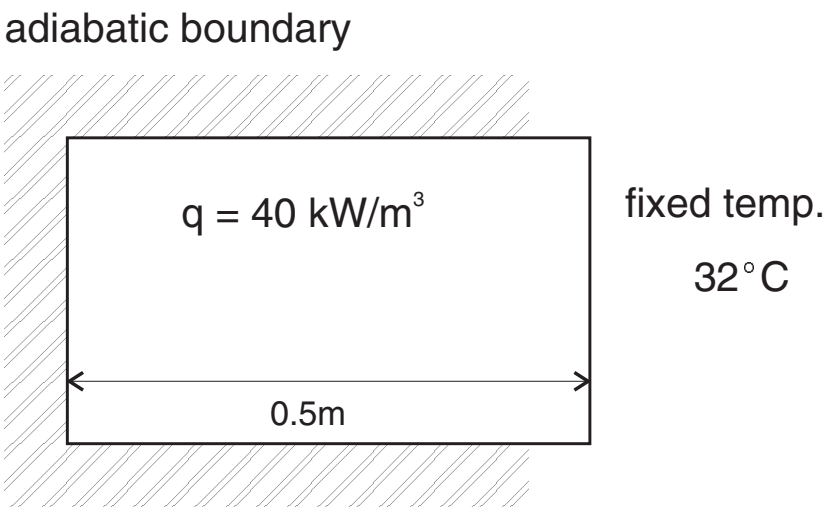

Figure 3.12 Model conditions for heat-generating slab

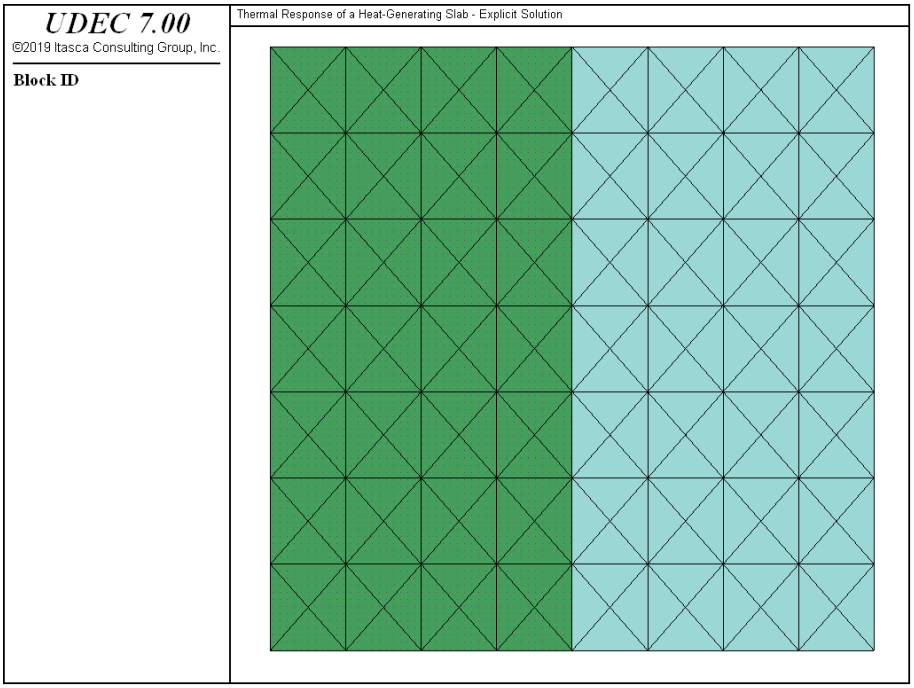

Figure 3.13 UDEC zone distribution

The model is run to a thermal time of 0.1, 0.5 and 5 seconds. The last time corresponds to a steady-state condition. Example 3.2 contains the input commands necessary to solve this problem with *UDEC*.

Example 3.2 Thermal response of a heat-generating slab

```
model new
model title ...
  'Thermal Response of a Heat-Generating Slab - Explicit Solution'
; --- problem constants ---
fish define constants
   tabo   = -1
   tabe   = 0
   v_l     = 0.5
   v_ti    = 60.
   v_ts    = 32.
   v_q     = 4.e4
   v_k     = 20.
   v_cp    = 0.2
   v_rho   = 500.
   v_kappa = v_k / (v_rho * v_cp)
   qk      = v_q / v_k
   tits    = v_ti - v_ts
   pi2l    = math.pi / (2.0 * v_l)
end
@constants
; store analytical results in odd numbered tables
; store numerical results in even numbered tables
fish define solution
  tabo = tabo + 2
  tabe = tabe + 2
  ib = block.head
  loop while ib # 0
    ig = block.gp(ib)
    loop while ig # 0
      if block.gp.pos.y(ig) > ym_tol then
        if block.gp.pos.y(ig) < yp_tol then
          x_p = block.gp.pos.x(ig)
          kt    = v_kappa * block.thermal.time.total
          s_old = 0.0
          monem = -1.
          m     = 0
          loop while m < 100
             monem  = -monem
             betam  = pi2l * (2.0 * m + 1.)
             betam2 = betam * betam
             e1     = math.exp(-kt*betam2)/betam
             s_new  = s_old+(tits-qk/betam2)*monem*math.cos(betam*x_p)*e1
             if s_new = s_old then
```

```
                m = 100
              else
                s_old = s_new
                m = m + 1
              end_if
            endloop
            v_temp = s_new*2./v_l+v_ts+qk*0.5*(v_l-x_p)*(v_l+x_p)
            u_temp = block.gp.temp(ig)
            table(tabo,x_p) = v_temp
            table(tabe,x_p) = u_temp
          endif
        endif
        ig = block.gp.next(ig)
      endloop
      ib = block.next(ib)
    endloop
end
fish set @ym_tol = .2
fish set @yp_tol = .3
;
block config thermal
block tolerance corner-round-length 1E-3
block tolerance minimum-edge-length 2E-3
block create polygon 0 0 0 0.5 0.5 0.5 0.5 0
block cut crack 0.25 0 0.25 1 join
block zone gen quad 0.075
block zone group 'block'
block zone cmodel assign elastic density 500 cond 20 specheat 0.2 ...
range group 'block'
block gridpoint init temperature 60.0
block gridpoint fix temperature 32.0 range pos-x 0.499 0.51 pos-y -1 2
block gridpoint fix volume-source 5000.0 0.0
;
block gridpoint history temperature 0.25 0.5
block gridpoint history temperature 0.0 0.5
block thermal history time-total
;
; thermal time = 0.1
;
block thermal cycle age 0.1 temp-change 200 tolerance-gridpoint 0.01
@solution
;
; thermal time = 0.5
block thermal cycle age 0.5 temp-change 200 tolerance-gridpoint 0.01
@solution
;
```

```
; thermal time = 5.0
block thermal cycle age 5.0 temp-change 2000 tolerance-gridpoint 0.01
@solution
;
table 1 label 'Temp. 0.1 Sec. - Analytic'
table 2 label 'Temp. 0.1 Sec. - UDEC'
table 3 label 'Temp. 0.5 Sec. - Analytic'
table 4 label 'Temp. 0.5 Sec. - UDEC'
table 5 label 'Temp. 5.0 Sec. - Analytic'
table 6 label 'Temp. 5.0 Sec. - UDEC'
model save 'heated_slab'
return
```

Figure 3.14 shows the evolution of temperature in the center of the slab (x = 0). Steady-state conditions are reached at t = 5. Figure 3.15 shows the temperature distribution at steady state.

The temperature distributions for t = 0.1, 0.5 and 5 seconds are compared to the analytical solution in Figure 3.16. The agreement is excellent, with an error of less than 1%.

