Chapter 7
Use of Commands in ANSYS

The distinct differences between the two modes of ANSYS usage, i.e., the *Graphical User Interface* (*GUI*) and *Batch Mode*, are covered briefly in Chap. 2, and the most common operations within the *Preprocessor*, *Solution*, and *Postprocessors*, mainly using the *GUI*, are covered in Chap. 4 and 5. This chapter is devoted to using the *Batch Mode* of ANSYS, which is the method preferred by advanced ANSYS users.

As mentioned in Chap. 2, every action taken by the user within the ANSYS GUI platform has an equivalent ANSYS *command*. Using ANSYS through the *Batch Mode* involves text (ASCII) files with specific ANSYS *commands*. These commands, along with specific rules, form a special programming language, *ANSYS Parametric Design Language*, or *APDL*, which utilizes concepts and structures very similar to common scientific programming languages such as BASIC, FORTRAN, etc. Using the *APDL*, the user can create (a) an *Input File* to solve a specific problem and (b) *Macro File(s)* that act as special functions, accepting several arguments as input. In either case, each line consists of a single command, and the lines are executed sequentially.

The basic ANSYS commands, operators, and functions are discussed in the following sections. After solving a simple problem by using the *Batch Mode*, more advanced APDL features are covered. The *Batch Mode* command files for each example problem included in this book are given on the accompanying CD-ROM.

7.1 Basic ANSYS Commands

There are around 1500 ANSYS commands, each with a specific syntax and function. It is impractical (and perhaps impossible) for the user to learn the use of all of the commands. This apparent obstacle is overcome by using the ANSYS *Help System*, accessible from within the program, which is covered in Sect. 2.7. However, the solution of a typical problem often involves a limited number of commonly used commands. A selection of these common commands is presented in tabular form in

The online version of this book (doi: 10.1007/978-1-4939-1007-6_7) contains supplementary material, which is available to authorized users

E. Madenci, I. Guven, *The Finite Element Method and Applications in Engineering Using ANSYS®,* DOI 10.1007/978-1-4899-7550-8_7

Table 7.1 Session and database commands

Command	Description
/CLEAR	Clear the database (and memory)
/PREP7	Enter the *Preprocessor*
/SOLU	Enter the *Solution*
/POST1	Enter the *General Postprocessor*
/POST26	Enter the *Time History Postprocessor*
FINISH	Exit the current processor; go to *Begin* level
/EOF	Marks the end of file (stop reading)
/FILNAME	Specify *jobname*
HELP	Display help pages related to the command
SAVE	Save the database
RESUME	Resume from an existing database
KSEL, LSEL,ASEL, VSEL, NSEL, ESEL,CMSEL	Select keypoints, lines, areas, volumes, nodes, elements, and components
ALLSEL	Select all entities
CLOCAL, LOCAL	Define local coordinate systems
CSYS	Switch between coordinate systems
!	Start comment—ANSYS ignores the characters to the right of the exclamation mark

Table 7.2 APDL commands

Command	Description
*AFUN	Switch between degrees and radians to be used for angles
*GET	Store model or result information into parameters
*VWRITE	Write formatted output to external files
*DO,*ENDDO	Beginning and ending of do loops
*IF,*ELSE,*ELSEIF, *ENDIF	Commands related to IF-THEN-ELSE blocks
*SET	Define parameters

this section. Within the context of this book, they are grouped into the following six categories:

- Session and Database Commands (Table 7.1).
- APDL Commands (Table 7.2).
- Preprocessor Solid Model Generation Commands (Table 7.3).
- Preprocessor Meshing Commands (Table 7.4).
- Solution Commands (Table 7.5).
- General Postprocessor Commands (Table 7.6).

In Tables 7.1–7.6, the first column gives the command and the corresponding description is given in the second column. With the exception of some APDL commands, the commands can also be issued as a command line input in the *Input*

Table 7.3 Preprocessor solid model generation commands

Command	Description
Command	Description
BLC4	Create rectangular area or prism volume
CYL4	Create circular area or cylindrical volume
K, L, A, AL, V, VA	Create keypoints, lines, areas, and volumes
LARC	Create circular arc
SPLINE, BSPLIN	Create line through spline fit to keypoints
ADRAG	Create an area by dragging a line along a path
VRAG	Create a volume by dragging an area along a path
VEXT	Create a volume by extruding an area
AAD, VADD	Add areas and volumes
LGLUE, AGLUE, VGLUE	Glue lines, areas, and volumes
LOVLAP, AOVLAP, VOVLAP	Overlap lines, areas, and volumes
CM	Create components
KDELE, LDELE, ADELE, VDELE, CMDELE	Delete keypoints, lines, areas, volumes, and components
KPLOT, LPLOT, APLOT, VPLOT	Plot keypoints, lines, areas, and volumes in the *Graphics Window*
KLIST, LLIST, ALIST, VLIST, CMLIST	List keypoints, lines, areas, volumes, and components

Table 7.4 Preprocessor meshing commands

Command	Description
ET	Specify element type
R	Specify real constants
MP	Specify material properties
N	Create nodes
E	Create elements
TYPE	Specify default element type attribute number
REAL	Specify default real constant set attribute number
MAT	Specify default material property set attribute number
LMESH, AMESH,VMESH	Mesh the lines, areas, and volumes
LCLEAR, ACLEAR, VCLEAR	Clear the mesh from lines, areas, and volumes (deletes the nodes and elements attached to those entities)
LESIZE	Specify number of elements or element sizes along selected lines
MSHKEY	Specify whether to use mapped or free meshing
NDELE, EDELE	Delete nodes and elements
NPLOT, EPLOT	Plot nodes and elements in the *Graphics Window*
NLIST, ELIST	List nodes and elements

Table 7.5 Solution commands

Command	Description
SOLVE	Start solution for the current load step
LSSOLVE	Start solution from multiple load step files
D	Specify DOF constraints on nodes
F	Specify concentrated load boundary conditions on nodes
SF, SFE,SFL, SFA	Specify surface (distributed) loads on nodes, elements, lines, and areas
BF, BFE	Specify body loads on nodes and elements
TUNIF	Specify uniform thermal load on all nodes
IC	Specify initial conditions
LSREAD, LSWRITE	Read from and write to load step files

Table 7.6 General postprocessor commands

Command	Description
FILE	Specify the results file for the results to be read from
SET	Specify the load step and substep numbers to be loaded
PLDISP	Plot deformed shape
PLNSOL	Plot contours of nodal solution
PLESOL	Plot contours of element solution
PRNSOL	List nodal solution items
PRESOL	List element solution items

Field in the ANSYS GUI. It is worth noting that some ANSYS commands are valid only in a specific processor or BEGIN level while the remaining ones are valid at all times. Most of the ANSYS commands require arguments separated by commas. For example, the syntax for the **K** command (to create keypoints) given in Table 7.3 is

```
K, NPT, X, Y, Z
```

where **NPT** is the keypoint number and **X**, **Y**, and **Z** are the *x*-, *y*-, and z-coordinates of the node,

As explained in Sect. 2.7, the help page related to the use of this command can be retrieved by issuing the following command line input in the *Input Field* in ANSYS:

```
HELP,K
```

This command brings up detailed information about the arguments.