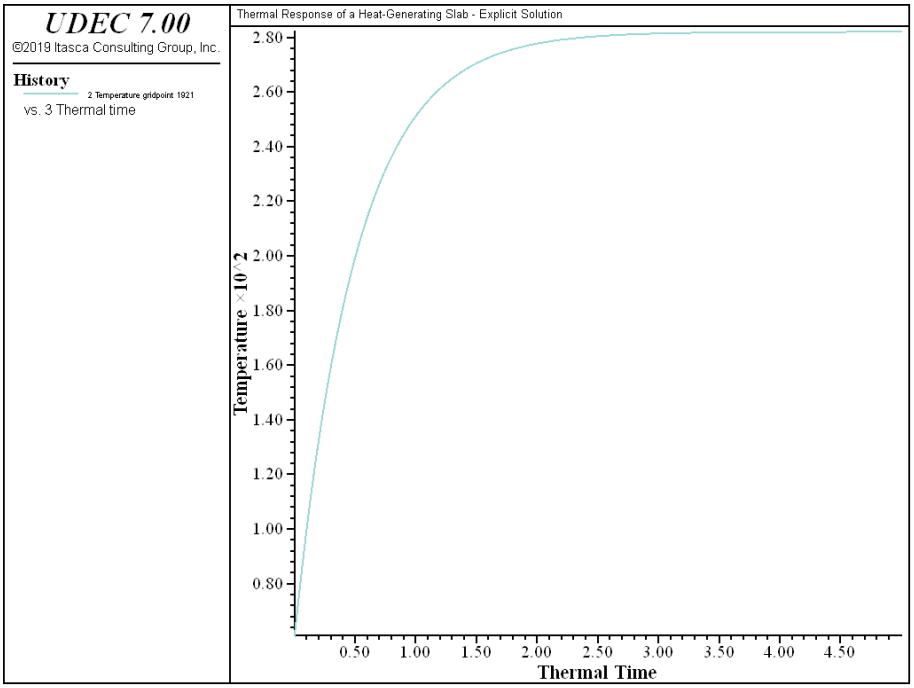

Figure 3.14 Temperature evolution in the center of the slab

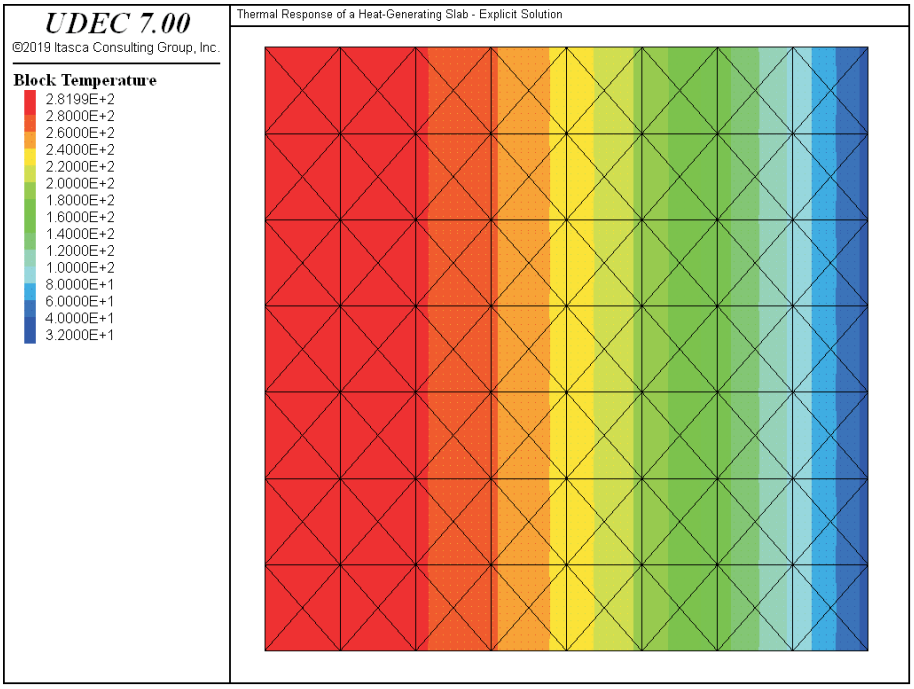

Figure 3.15 Temperature distribution at steady-state

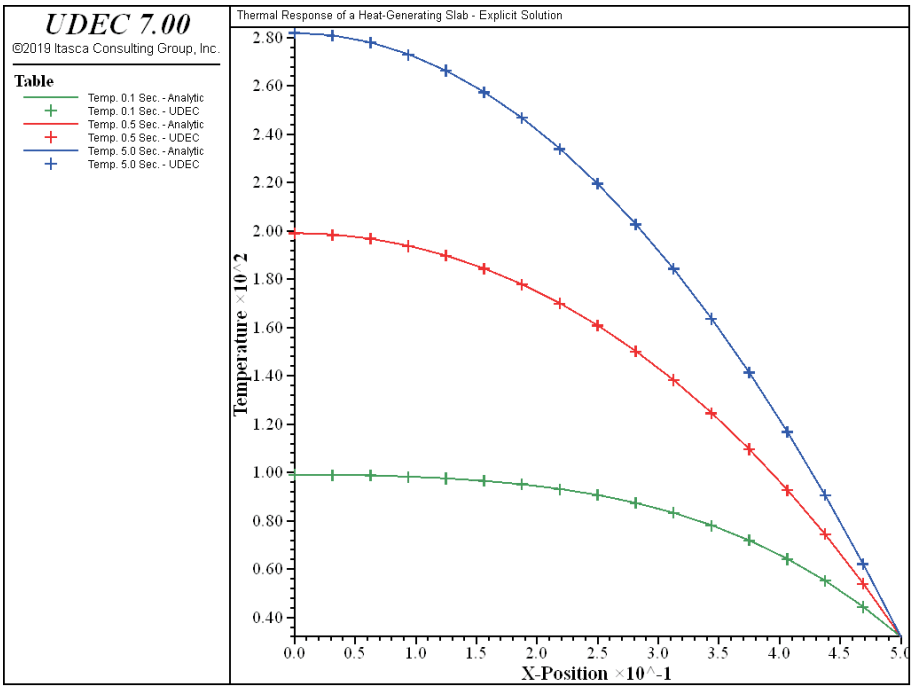

Figure 3.16 UDEC and analytical temperature distributions at thermal time = 0.1, 0.5 and 5.0 seconds (analytical values = odd-numbered tables; numerical values = even-numbered tables)

3.6.3 Heating of a Hollow Cylinder

A hollow cylinder of infinite length is initially at a constant temperature of 0°C. The inner radius of the cylinder is exposed to a constant temperature of 100°C, and the outer radius is kept at 0°C. The problem is to determine the temperatures and thermally induced stresses in the cylinder when the equilibrium thermal state is reached.

Nowacki (1962) provides the solution to this problem in terms of the temperatures and radial, tangential and axial stresses at the steady-state thermal state:

$$\frac{T(r)}{T_a} = \frac{\ln(b/r)}{\ln(b/a)} \tag{3.27}$$

$$\frac{\sigma_r(r)}{mGT_a} = -\left[\frac{\ln(b/r)}{\ln(b/a)} - \frac{(b/r)^2 - 1}{(b/a)^2 - 1}\right] \tag{3.28}$$

$$\frac{\sigma_t(r)}{mGT_a} = -\left[\frac{\ln(b/r) - 1}{\ln(b/a)} + \frac{(b/r)^2 + 1}{(b/a)^2 - 1}\right] \tag{3.29}$$

$$\frac{\sigma_a(r)}{mGT_a} = -\left[\frac{2\ln(b/r) - \frac{\lambda}{2(\lambda+G)}}{\ln(b/a)} + \left(\frac{\lambda}{2\lambda + G}\right)\left(\frac{2}{(b/a)^2 - 1}\right)\right] \tag{3.30}$$

where T = temperature;

r = radial distance from the cylinder center;

a = inner radius of the cylinder;

b = outer radius of the cylinder;

T_a = temperature at the inner radius;

σ_r = radial stress;

σ_t = tangential stress;

σ_a = axial stress;

$m = \frac{3K\alpha}{\lambda+2G}$;

$\lambda = K - \frac{2}{3}G$;

K is the bulk modulus;

G is the shear modulus; and

α is the linear thermal expansion coefficient.