Table 7.7 List of operators within ANSYS

Operator	Description
+	Addition
–	Subtraction
*	Multiplication
/	Division
**	Exponentiation
<	Less-than comparison
>	Greater-than comparison
=	Equal to (used in defining parameters)

Table 7.8 Selected ANSYS functions

Function	Description
ABS(X)	Absolute value of *X*
EXP(X)	Exponential of *X*
LOG(X)	Natural logarithm of *X*
LOG10(X)	Base 10 logarithm of *X*
SQRT(X)	Square root of *X*
NINT(X)	Nearest integer to *X*
RAND(X, Y)	Random number within the range *X*–*Y*
SIN(X), COS(X), TAN(X)	Sine, cosine, and tangent of *X*
SINH(X), COSH(X), TANH(X)	Hyperbolic sine, hyperbolic cosine, and hyperbolic tangent of *X*
ASIN(X), ACOS(X), ATAN(X)	Inverse sine, inverse cosine, and inverse tangent of *X*

Tables 7.1–7.6 serve as an introduction to the ANSYS commands. However, it is highly recommended that the user read the help pages before usage.

7.1.1 Operators and Functions

In the ANSYS Parametric Design Language (APDL), several fundamental mathematical operations can be utilized through the use of common operators and functions. A complete list of operators is given in Table 7.7. Table 7.8 lists selected mathematical functions available within APDL. Section 7.1.2 provides several examples demonstrating the definition and use of parameters in APDL. These examples are also useful in understanding the way mathematical operators and functions are used in ANSYS.

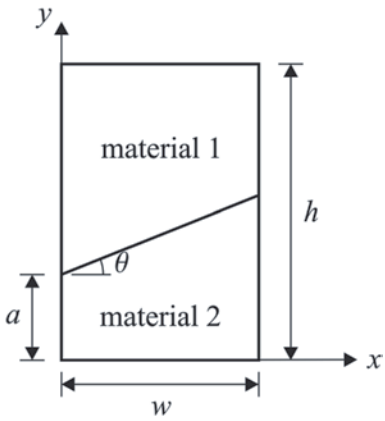

Fig. 7.1 A rectangular area consisting of two dissimilar materials

7.1.2 Defining Parameters

Parameters in APDL can be defined by using either the ***SET** command or the "equal to" sign (=). For example the parameter "**USRPRM**" can be defined to have the value 22 by either

```
*SET,USRPRM,22
```

or

```
USRPRM=22
```

The rules for naming of parameters are:

- The first character of a parameter name must be a letter.
- Within the parameter name, only letters, numbers, and the underscore character (_) are allowed.
- The maximum number of characters within a parameter name is 32.

The use of common mathematical operations and functions (Tables 7.7–7.8) in parameter definitions is illustrated in the example below. Similar input files for various examples considered in this book are also provided on the CD-ROM.

A rectangular area consisting of two dissimilar materials, shown in Fig. 7.1, has a width and height of w and h, respectively. The material interface starts on the left edge at point (0, a), with an inclination angle θ. Assuming the numerical values of $w=2$, $h=4$, $a=1$, and $\theta=30^{o}$, the following APDL block creates the solid model shown in Fig. 7.2:

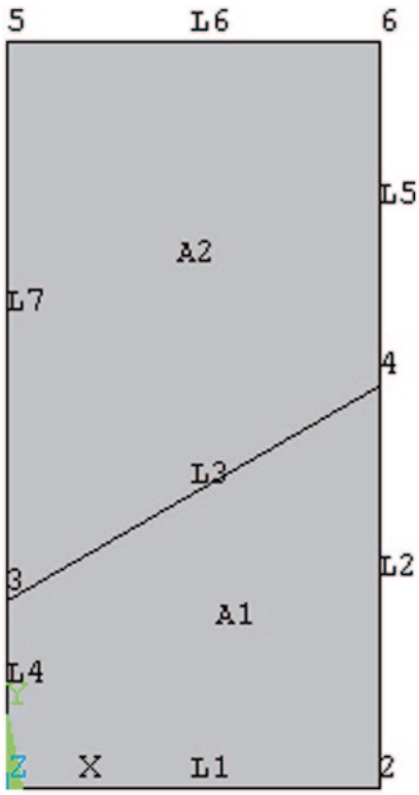

Fig. 7.2 ANSYS model created using of two dissimilar materials. using numerical values of $w=2$, $h=4$, $a=1$, and $\theta = 30°$

```
/PREP7                    ! ENTER PREPROCESSOR

*AFUN,DEG                 ! SWITCH TO DEGREES
W=2                       ! WIDTH
H=4                       ! HEIGHT
A=1                       ! Y-COORDINATE OF MATERIAL
                          !INTERFACE AT LEFT EDGE
THETA=30                  ! INCLINATION ANGLE
B= W*TAN(THETA)

! CREATE KEYPOINTS
K,1,0,0
K,2,W,0
K,3,0,A
K,4,W,A+B
K,5,0,H
K,6,W,H
```