The analytical solutions for temperature and stresses are programmed as *FISH* functions in the *UDEC* data file. The analytical and numerical results can then be compared directly in tables.

The following properties are prescribed for this example.

Geometry

inner radius of cylinder (a)	1.0 m
outer radius of cylinder (b)	2.0 m

Material Properties

density (ρ)	2000 kg/m^3
specific heat (C_p)	880.0 J/kg°C
thermal conductivity (k)	4.2 W/m°C
linear thermal expansion coefficient (κ)	5.4×10^{-6}/°C
shear modulus (G)	28.0 GPa
bulk modulus (K)	48.0 GPa

A quarter-section of the cylinder is modeled with *UDEC*. Figure 3.17 shows the *UDEC* zoning. A constant-temperature boundary of 100°C is specified for the inner radius of the model; the temperature at the outer radius is specified to be 0°C.

The *UDEC* model can be run as either a coupled or uncoupled thermal-mechanical analysis. In this example, we run the model in an uncoupled mode: the thermal calculation is performed first to reach the equilibrium heat flux state; then the thermally induced mechanical stresses are calculated. The *UDEC* data file is listed in Example 3.3.

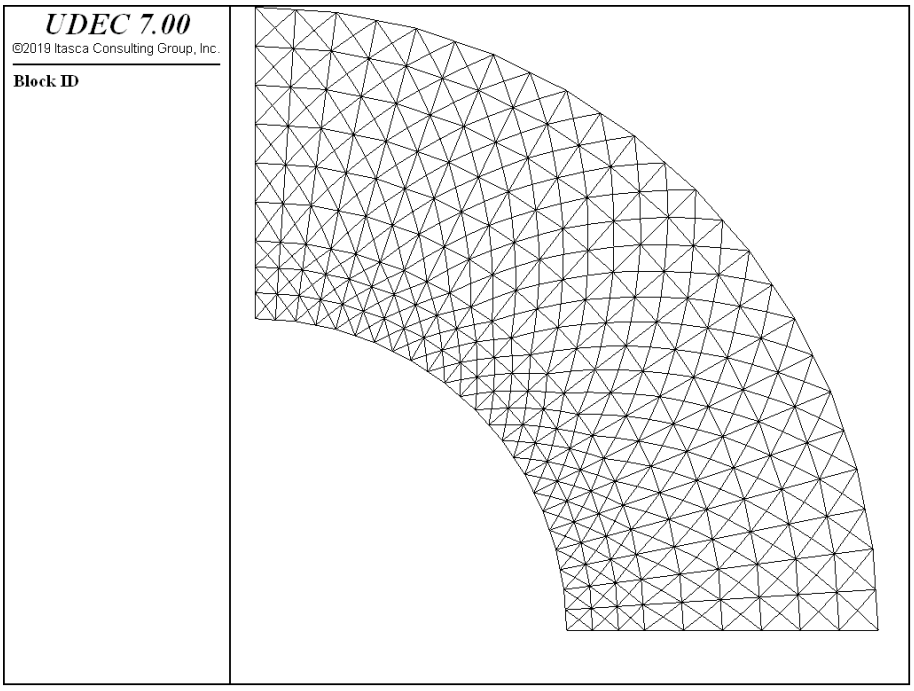

Figure 3.17 UDEC grid for heating of a hollow cylinder

Example 3.3 Heating of a hollow cylinder

```
model new
block config thermal
model title 'Heating of a Hollow Cylinder'
block tolerance corner-round-length=.001
; set geometry (one quarter of a rod)
block create polygon 0 0 0 2 2 2 2 0
block cut arc (0,0) (1.0,0) 90 12
block cut arc (0,0) (1.25,0) 90 12
block cut arc (0,0) (1.5,0) 90 12
block cut arc (0,0) (1.75,0) 90 12
block cut arc (0,0) (2.0,0) 90 12
; hollow out the rod to make a cylinder
block del range annulus center (0,0) rad 0.0 1.0
; delete outside of cylinder
block del range annulus center (0,0) rad 2.0 3.0
; add construction joints for zoning
block cut joint-set angle 22.5 spacing 4 origin 0,0
block cut joint-set angle 45 spacing 4 origin 0,0
block cut joint-set angle 67.5 spacing 4 origin 0,0
block contact join by-cont
block zone gen quad 0.1  range ann center (0,0) rad 0 1.2
block zone gen quad 0.15 range ann center (0,0) rad 0 1.5
block zone gen quad 0.2
; set material properties
block zone group 'block'
block zone cmodel assign elastic density 2E3 bulk 4.8E10 ...
  shear 2.8E10 cond 4.2 specheat ...
 880 thexp 5.4E-6 range group 'block'
; set boundary conditions
bl grid app vel-y 0.0 range pos-x (0,3) pos-y (-.01,.01)
bl grid app vel-x 0.0 range pos-x (-.01,.01) pos-y (0,3)
; temperature=100 fixed at radius= 1
block gridpoint fix temperature 100.0  ...
  range ann center (0,0) rad 0.99 1.01
; temperature=0 fixed at radius= 2
block gridpoint fix temperature    0.0  ...
  range ann center (0,0) rad 1.99 2.01
; thermal histories to check thermal equilibrium
block gridpoint history temperature 1.25,1.25
block gridpoint history temperature 1.0,1.0
block thermal history time-total
; mechanical histories at different joints to check mech. equil.
block contact history stress-normal (1.25,0)
block contact history stress-normal (1.25,1.25)
block contact history stress-normal (0,1.25)
```

```
block contact history stress-shear (1.25,0)
block contact history stress-shear (1.25,1.25)
block contact history stress-shear (0,1.25)
; run thermal problem until equilibrium (explicit procedure)
block thermal cycle temp=15000 step=10000 tol .001
model save 'cy1.sav'

;
; then run mechanical problem
block solve ratio 1e-6
model save 'cy2.sav'

;
; --- fish constants ---
fish def constants
   c_b = 2.
   eps = 1.e-4
   xtol = 0.01
   yp_tol = 0.05
   c_g = 28e9       ; shear modulus
   c_k = 48e9       ; bulk  modulus
   c_al = 5.4e-6    ; coefficient of thermal expansion
   t1 = 100.        ; boundary temperature
   oc1 = 1. / math.ln(c_b)
   oc2 = 1. / (c_b * c_b - 1.)
   oc3 = 0.5 * (c_k - c_g * 2. / 3.)/(c_k + c_g / 3.)
   c_mmu = c_g * (3. * c_k * c_al) / (c_k + 4. * c_g / 3.)
   tab1 = 1        ; numerical  temperature
   tab2 = 2        ; analytical temperature
   tab3 = 3        ; numerical  radial     stress
   tab4 = 4        ; analytical radial     stress
   tab5 = 5        ; numerical  tangential stress
   tab6 = 6        ; analytical tangential stress
   tab7 = 7        ; numerical  axial      stress
   tab8 = 8        ; analytical axial      stress
end
@constants
fish def num_solt
   ib = block.head
   loop while ib # 0
     ig = bl.gp(ib)
     loop while ig # 0
       if bl.gp.pos.y(ig) < eps then
          x = bl.gp.pos.x(ig)
          table(tab1,x) = gp_temp(ig) / t1
       endif
```

```
        ig = bl.gp.next(ig)
      endloop
      ib = bl.next(ib)
    endloop
end
fish def ana_solt
    nn = 0
    ib = block.head
    loop while ib # 0
      ig = bl.gp(ib)
      loop while ig # 0
        if bl.gp.pos.y(ig) < eps then
           x = bl.gp.pos.x(ig)
           table(tab2,x) = math.ln(c_b / x) * oc1
           nn = nn + 1
        endif
        ig = bl.gp.next(ig)
      endloop
      ib = bl.next(ib)
    endloop
end
;
fish def num_solst                  ; table tab1 must be available
    ns = 1
    nz = 0
    loop while ns < nn
       x = (table.x(tab1,ns) + table.x(tab1,ns+1)) * 0.5
       xp_tol = x + xtol
       xm_tol = x - xtol
       ib = block.head
       loop while ib # 0
         iz = bl.zone(ib)
         loop while iz # 0
             if bl.zone.pos.x(iz) < xp_tol then
               if bl.zone.pos.x(iz) > xm_tol then
                 if bl.zone.pos.y(iz) < yp_tol then
                   nz = nz + 1
                   xc = bl.zone.pos.x(iz)
                   yc = bl.zone.pos.y(iz)
                   ra2 = xc*xc + yc*yc
                   ra = math.sqrt(ra2)
                   table.x(tab3,nz) = ra
                   xstr = bl.zo.str.xx(iz)*xc*xc
                   ystr = bl.zo.str.yy(iz)*yc*yc
                   xystr = 2.*bl.zo.str.xy(iz)*xc*yc
                   val = (xstr + ystr + xystr)/ra2
```

```
                table.y(tab3,nz) = val / (c_mmu * t1)
                table.x(tab5,nz) = ra
                xstr = bl.zo.str.xx(iz)*yc*yc
                ystr = bl.zo.str.yy(iz)*xc*xc
                xystr = 2.*bl.zo.str.xy(iz)*xc*yc
                val = (xstr + ystr - xystr)/ra2
                table.y(tab5,nz) = val / (c_mmu * t1)
                table.x(tab7,nz) = ra
                val = bl.zo.str.zz(iz)
                table.y(tab7,nz) = val / (c_mmu * t1)
              endif
            endif
          endif
          iz = bl.zone.next(iz)
        end_loop
        ib = bl.next(ib)
      end_loop
      ns = ns + 1
   end_loop
end
fish def ana_solst                ; table tab1 must be available
   ns = 1
   nz = 0
   loop while ns < nn
      x = (table.x(tab1,ns) + table.x(tab1,ns+1)) * 0.5
      xp_tol = x + xtol
      xm_tol = x - xtol
      ib = block.head
      loop while ib # 0
        iz = bl.zone(ib)
        loop while iz # 0
            if bl.zone.pos.x(iz) < xp_tol then
              if bl.zone.pos.x(iz) > xm_tol then
                if bl.zone.pos.y(iz) < yp_tol then
                  nz = nz + 1
                  xc = bl.zone.pos.x(iz)
                  yc = bl.zone.pos.y(iz)
                  ra = math.sqrt(xc*xc + yc*yc)
                  table.x(tab4,nz) = ra
                  val = c_b / ra
                  table.y(tab4,nz) = -(math.ln(val)*oc1 ...
                                      -(val*val-1.)*oc2)
                  table.x(tab6,nz) = ra
                  table.y(tab6,nz) = -((math.ln(val)-1.)*oc1 ...
                                       +(val*val+1.)*oc2)
                  table.x(tab8,nz) = ra
```

```
                    table.y(tab8,nz) = -((2.*math.ln(val)-oc3)*oc1 ...
                                         +2.*oc3*oc2)
                  endif
                endif
              endif
              iz = bl.zone.next(iz)
          end_loop
          ib = bl.next(ib)
       end_loop
       ns = ns + 1
    end_loop
end
model save 'cy3.sav'

;
@num_solt
@ana_solt
@num_solst
@ana_solst
table 1 label 'Temperature - UDEC'
table 2 label 'Temperature - Analytic'
table 3 label 'Radial Stress - UDEC'
table 4 label 'Radial Stress - Analytic'
table 5 label 'Tangential Stress - UDEC'
table 6 label 'Tangential Stress - Anal'
table 7 label 'Axial Stress - UDEC'
table 8 label 'Axial Stress - Analytic'
model save 'cy4.sav'
```