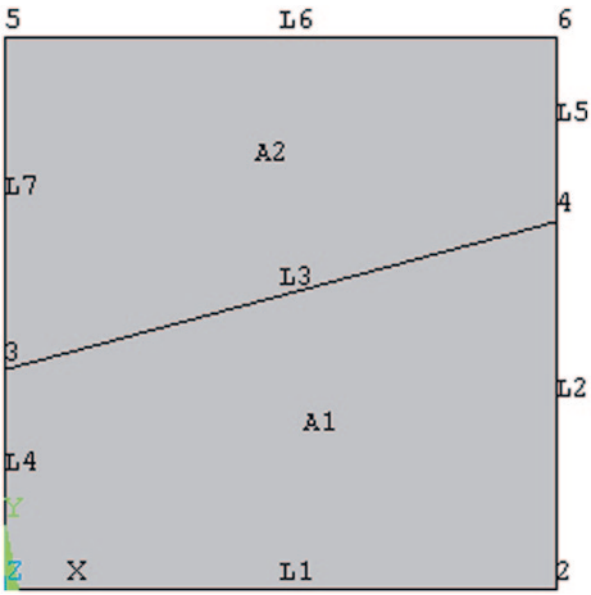

Fig. 7.3 ANSYS model created using numerical values of $w=5$, $h=5$, $a=2$, and $\theta=15°$

```
! CREATE LINES
L,1,2
L,2,4
L,4,3
L,1,3
L,4,6
L,6,5
L,3,5

! CREATE AREAS
AL,1,2,3,4
AL,3,5,6,7
```

Note that the distance b is calculated using a mathematical operator (*) and the function **TAN** to create *keypoint* 4. The input block can be saved as a text file, "*example.txt,*" in the *Working Directory* and read from within ANSYS using the following menu path:

Utility Menu > File > Read Input from

It is also possible to read input files by issuing the **/INPUT** command in the *Input Field* in ANSYS GUI as follows:

```
/INPUT,EXAMPLE,TXT
```

Convenience in using the *Batch Mode* is demonstrated by modifying the length and angle parameters defined in the previous example. Fig. 7.3 shows the solid model generated using $w=5$, $h=5$, $a=2$, and $\theta=15^{o}$ and Fig. 7.4 shows the one using $w=1$, $h=5$, $a=2$, and $\theta=60°$.

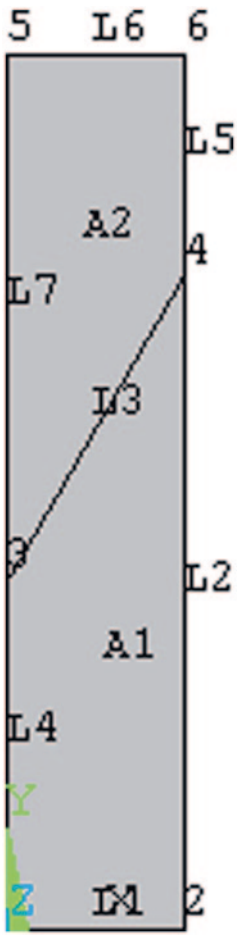

Fig. 7.4 ANSYS model created using numerical values of $w=1$, $h=5$, $a=2$, and $\theta=60°$

It is worth noting that if a parameter is redefined in the input file, the new value is ***not*** reflected in the entities or parameters defined previously. For example, *keypoint* 4 is created using parameters *w*, *a*, and *b*. If the parameter *w* is redefined (from 2 to 5) after the creation of *keypoint* 4 as shown below,

the new value of *w* is not reflected in the definition of *keypoint* 4 and the *x*-coordinate of *keypoint* 4 remains as 2.

```
W=2
A=1
THETA=30
B= W*TAN(THETA)
K,4,W,A+B ! CREATE KEYPOINT 4
W=5
```

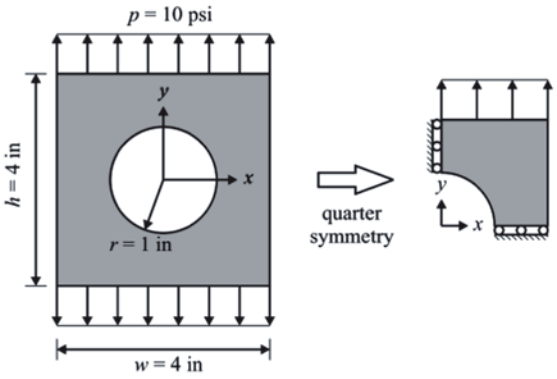

Fig. 7.5 A thin, square structure with a centric circular hole subjected to tensile loading in the *y*-direction (*left*); due to symmetry, only one-fourth of the structure is modeled (*right*)

7.2 A Typical Input File

Typical steps involved in solving an engineering problem are listed in Sect. 5.1. A similar data structure is observed in the *Input Files* used in the *Batch Mode*. In order to demonstrate the use of the *Batch Mode* for a complete analysis, a thin, square structure with a centric circular hole subjected to tensile loading in the *y*-direction, as shown in Fig. 7.5, is considered.

The length of the square and the radius of the circular hole are $w=4$ in. and $r=1$ in., respectively, and the thickness of the structure is $t=0.1$ in. The geometry and material possess quarter-symmetry, therefore only one-fourth of the domain is modeled. Because the thickness is significantly smaller than the in-plane dimensions, a plane stress assumption is used. The elastic modulus and Poisson's ratio are $E=30\times10^6$ psi and $\nu=0.30$, respectively. The distributed tensile load is specified as $q=10$ lbs/in, and it is applied in the form of pressure loading with $q=-10$. The corresponding pressure is input as $q=-10$. The analysis is demonstrated by utilizing two separate solid modeling approaches: *Bottom-up* and *Top-down*.