Numerical and analytical results are compared in Figures 3.18 through 3.21. The figures show plots of tables for temperature and stress distributions through the cylinder at steady state. In each figure, the numerical values are plotted as the odd numbered table, and the analytical values are plotted as the even numbered table. The plotted values are normalized. Temperature is normalized by dividing by T_a, and stress is normalized by dividing by mGT_a. Figure 3.18 shows the temperature distribution at steady state for the numerical and analytical solutions. Comparisons of results for radial, tangential and axial stress distributions at steady state are provided in Figures 3.19, 3.20 and 3.21, respectively.

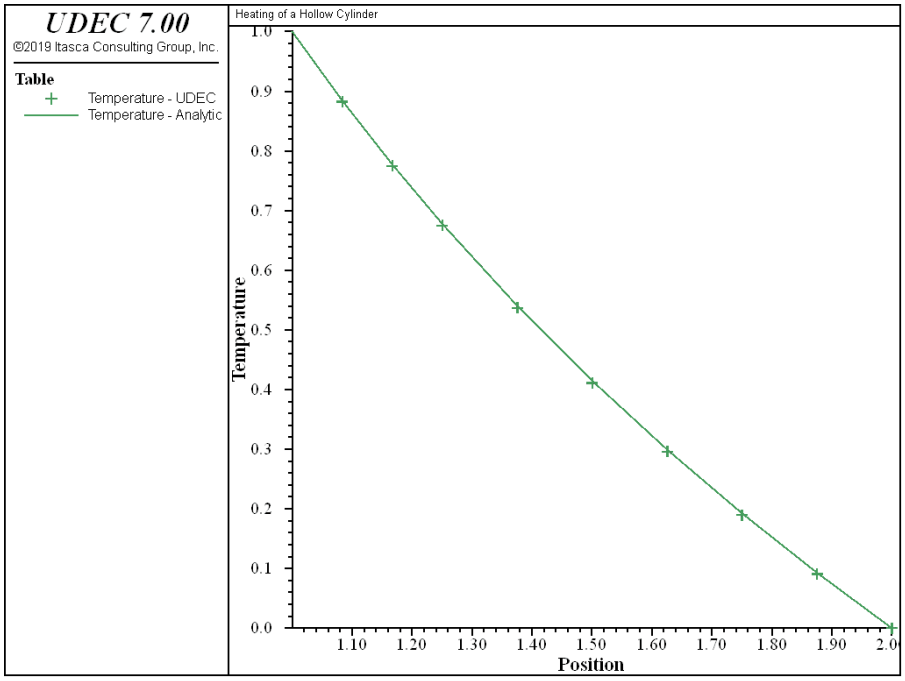

Figure 3.18 Temperature distribution at steady state for heating of a hollow cylinder

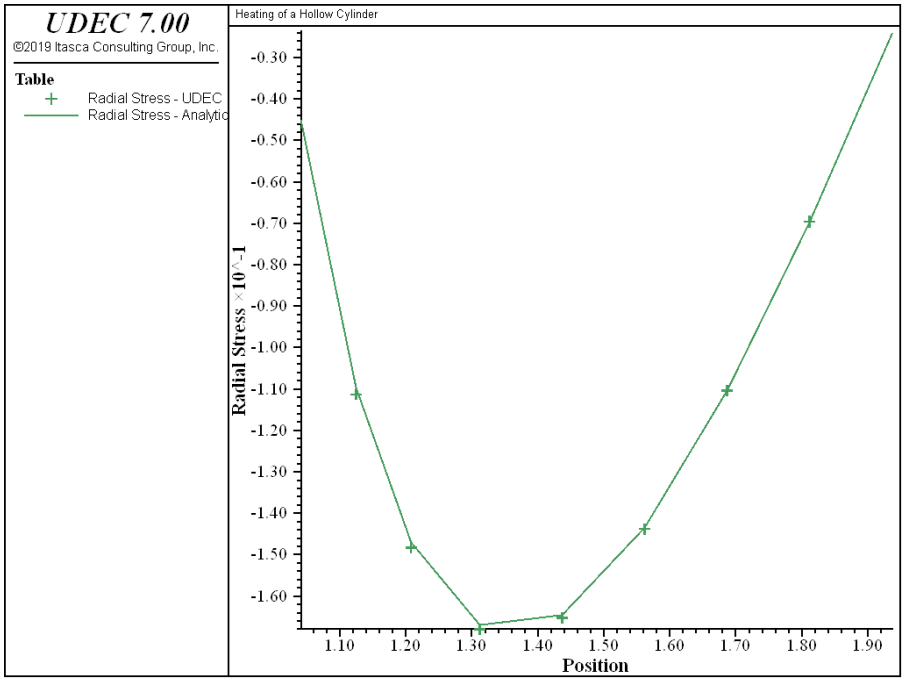

Figure 3.19 Radial stress distribution at steady state for heating of a hollow cylinder