The *Input File* below uses the *Bottom-up* approach, which starts building the model with *keypoints*, then *line* from *keypoints,* and, finally, *areas* using *lines* (explanations are given along with the commands; the commands between the dashed lines correspond to the *Bottom-up* approach in solid modeling).

```
/FILNAM,BOT-UP          !SPECIFY JOBNAME
/PREP7                  !ENTER PREPROCESSOR
ET,1,182                !SELECT ELEMENT TYPE AS
                        !PLANE182
KEYOPT,1,3,3            !SPECIFY PLANE STRESS WITH
                        !THICKNESS
R,1,0.1                 !SPECIFY REAL CONSTANT
                        !(THICKNESS)
MP,EX,1,30E6            !SPECIFY ELASTIC MODULUS
MP,NUXY,1,0.3           !SPECIFY POISSON'S RATIO
W=4                     !SIDE LENGTH OF SQUARE
R=1                     !HOLE RADIUS
P=10                    !APPLIED SURFACE LOAD
```

```
K,1,0,0                 !CREATE KEYPOINTS
K,2,R,0
K,3,W/2,0
K,4,0,R
K,5,0,W/2
K,6,W/2,W/2
L,2,3                   !CREATE LINES
L,3,6
L,6,5
L,5,4
LARC,4,2,1,R            !CREATE ARC
LESIZE,1,,,10           !SPECIFY NUMBER OF
LESIZE,4,,,10           !ELEMENTS ALONG LINES
LESIZE,2,,,15
LESIZE,3,,,15
LESIZE,5,,,30
AL,1,2,3,4,5            !CREATE AREA
LCCAT,2,3               !CONCATENATE LINES FOR MAPPED
                        !MESHING
```

```
MSHKEY,1                !USE MAPPED MESHING
AMESH,ALL               !MESH AREA
/SOLU                   !ENTER SOLUTION
NSEL,S,LOC,X,0          !SELECT NODES AT X = 0
D,ALL,UX                !SUPPRESS X-DISPLACEMENTS
                        !AT SELECTED NODES
NSEL,S,LOC,Y,0          !SELECT NODES AT Y = 0
D,ALL,UY                !SUPPRESS Y-DISPLACEMENTS
                        !AT SELECTED NODES
NSEL,S,LOC,Y,W/2        !SELECT NODES AT Y = W/2
SF,ALL,PRES,-P          !APPLY SURFACE LOADS
ALLSEL                  !SELECT EVERYTHING
SOLVE                   !SOLVE
/POST1                  !ENTER POSTPROCESSOR
PLDISP,2                !PLOT DEFORMED SHAPE
PLNSOL,S,Y              !PLOT STRESS IN Y-DIR
/EOF                    !MARK THE END OF FILE
```

Solid modeling using the *Top-down* approach to accomplish the same task is given below; the methods are interchangeable and the results are the same.

```
RECTNG,0,W/2,0,W/2      ! CREATE RECTANGLE
PCIRC,1                 ! CREATE CIRCLE
ASBA,1,2                ! SUBTRACT CIRCLE FROM
                        ! RECTANGLE
LSEL,S,LOC,X,0          ! SELECT LINES AT X = 0
LSEL,A,LOC,Y,0          ! ADD LINES AT Y = 0 TO THE
                        ! SELECTED SET
LESIZE,ALL,,,10         ! SPECIFY NUMBER OF ELEMENTS
                        ! ALONG LINES
LSEL,S,LOC,X,W/2        ! SELECT LINES AT X = W/2
LSEL,A,LOC,Y,W/2        ! ADD LINES AT Y = W/2 TO
                        ! THE SELECTED SET
LESIZE,ALL,,,15         ! SPECIFY NUMBER OF ELEMENTS
                        ! ALONG LINES
LCCAT,ALL               ! CONCATENATE SELECTED LINES
CSYS,1                  ! SWITCH TO GLOBAL CYLINDRICAL
                        ! COORDINATE SYSTEM
LSEL,S,LOC,X,R          ! SELECT LINES AT r = R
LESIZE,ALL,,,30         ! SPECIFY NUMBER OF ELEMENTS
                        ! ALONG LINES
CSYS,0                  ! SWITCH TO GLOBAL CARTESIAN
                        ! COORDINATE SYSTEM
ALLSEL                  ! SELECT EVERYTHING
```

The deformed shape of the structure and the contour variation of stresses in the *y*-direction after the solution are shown in Fig. 7.6 and 7.7, respectively.

7.3 Selecting Operations

Selecting operations play a key role when programming with APDL. The most commonly used ANSYS commands for selecting operations are given in Table 7.9.

The basic group of selection commands involves the ones that allow the user to select a subset of entities, i.e., KSEL, LSEL, ASEL, VSEL, NSEL, and ESEL. The syntax for these commands is as follows:

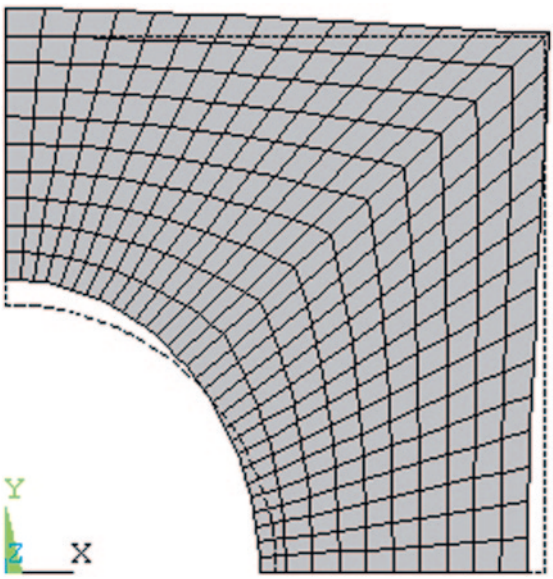

Fig. 7.6 Deformed shape of the structure

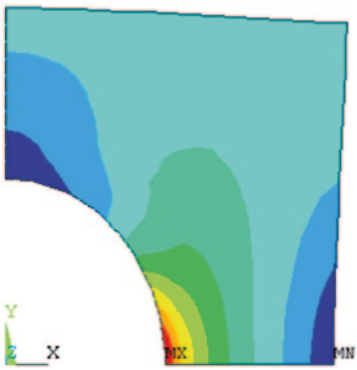

Fig. 7.7 Contour plot of normal stress in the *y*-direction

```
KSEL, Type, Item, Comp, VMIN, VMAX, VINC, KABS
LSEL, Type, Item, Comp, VMIN, VMAX, VINC, KSWP
ASEL, Type, Item, Comp, VMIN, VMAX, VINC, KSWP
VSEL, Type, Item, Comp, VMIN, VMAX, VINC, KSWP
NSEL, Type, Item, Comp, VMIN, VMAX, VINC, KABS
ESEL, Type, Item, Comp, VMIN, VMAX, VINC, KABS
```

Table 7.9 Commonly used ANSYS commands for selecting operations

Command	Description
ALLSEL	Select all the entities
KSEL, LSEL,ASEL, VSEL, NSEL, ESEL	Select subsets of keypoints, lines, areas, volumes, nodes, and elements
NSLE	Select nodes attached to the selected elements
ESLN	Select elements containing the selected nodes
NSL, NSLA,NSLV	Select nodes associated with the selected lines, areas, and volumes
ESL, ESLA,ESLV	Select elements associated with the selected lines, areas, and volumes
KSLL	Select keypoints contained in the selected lines
LSLK	Select lines containing the selected keypoints
LSLA	Select lines contained in the selected areas
ASLL	Select areas containing the selected lines
ASLV	Select areas contained in the selected volumes
VSLA	Select volumes containing the selected areas

The first argument, "**Type**," determines the specific type of selection with the following possible values:

S Select a subset from the full set.
R Select a subset from the current selected set.
A Select a subset from the full set and add it to the current selected set.
U Unselect a subset from the current selected set.
ALL Restore the full set.
NONE Unselect the full set.
INVE Invert the current selected set, which unselects the current selected set and selects the current unselected set.