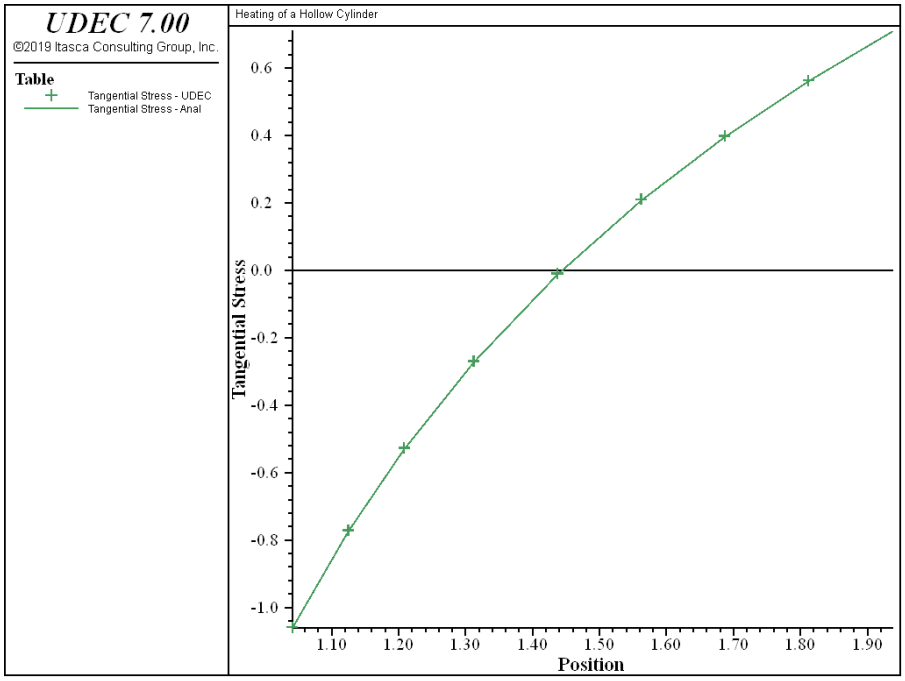

Figure 3.20 Tangential stress distribution at steady state for heating of a hollow cylinder

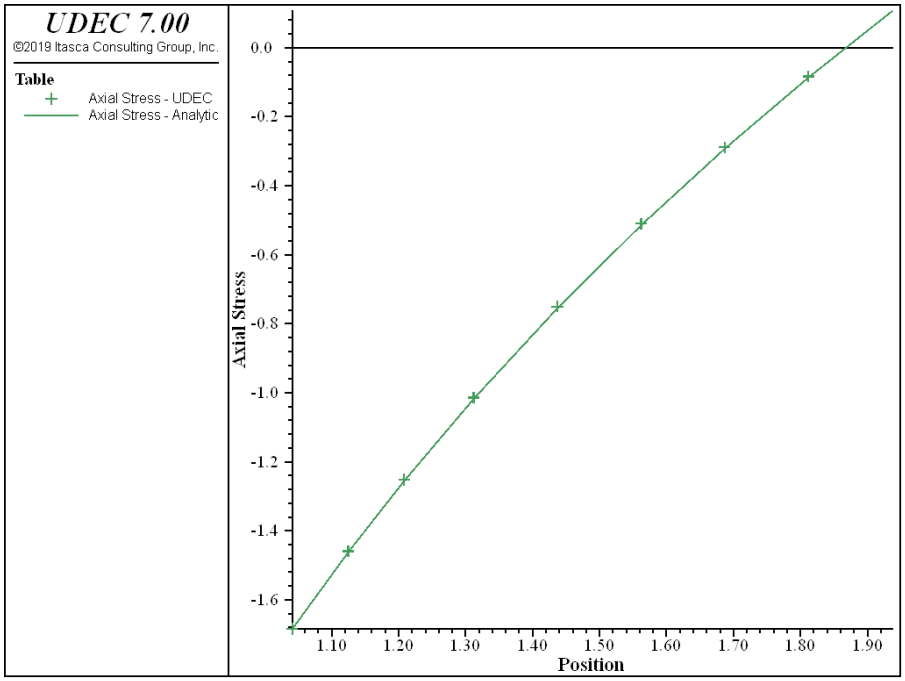

Figure 3.21 Axial stress distribution at steady state for heating of a hollow cylinder

3.6.4 Infinite Line Heat Source in an Infinite Medium

An infinite line heat source with a constant heat-generating rate is located in an infinite elastic medium with constant thermal properties. Nowacki (1962) provides the solution to this problem for the transient values of temperature, radial and tangential stress and radial displacement:

$$\frac{T}{a} = \frac{1}{4\pi} E_1\ (\xi) \tag{3.31}$$

$$\frac{\sigma_r}{bG} = \frac{1}{-4\pi} \left[E_1\ (\xi) + \frac{1 - e^{-\xi}}{\xi} \right] \tag{3.32}$$

$$\frac{\sigma_t}{bG} = \frac{1}{-4\pi} \left[E_1\ (\xi) - \frac{1 - e^{-\xi}}{\xi} \right] \tag{3.33}$$

$$\frac{u_r}{bL} = \frac{1}{8\pi}\, r \left[E_1\ (\xi) + \frac{1 - e^{-\xi}}{\xi} \right] \tag{3.34}$$

where ξ $= \frac{r^2}{4\kappa t}$;

r = radial distance to the line source;

κ $= \frac{k}{\rho C_p}$;

a $= \frac{q}{k}$;

b $= \alpha\ a \frac{9K}{3K+4G}$;

L = unit length; and

$E_1(\xi) = \int_\xi^\infty \frac{e^{-u}}{u} du$ is the exponential integral.

The material properties and initial and boundary conditions for this example are defined as follows.

Material Properties

density (ρ)	2000 kg/m^3
shear modulus (G)	30 GPa
bulk modulus (K)	50 GPa
specific heat (C_p)	1000 J/kg°C
thermal conductivity (k)	4 W/m°C
linear thermal expansion coefficient (α)	5×10^{-6}/°C

Initial/Boundary Conditions

initial uniform temperature	0°C
initial stress state	no stresses

Line Heat Source

energy release per unit length (Q)	1600 W/m

It is assumed that the material properties are temperature-independent, the thermal output of the source is constant (no decay), and the heat line source is of infinite length.

The *UDEC* model for this problem is a quarter-section of a cylindrical disk with a hole in the center. The axis of the line heat source coincides with the centroid of the disk. The zoning in the model is radially graded in the xy-plane by cutting the original block with a series of construction arcs; each arc has a radius that is 1.1 times larger than the previous arc. The model is shown in Figure 3.22. A close-up view of the block zoning near the heat source is shown in Figure 3.23.

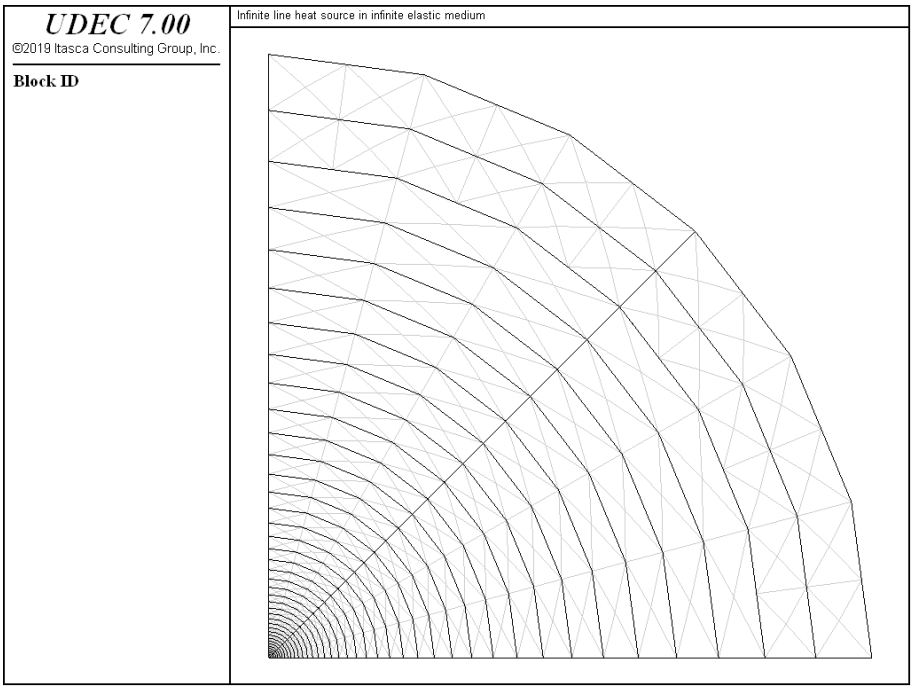

Figure 3.22 UDEC grid for an infinite line heat source

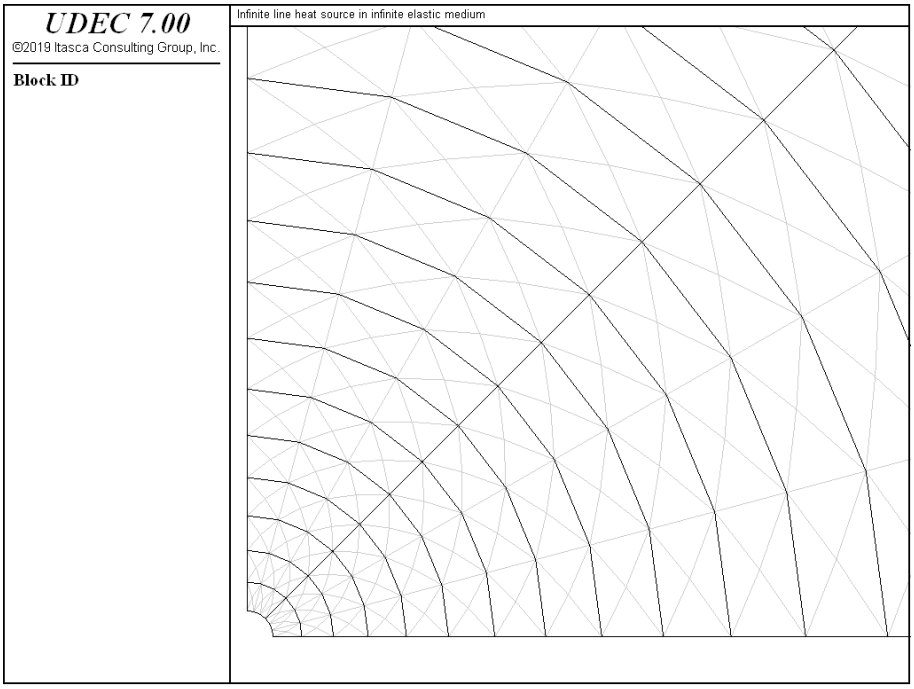

Figure 3.23 Close-up view of zoning in blocks

The line heat source is simulated by a constant heat flux applied at the inner hole boundary of the disk. The line heat source is assumed to have a fictitious radius of R = 1.0 m, so that the applied flux will be

$$\text{Flux} = q = \frac{Q}{2\pi R} = 254.65 \text{ w/m}^2$$

The other boundaries of the model are kept adiabatic to represent thermal symmetry planes. The disk is extended to a radius of 500 m to simulate infinity. The far boundary is mechanically fixed; the boundaries along the x-axis and y-axis are fixed to represent shear-free symmetry planes.

The problem is first solved thermally to an age of one year using the implicit solution algorithm, and then stepped to mechanical equilibrium. The *UDEC* data file is listed in Example 3.4.

The dimensionless form of the analytical solutions in Eqs. (3.31) to (3.34) are programmed as *FISH* functions in Example 3.4. The analytical and numerical values can then be compared directly in tables. The analytical solutions for temperature and radial displacement are programmed in *FISH* function **ana_soltu**, and for radial and tangential stresses in **ana_solst**. The exponential integral function used in the analytical solutions is programmed as a separate *FISH* function contained in file "EXP_INT.FIS" (see Example 3.5). The dimensionless values for the numerical results for temperature and displacement are calculated in *FISH* function **num_soltu**, and for radial and tangential stresses in **num_solst**. The numerical values for dimensionless temperature, radial stress,

tangential stress and radial displacement are stored in Tables 1, 3, 5 and 7, respectively. The analytical values for dimensionless temperature, radial stress, tangential stress and radial displacement are stored in Tables 2, 4, 6 and 8, respectively.