Figure 7.8 graphically illustrates the concepts behind the argument **Type**. The following examples demonstrate the use of **Type**, along with the remaining arguments.

The argument **Item**, depending on the entity, may have several different meanings. The third through sixth arguments (**Comp**, **VMIN**, **VMAX**, and **VINC**) refer to the argument Item. The most commonly used Item arguments are:

- **Entity name**: **KP** for keypoints, **LINE** for lines, **AREA** for areas, **VOLU** for volumes,**NODE** for nodes, and **ELEM** for elements. In this case, the Comp field (stands for component) is left blank, and **VMIN**, **VMAX**, and **VINC** refer to the minimum and maximum values of the item range and value increment in range (if **VINC** is not specified, its default value is 1), respectively. For example, in order to select keypoints 21 through 30, the following statement is used:

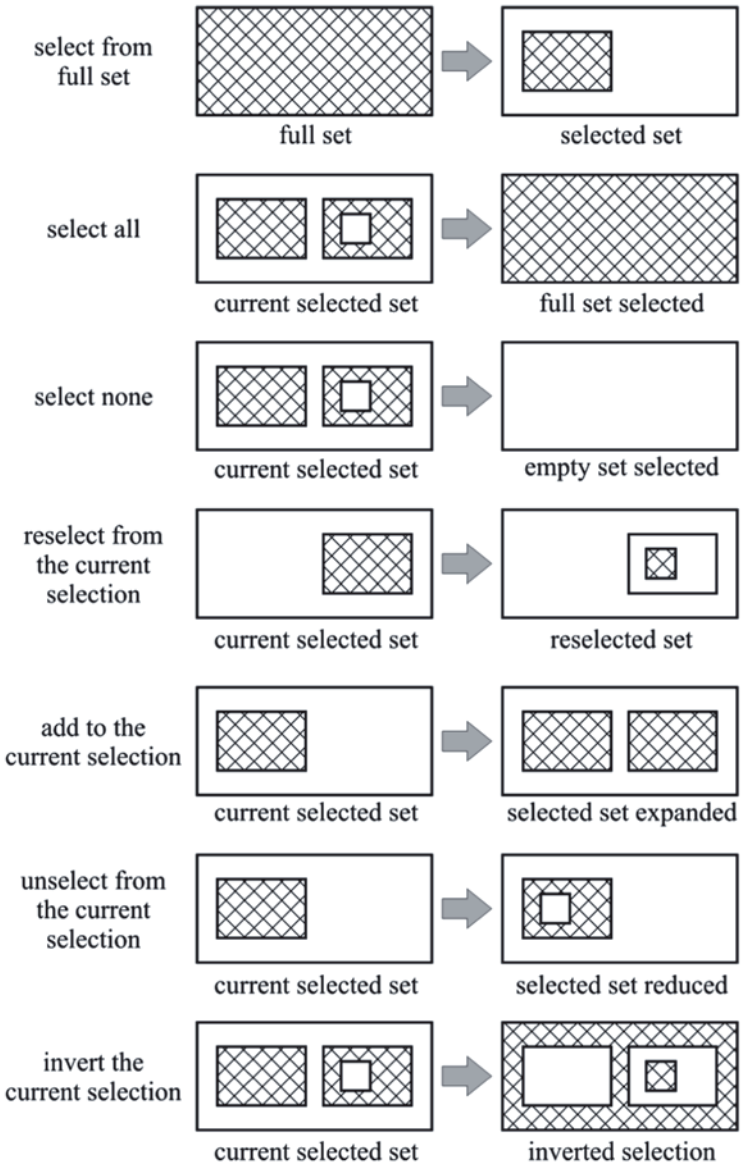

Fig. 7.8 Graphical representation of the argument Type in selection logic

```
KSEL,S,KP,,21,30
```

- **MAT**, **REAL**, **Type**: Selects the entities based on their association with material, real constant, and element type attributes, with the exception of nodes. Similar to the entity name, the Comp field is left blank. The use of this item is demonstrated

in the following example in which the elements with material property attribute number 2 are unselected:

```
ESEL,U,MAT,,2
```

- **LOC**: This item allows the user to perform the selection operations based on the location of the entities, with the exception of elements. The Comp field in this case corresponds to the direction (x, y, z for a Cartesian coordinate system; r, θ, z for a cylindrical coordinate system; etc.). For example, the nodes located within the range $2.5 \le z \le 4$ can be added to the currently selected set of nodes using the following statement:

```
NSEL,A,LOC,Z,2.5,4
```

Several examples demonstrating the concepts used in selection operations are given below.

- Select nodes along planes $x = 1$ and $x = 1.5$ (but ***not*** the ones in between):

```
NSEL,S,LOC,X,1
NSEL,A,LOC,X,1.5
```

or

```
NSEL,S,LOC,X,1,1.5,0.5
```

- Select nodes along planes $x = 1$ and $x = 1.5$ (but ***not*** the ones in between) and within the range $0 \le y \le 4$:

```
NSEL,S,LOC,X,1
NSEL,A,LOC,X,1.5
NSEL,R,LOC,Y,0,4
```

- Select keypoints within the range $10 < x < 15$ (note that $x = 10$ and $x = 15$ are excluded):

```
TINY=1E-6 ! DEFINE SMALL NUMBER
KSEL,S,LOC,X,10+TINY,15-TINY
```

- Select keypoints with $x \le 10$ and $x > 15$:

```
TINY=1E-6
KSEL,S,LOC,X,10+TINY,15
KSEL,INVE ! INVERT SELECTION
```

- Select elements 1 through 101 with the increment 2:

```
ESEL,S,ELEM,,1,101,2
```

- Select elements with material attribute number 5 but without those whose real constant attribute number is 3:

```
ESEL,S,MAT,,5
ESEL,U,REAL,,3
```

The remaining commands included in Table 7.9 perform more specific tasks, mostly utilizing the association between the entities. For example **NSLE**, the command is used for performing selection operations on nodes associated with the currently selected set of elements. The command line input given in the following example unselects the nodes that are attached to the selected set of elements:

```
NSLE,U
```

Command **ESLN** selects elements attached to the currently selected set of nodes. Analogous to the other selection commands, the first argument is **Type**, which determines the type of selection. The value of the second argument, **EKEY**, determines which elements are to be selected:

If **EKEY** = 0, elements are selected if **any** of their nodes are in the selected node set.

If **EKEY** = 1, elements are selected only if **all** of their nodes are in the selected node set.

The following lines demonstrate the use of this command:

```
ESLN,S,0 ! CASE 1
ESLN,S,1 ! CASE 2
```

7.4 Extracting Information from ANSYS

Programming with the ANSYS Parametric Design Language often requires the extraction of data such as entity numbers and locations, geometric information, results, etc. Considering the fact that a typical FEA mesh consists of thousands of

nodes and elements, the user does not usually have control over the entity numbering. Thus, the information that nodes exist at a specific location may be known without knowledge of their numbers. So, if the user is interested in extracting the variation of a certain solution item along a specific path, the aforementioned data extraction tasks must be performed. These tasks are achieved using the ***GET** command. The syntax of the ***GET** command is as follows:

```
*GET,Par,Entity,ENTNUM,Item1,IT1NUM,Item2,IT2NUM
```

The ***GET** command retrieves and subsequently stores data into parameters. The first argument, **Par** , is the parameter name given by the user. The help page for ***GET** the command provides a complete list of possible argument combinations, and it is highly recommended that the user refer to it. In order to explain the use of the ***GET** command, we consider the examples given below.

- Store the maximum and minimum node numbers in the currently selected node set in parameters *maxnod* and *minnod*:

```
*GET,maxnod,NODE,0,NUM,MAX
*GET,minnod,NODE,0,NUM,MIN
```

- Store the maximum and minimum element numbers in the currently selected element set in parameters *maxel* and *minel*:

```
*GET,maxel,ELEM,0,NUM,MAX
*GET,minel,ELEM,0,NUM,MIN
```

- Store the number of nodes and elements in the currently selected node and element sets in parameters *numnod* and *numel*:

```
*GET,numnod,NODE,0,COUNT
*GET,numel,ELEM,0,COUNT
```

- Store the x-, y-, and z-coordinates of the node numbered *maxnod* in parameters x1, y1, and z1:

```
*GET,x1,NODE,maxnod,LOC,X
*GET,y1,NODE,maxnod,LOC,Y
*GET,z1,NODE,maxnod,LOC,Z
```

- Store the x-, y-, and z-displacements of the node numbered *minnod* in parameters u2, v2, and w2:

```
*GET,u2,NODE,minnod,U,X
*GET,v2,NODE,minnod,U,Y
*GET,w2,NODE,minnod,U,Z
```

- Store the rotations of the node numbered *minnod* about the x-, y-, and z-axes in parameters r_x, r_y, and r_z:

```
*GET,r_x,NODE,minnod,ROT,X
*GET,r_y,NODE,minnod,ROT,Y
*GET,r_z,NODE,minnod,ROT,Z
```

- Store the shear stresses σ_{xy}, σ_{yz}, and σ_{xz} and von Mises stress σ_{eqv} at the node numbered *maxnod* in parameters *s_xy, s_yz, s_xz*, and *s_eqv*:

```
*GET,s_xy,NODE,maxnod,S,XY
*GET,s_yz,NODE,maxnod,S,YZ

*GET,s_xz,NODE,maxnod,S,XZ
*GET,s_eqv,NODE,maxnod,S,EQV
```

- Store the normal strains ε_{xx}, ε_{yy}, and ε_{zz} at the node numbered *minnod* in parameters *eps_xx, eps_yy*, and *eps_zz*:

```
*GET,eps_xx,NODE,minnod,EPEL,X
*GET,eps_yy,NODE,minnod,EPEL,Y
*GET,eps_zz,NODE,minnod,EPEL,Z
```

- Store the x-, y-, and z-coordinates of the centroid of the element numbered *maxel* in parameters *ce_x, ce_y*, and *ce_z*:

```
*GET,ce_x,ELEM,maxel,CENT,X
*GET,ce_y,ELEM,maxel,CENT,Y
*GET,ce_z,ELEM,maxel,CENT,Z
```

- Store the area of the element numbered *minel* in parameters *e_area*:

```
*GET,e_area,ELEM,minel,AREA
```

As an alternative to the syntax given above, one can use readily available ***GET** functions that are predefined in compact form. A few of these functions are listed in Table 7.10. For example, the *x*-, *y*-, and *z*-displacements of the node numbered *minnod* can be stored in parameters *u2*, *v2,* and *w2* by using the following:

```
u2=UX(minnod)
v2=UY(minnod)
w2=UZ(minnod)
```

Table 7.10 Selected compact ***GET** functions

Command	Description
NX(**n**), NY(**n**), NZ(**n**)	Retrieve *x*-, *y*-, and *z*-coordinates of node numbered **n**
NDNEXT(**n**)	Retrieve node number of the next selected node having a node number greater than node **n**
ELNEXT(**e**)	Retrieve element number of the next selected element having an element number greater than element **e**
UX(**n**), UY(**n**), UZ(**n**)	Retrieve *x*-, *y*-, and *z*-displacements of node numbered **n**
ROTX(**n**), ROTY(**n**), ROTZ(**n**)	Retrieve rotations about *x*-, *y*-, and *z*-axes of node numbered **n**
TEMP(**n**)	Retrieve temperature at node numbered **n**
PRES(**n**)	Retrieve pressure at node numbered **n**

7.5 Programming with ANSYS

The ANSYS Parametric Design Language contains features that are common to other scientific programming languages. These include looping (**DO** loops) and conditional branching (**IF** statements), as well as writing formatted output to text files (**/OUTPUT** and ***VWRITE** commands). These concepts are discussed in the following subsections.