Example 3.4 Infinite line heat source in an infinite medium

```
model new
;file: linesource.dat
model title 'Infinite line heat source in infinite elastic medium'
block config thermal
block tolerance corner-round-length 1E-3
block tolerance minimum-edge-length 2E-3
block create polygon 0 0 0 500 500 500 500 0
;Name:arc_cut
;Input:xcut/float/1.0/x-coord of cut
;Input:narc/int/6/number of segments in arc
;Input:ntot/int/48/number of arcs
;Input:rat/float/1.1/geometric ratio
fish define arc_cut
   nc = 1
   xloc = xcut
   numarc = narc
   loop while nc < ntot
      command
         block cut arc 0 0 @xloc 0 90 @numarc
      endcommand
      xcut = rat * xcut
      xloc = xloc + xcut
      nc = nc + 1
   endloop
end
fish set @xcut=1.0
fish set @narc=6
fish set @ntot=48
fish set @rat=1.1
@arc_cut
block delete range annulus center 0 0 rad 450 800
block delete range annulus center 0 0 rad 0 1
block cut joint-set angle 45 spacing 800 origin 0 0
block contact join by-contact
block zone gen quad 100.0
;
; set material properties
block zone group 'block'
block zone cmodel assign elastic density 2E3 bulk 5E10 shear 3E10 ...
  cond 4 specheat 1E3 thexp 5E-6 range group 'block'
```

```
;
; set boundary conditions
block gridpoint apply velocity-y 0 range pos-x 0 500 pos-y -0.1 0.1
block gridpoint apply velocity-x 0 range pos-x -0.1 0.1 pos-y 0 500
block gridpoint apply velocity-normal 0 ...
  range annulus center 0 0 radius 440 500
; apply heat source for 1600 W/m
block edge apply flux 254.65 0 range annulus center 0 0 radius 0.9 1.1
;
; thermal histories to check thermal equilibrium
block gridpoint history temperature 1.25 1.25
block gridpoint history temperature 5.0 5.0
block thermal history time-total
;
; mechanical histories at different joints to check mech. equil.
block contact history stress-normal 1.25 0
block contact history stress-normal 1.25 1.25
block contact history stress-normal 0 1.25
block contact history stress-shear 1.25 0
block contact history stress-shear 1.25 1.25
block contact history stress-shear 0 1.25
;
model save 'line.sav'
;
;
; run thermal problem for 1 year of heating (implicit procedure)
block thermal timestep 6480.0
block thermal cycle implicit step 4800 temperature-change 500000.0
model save 'line_th.sav'
;
;
model restore 'line_th.sav'
;
;
; then run mechanical problem
block contact tolerance overlap 0.1
block mechanical timestep-factor 0.1 0.5
block mechanical damping global
block cycle 9000
;
model save 'line_1yr.sav'
;
;
; ------------------------------------------------------------
; line source in infinite medium
; comparison of numerical and analytical solutions
```

```
; ------------------------------------------------------------
; --- fish functions ---
fish define cons
;   t_time   = 3.11e7
   xtol     = 0.3
   yp_tol   = 5.0
   eps  = 1.e-4
   c_g  = 3.e10        ; shear modulus
   c_k  = 5.e10        ; bulk  modulus
   c_al = 5.e-6        ; coefficient of thermal expansion
   c_tk = 4.           ; conductivity
   c_cp = 1e3          ; specific heat
   q_q  = 1600.        ; line source intensity
   c_density = 2.e3
   kappa = c_tk / (c_density * c_cp)
   o4c = 1. / (4. * kappa)
   o4p = 1. / (4. * math.pi)
   o8p = o4p * 0.5
   val = c_k / c_g
   c_nu = (3.*val-2.)/(6.*val+2.)
   c_eta = c_al * c_g * (1.+c_nu)/(1.-c_nu)
   a_a = q_q / c_tk
   b_b = c_eta * a_a
   c_c = b_b * 1.0 / c_g
   a_a = 1. / a_a
   b_b = 1. / b_b
   c_c = 1. / c_c
   tab1 = 1        ; numerical  temperature
   tab2 = 2        ; analytical temperature
   tab3 = 3        ; numerical  radial     stress
   tab4 = 4        ; analytical radial     stress
   tab5 = 5        ; numerical  tangential stress
   tab6 = 6        ; analytical tangential stress
   tab7 = 7        ; numerical  radial     displacement
   tab8 = 8        ; analytical radial     displacement
end
@cons
call 'exp_int.fis'
fish define num_soltu
   nn = 0
   ib = block.head
   loop while ib # 0
     ig = block.gp(ib)
     loop while ig # 0
       if block.gp.pos.y(ig) < eps then
         x = block.gp.pos.x(ig)
```

```
            if x > 0.0 then
              table(tab1,x) = block.gp.temp(ig) * a_a
              table(tab7,x) = block.gp.disp.x(ig) * c_c
              nn = nn + 1
            end_if
          end_if
          ig = block.gp.next(ig)
        endloop
        ib = block.next(ib)
     end_loop
  end
  @num_soltu
  fish define ana_soltu
     ib = block.head
     loop while ib # 0
       ig = block.gp(ib)
       loop while ig # 0
         if block.gp.pos.y(ig) < eps then
            x = block.gp.pos.x(ig)
            if x > 0.0 then
              e_val = x * x * o4c / block.thermal.time.total
              val = exp_int
              table(tab2,x) = val * o4p
              table(tab8,x) = (val + (1.-math.exp(-e_val))/e_val) * x * o8p
            end_if
         endif
         ig = block.gp.next(ig)
       endloop
       ib = block.next(ib)
     end_loop
  end
  ;
  fish define num_solst
     ns = 2
     nz = 0
     loop while ns < nn
        x = (table.x(tab1,ns) + table.x(tab1,ns+1)) * 0.5
        if ns > 10 then
           xtol = 1.0
        endif
        if ns > 50 then
           xtol = 5.0
        endif
        xp_tol = x + xtol
        xm_tol = x - xtol
        ib = block.head
```

```
    loop while ib # 0
      iz = block.zone(ib)
      loop while iz # 0
          if block.zone.pos.x(iz) < xp_tol then
            if block.zone.pos.x(iz) > xm_tol then
              nzgp = 0
              igz1 = block.zone.gp(iz,1)
              ygz1 = block.gp.pos.y(igz1)
              if ygz1 < eps then
                 nzgp = nzgp + 1
              endif
              igz2 = block.zone.gp(iz,2)
              ygz2 = block.gp.pos.y(igz2)
              if ygz2 < eps then
                 nzgp = nzgp + 1
              endif
              igz3 = block.zone.gp(iz,3)
              ygz3 = block.gp.pos.y(igz3)
              if ygz3 < eps then
                 nzgp = nzgp + 1
              endif
              if nzgp = 2 then
                nz = nz + 1
                xc = block.zone.pos.x(iz)
                yc = block.zone.pos.y(iz)
                ra2 = xc*xc + yc*yc
                ra = math.sqrt(ra2)
                xstr = block.zone.stress.xx(iz)*xc*xc
                ystr = block.zone.stress.yy(iz)*yc*yc
                xystr = 2.*block.zone.stress.xy(iz)*xc*yc
                val = (xstr + ystr + xystr)/ra2
                table(tab3,ra) = val * b_b
                xstr = block.zone.stress.xx(iz)*yc*yc
                ystr = block.zone.stress.yy(iz)*xc*xc
                xystr = 2.*block.zone.stress.xy(iz)*xc*yc
                val = (xstr + ystr + xystr)/ra2
                table(tab5,ra) = val * b_b
              endif
            endif
          endif
          iz = block.zone.next(iz)
      end_loop
      ib = block.next(ib)
    end_loop
    ns = ns + 1
end_loop
```

```
end
fish define ana_solst
   ns = 2
   nz = 0
   loop while ns < nn
      x = (table.x(tab1,ns) + table.x(tab1,ns+1)) * 0.5
      if ns > 10 then
         xtol = 1.0
      endif
      if ns > 50 then
         xtol = 5.0
      endif
      xp_tol = x + xtol
      xm_tol = x - xtol
      ib = block.head
      loop while ib # 0
        iz = block.zone(ib)
        loop while iz # 0
            if block.zone.pos.x(iz) < xp_tol then
              if block.zone.pos.x(iz) > xm_tol then
                nzgp = 0
                igz1 = block.zone.gp(iz,1)
                ygz1 = block.gp.pos.y(igz1)
                if ygz1 < eps then
                   nzgp = nzgp + 1
                endif
                igz2 = block.zone.gp(iz,2)
                ygz2 = block.gp.pos.y(igz2)
                if ygz2 < eps then
                   nzgp = nzgp + 1
                endif
                igz3 = block.zone.gp(iz,3)
                ygz3 = block.gp.pos.y(igz3)
                if ygz3 < eps then
                   nzgp = nzgp + 1
                endif
                if nzgp = 2 then
                  nz = nz + 1
                  xc = block.zone.pos.x(iz)
                  yc = block.zone.pos.y(iz)
                  ra2 = xc*xc + yc*yc
                  ra = math.sqrt(ra2)
                  e_val = ra2 * o4c / block.thermal.time.total
                  val1 = exp_int
                  val2 = (1. - math.exp(-e_val)) / e_val
                  table(tab4,ra) = - (val1 + val2) * o4p
```

```
                  table(tab6,ra) = - (val1 - val2) * o4p
                endif
              endif
            endif
            iz = block.zone.next(iz)
         end_loop
         ib = block.next(ib)
      end_loop
      ns = ns + 1
   end_loop
end
;
model save 'line_fish.sav'
;
;
@num_soltu
@ana_soltu
@num_solst
@ana_solst
table 1 label 'Temp. 1 Year - UDEC'
table 2 label 'Temp. 1 Year - Analytic'
table 3 label 'Radial Stress     - UDEC'
table 4 label 'Radial Stress     - Anal'
table 5 label 'Tangential Stress - UDEC'
table 6 label 'Tangential Stress - Anal'
table 7 label 'Radial Displacement - UDEC'
table 8 label 'Radial Displacement - Anal'
;
model save 'line_compare.sav'
return
```

Example 3.5 Exponential integral function

```
; --- Exponential integral E1(e_val) ---
;     Input : e_val
;
fish define exp_int
 if e_val < 0.0 then
    ii=io.out(' Argument of Exponential function must be positive')
    exit
 end_if
 if e_val = 0.0  then
    exp_int = 1.e12
    exit
 endif
 if e_val < 1. then
    e_e1 = ((.00107857 * e_val - 0.00976004) * e_val + .05519968) * e_val
    e_e1 = ((e_e1              -  .24991055) * e_val + .99999193) * e_val
    exp_int = e_e1 - .57721566 - math.ln(e_val)
 else
    e_e1 = .250621 + e_val * (2.334733 + e_val)
    e_e1 = e_e1 / (1.681534 + e_val * (3.330657 + e_val))
    exp_int = e_e1 * math.exp(-e_val) / e_val
 end_if
end
;
;
;
```

The results for temperature, radial displacement, and radial and tangential stress distributions at 1 year are presented in the table plots in Figures 3.24 through 3.26. The differences between numerical and analytical values are generally within 10%. Improved agreement can be expected as the block rounding length is decreased. The outer boundary also has an influence on the numerical results farther from the source.