7.5.1 DO Loops

Do loops are program blocks containing a series of commands executed repeatedly, once for each value of the *loop index*. The APDL commands ***DO** and ***ENDDO** define the beginning and ending of a do loop, respectively. The syntax for the ***DO** command is

```
*DO,Par,IVAL,FVAL,INC
```

in which **Par** is the loop index and **IVAL** and **FVAL** designate the initial and final values of **Par** to be incremented by **INC**. For example, the following input block is used to find the arithmetic average of *x*-displacements along the boundary defined by $x = -2.5$:

```
/POST1                          ! ENTER POSTPROCESSOR
NSEL,S,LOC,X,-2.5               ! SELECT NODES ALONG X = -2.5
*GET,numnod,NODE,0,COUNT        ! RETRIEVE NUMBER OF NODES
*GET,minnod,NODE,0,NUM,MIN      ! RETRIEVE MINIMUM NODE NUMBER
sum=0                           ! INITIALIZE SUM OF
                                ! DISPLACEMENTS
curnod=minnod                   ! INITIALIZE CURRENT NODE
                                ! NUMBER
-------------------------------------------------------------------------------
*DO,ii,1,numnod                 ! BEGIN DO LOOP
*GET,cux,NODE,curnod,U,X        ! RETRIEVE X- DISPLACEMENT OF
                                ! THE CURRENT NODE
sum=sum+cux                     ! UPDATE SUMMATION
*GET,nextnod,NODE,curnod,NXTH    ! STORE NEXT HIGHER NODE
                                 ! NUMBER IN nextnod
curnod=nextnod                  ! UPDATE CURRENT NODE
*ENDDO                          ! END DO LOOP
-------------------------------------------------------------------------------
avgdisp=sum/numnod              ! CALCULATE ARITHMETIC AVERAGE
```

In the example above, **ii** is the loop index with the initial and final values of **1** and **numnod**, respectively. Before the do loop begins, the necessary information is obtained by using the ***GET** command (**numnod** for number of nodes **minnod** and for the minimum node number in the selected set of nodes). Also, two new parameters are defined:

- **sum** for the summation of displacements, which is updated within the do loop and finally divided by the number of nodes (**numnod**) to find the arithmetic average.
- **curnod** designating the node number of the "current node" within the do loop. Its initial value is set as the minimum node number in the selected set of nodes (**minnod**), and it is updated within the loop.

Additional do loops may be used within do loops. For example, the following input block creates 216 nodes, starting from the origin, with increments of 0.25, 0.1, and 0.5 in the *x*-, *y*-, and *z*-directions, respectively.

```
/PREP7      !  ENTER PREPROCESSOR
dx=0.25     !  DEFINE PARAMETER FOR INCREMENT IN X
dy=0.1      !  DEFINE PARAMETER FOR INCREMENT IN Y
dz=0.5      !  DEFINE PARAMETER FOR INCREMENT IN Z
-------------------------------------------------------------------------------
*DO,i,1,6                          ! BEGIN DO LOOP IN i
*DO,j,1,6                          ! BEGIN DO LOOP IN j
*DO,k,1,6                          ! BEGIN DO LOOP IN k
N,,(i-1)*dx,(j-1)*dy,(k-1)*dz      ! CREATE NODE
*ENDDO                             ! END DO LOOP IN k
*ENDDO                             ! END DO LOOP IN j
*ENDDO                             ! END DO LOOP IN i
```

7.5.2 IF Statements

Conditional branching, which is the execution of an input block based on a condition, is accomplished by using the ***IF** command. The syntax for the ***IF** command is

```
*IF,VAL1,Oper1,VAL2,Base1,VAL3,Oper2,VAL4,Base2
```

in which **VAL1**, **VAL2**, **VAL3**, and **VAL4** are numerical values or parameters that are compared (**VAL1** is compared to **VAL2**, and **VAL3** is compared to **VAL4**). The types of these comparisons are dictated by operator arguments **Oper1** and **Oper2**. Finally, the arguments **Base1** and **Base2** specify the action to be taken based on the comparisons. Operator arguments **Oper1** and **Oper2** may take the following selected logical values:

EQ Equal to (**VAL1**=**VAL2**).
NE Not equal (**VAL1**≠**VAL2**).
LT Less than (**VAL1**<**VAL2**).
GT Greater than (**VAL1**>**VAL2**).
LE Less than or equal to (**VAL1**≤**VAL2**).
GE Greater than or equal to (**VAL1**≥**VAL2**).

Common logical values for action arguments **Base1** and **Base2** are:

AND True if both comparisons dictated by **Oper1** and **Oper2** are true.
OR True if either one of the comparisons dictated by **Oper1** and **Oper2** is true.
XOR True if either one but not both of the comparisons dictated by **Oper2** and is true.
THEN If the preceding logical comparison is true, continue to the next line, otherwise skip to one of the following commands (whichever appears first): ***ELSE**, ***ELSEIF**, or ***ENDIF**. This point is explained in further detail below.

In the event that the first action argument **Base1** has the logical value **THEN**, which is often the case, then the conditional branching has the form

```
*IF,VAL1,Oper1,VAL2,THEN
```

and implies that this is an **IF-THEN-ELSE** block and that it ***must*** be ended by a ***ENDIF** command. Between the ***IF** (marking the beginning of the block) and ***ENDIF** (marking the end of the block) commands, the user may use ***ELSEIF** and ***ELSE** commands. A typical **IF-THEN-ELSE** block has the following form:

```
*IF,VAL1,Oper1,VAL2,THEN      ! COMPARISON-1
...                           ! APDL-1 (ANY NUMBER OF APDL
                              ! COMMANDS)
*ELSEIF,VAL1,Oper1,VAL2       ! COMPARISON-2 (OPTIONAL)
...                           ! APDL-2 (ANY NUMBER OF APDL
!                             ! COMMANDS)
*ELSE                         ! COMPARISON-3 (OPTIONAL)
...                           ! APDL-3 ANY NUMBER OF APDL
                              ! COMMANDS
*ENDIF
```

There may be several ***ELSEIF** commands ***ELSEIF**. command usage is the same as for the ***IF** command (as far as the arguments are concerned) whereas the ***ELSE** command does ***not*** have any arguments. There can only be one ***ELSE** command, and it is the last **IF-THEN-ELSE** block command before the ***ENDIF** command. In the example above, note that:

If **COMPARISON-1** is true, then the input block **APDL-1** is executed and the input blocks **APDL-2** and **APDL-3** are ignored. If **COMPARISON-1** is false ***and*** if **COMPARISON-2** is true, then the input block **APDL-2** is executed and the input blocks **APDL-1** and **APDL-3** are ignored. Finally, if neither **COMPARISON-1** nor **COMPARISON-2** is true, then the input block **APDL-3** is executed and the input blocks **APDL-1** and **APDL-2** are ignored.