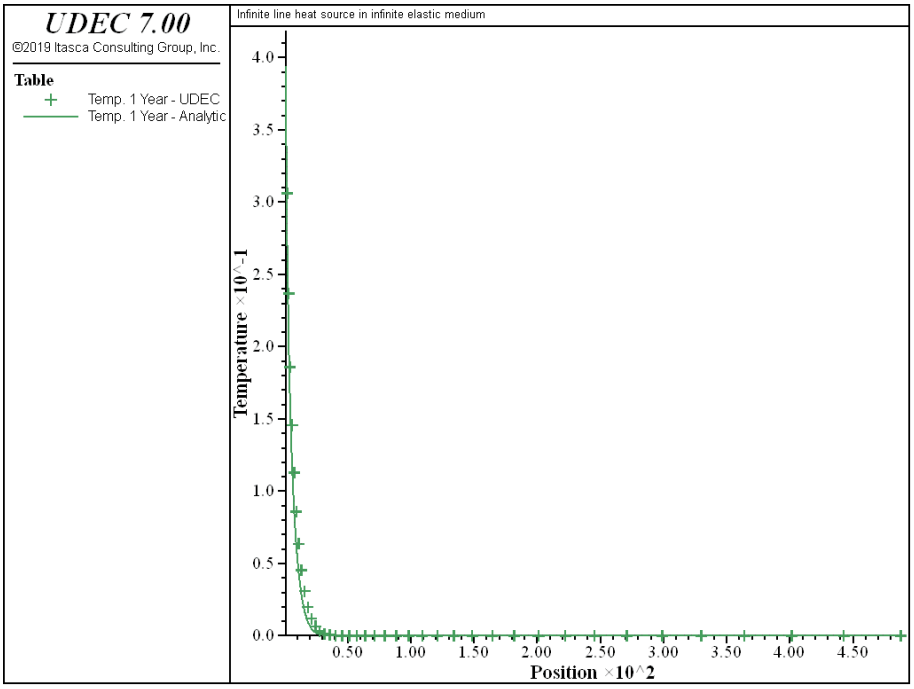

Figure 3.24 Temperature distribution at 1 year

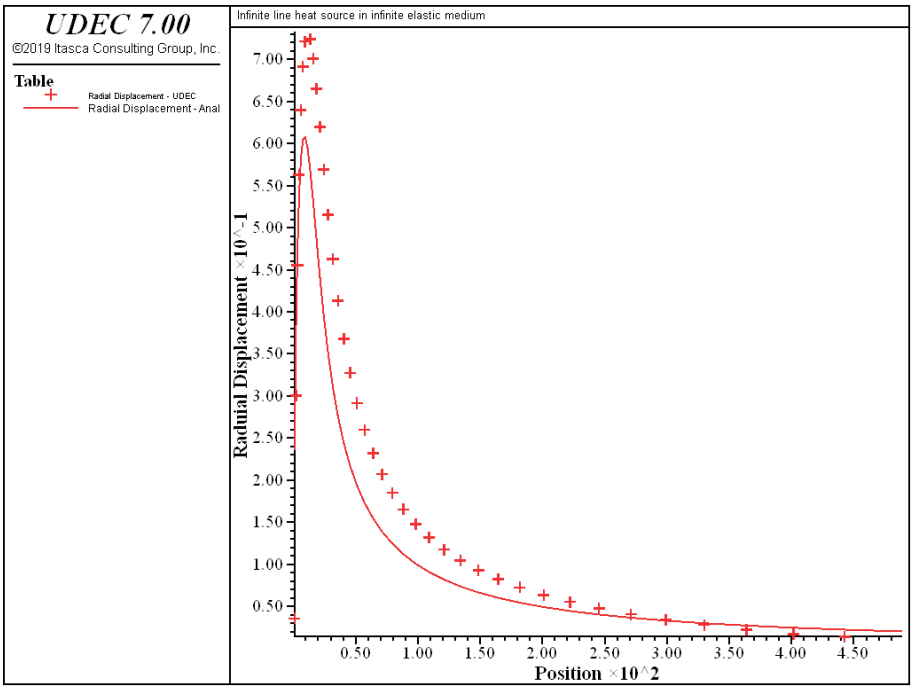

Figure 3.25 Radial displacement distribution at 1 year

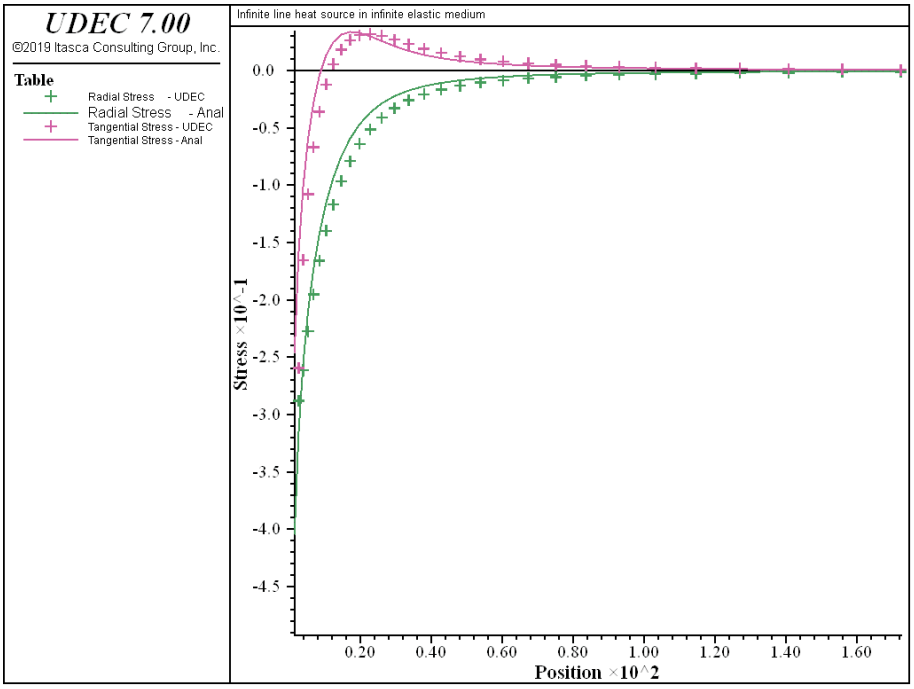

Figure 3.26 Radial and tangential stress distributions at 1 year

3.7 References

Carslaw, H. S., and J. C. Jaeger. ***Conduction of Heat in Solids***. London: Oxford University Press (1959).

Karlekar, B. V., and R. M. Desmond. ***Heat Transfer: Solutions Manual***, 2nd ed. West Publishing Co. (May 1982).

Nowacki, W. ***Thermoelasticity***. Addison-Wesley (1962).

Ozisik, M. N. ***Heat Conduction***. John Wiley & Sons (1980).

Schneider, P. J. ***Conduction Heat Transfer***. Cambridge, Mass.: Addison-Wesley (1955).

Timoshenko, S. P., and J. N. Goodier. ***Theory of Elasticity***. New York: McGraw-Hill (1970).