The arithmetic average of x-displacements along the boundary defined by $x = -2.5$ was evaluated in the example considered in Sect. 7.5.1. In order to demonstrate the use of **IF-THEN-ELSE** blocks, the example is modified by computing the arithmetic averages of positive and negative x-displacements separately, and the number of nodes with zero x-displacement along the boundary specified as $x = -2.5$. This task can be performed by the following input block:

```
/POST1                              ! ENTER POSTPROCESSOR
NSEL,S,LOC,X,-2.5                   ! SELECT NODES ALONG
                                    ! X = -2.5
*GET,numnod,NODE,0,COUNT            ! RETRIEVE NUMBER OF NODES
*GET,minnod,NODE,0,NUM,MIN          ! RETRIEVE MINIMUM NODE
                                    ! NUMBER
sum_p=0     ! INITIALIZE SUM OF POSITIVE DISPLACEMENTS
sum_n=0     ! INITIALIZE SUM OF NEGATIVE DISPLACEMENTS
cnt_p=0     ! INITIALIZE # OF NODES WITH POSITIVE
            ! DISPLACEMENTS
cnt_n=0     ! INITIALIZE # OF NODES WITH NEGATIVE
            ! DISPLACEMENTS
cnt_z=0     ! INITIALIZE # OF NODES WITH ZERO DISPLACEMENTS
curnod=minnod                       ! INITIALIZE CURRENT NODE
                                    ! NUMBER
*DO,ii,1,numnod                     ! BEGIN DO LOOP
*GET,cux,NODE,curnod,U,X            ! RETRIEVE X-DISPLACEMENT
                                    ! OF THE CURRENT NODE
*IF,cux,GT,0,THEN      ! BEGIN IF-THEN-ELSE BLOCK
sum_p=sum_p+cux        ! UPDATE SUM FOR POSITIVE
                       ! DISPLACEMENTS
cnt_p=cnt_p+1          ! UPDATE NUMBER OF NODES
*ELSEIF,cux,LT,0       ! cux IS NEGATIVE
sum_n=sum_n+cux        ! UPDATE SUM FOR NEGATIVE
                       ! DISPLACEMENTS
cnt_n=cnt_n+1          ! UPDATE NUMBER OF NODES
*ELSE                  ! cux IS ZERO
cnt_z=cnt_z+1          ! UPDATE NUMBER OF NODES
*ENDIF                 ! END IF-THEN-ELSE BLOCK
-------------------------------------------------------------------
*GET,nextnod,NODE,curnod,NXTH     ! STORE NEXT HIGHER NODE
                                  ! NUMBER IN nextnod
curnod=nextnod                    ! UPDATE CURRENT NODE
*ENDDO                            ! END DO LOOP
ave_d_p=sum_p/cnt_p               ! CALCULATE AVERAGE
                                  ! POSITIVE
ave_d_n=sum_n/cnt_n               ! CALCULATE AVERAGE
                                  ! NEGATIVE
```

The arithmetic averages of positive and negative *x*-displacements are stored in parameters **ave_d_p** and **ave_d_n**, respectively. Also, the number of nodes with zero *x*-displacement is stored in parameter **cnt_z**.

*7.5.3 /OUTPUT and *VWRITE Commands*

APDL offers the option of writing formatted output to text files through use of **/OUTPUT** and ***VWRITE** commands. The **/OUTPUT** command redirects the text output, normally written in the *Output Window*, to a text (ASCII) file whereas the

***VWRITE** command allows desired parameters to be written with FORTRAN (or C) formatting.

The syntax for the **/OUTPUT** command is

```
/OUTPUT,Fname,Ext, ,Loc
```

in which **Fname** and **Ext** are the file name and extension, respectively, and **Loc** decides whether to start writing from the top of this file or to append to it. If the field for **Loc** is left blank, then the output is written from the top of the file. If the value of **Loc** is specified as **APPEND**, then the output is appended. Once the desired data are written to the text file, the output should be redirected back to the *Output Window* using the same command with ***no*** arguments, i.e.,

```
/OUTPUT
```

The syntax for the ***VWRITE** command is

```
*VWRITE,Par1,Par2,...,Par19
```

in which Par1 through Par19 are the parameters to be written with formatting. As observed, up to 19 parameters can be written at a time. A FORTRAN or C format can be used and ***must*** be supplied in the ***next*** line. The FORTRAN format must be enclosed in parentheses and ***only*** *real* or *alphanumeric* formatting is allowed.

When appended to the input block given in Sect. 7.5.2, the commands in the following input block write the arithmetic averages of positive and negative x-displacements, and the number of nodes with zero x-displacement along the boundary of $x = -2.5$ to three parameters (**ave_d_p**, **ave_d_n**, and **cnt_Z**) in a text file named ***data.out***:

```
/OUTPUT,data,out              ! REDIRECT OUTPUT TO FILE
*VWRITE,ave_d_p,ave_d_n       ! WRITE PARAMETERS ON THE
                              ! SAME LINE



(E15.8,2X,E15.8)              ! FORMAT STATEMENT
/OUTPUT                       ! REDIRECT OUTPUT BACK TO
                              ! OUTPUT WINDOW
/OUTPUT,data,out,,APPEND      ! APPEND TO EXISTING FILE
*VWRITE,cnt_z                 ! WRITE THE PARAMETER
(F8.0)                        ! FORMAT STATEMENT
/OUTPUT                       ! REDIRECT OUTPUT BACK TO
                              ! OUTPUT WINDOW
```

In this particular example, **E15.8** in the format statement allocates 15 spaces for the parameter, 8 of which are used for the numbers after the decimal point. The **2X** enforces 2 blank spaces between the parameters. The parameters **ave_d_p** and **ave_d_n** are written in the following format:

```
±0.12345678E±00 ±0.12345678E±00
```

Similarly, the format statement **F8.0** allocates a total of 8 spaces for the parameter with no space for the numbers after the decimal point, and the parametercnt_z must be an integer.

7.6 Macro Files

A more advanced level of APDL use is the *Macro Files*, which are similar to *subroutines* in the FORTRAN programming language. *Macro Files* are saved in separate text files with the file extension **mac** (e.g. ***macro1.mac***) and written using the APDL. If they are saved in the *Working Directory*, they are automatically recognized by the ANSYS program. Otherwise, the user must declare their location using the **/PSEARCH** command. They are particularly useful for tasks that are repeated many times with different values of model variables such as geometry, material properties, boundary conditions, etc. A simple example on how *Macro Files* are used is given below.

The example under consideration involves the modeling of a spring that has a helix shape. The user needs to generate several models with different geometric properties as part of a design requirement. The coordinates of a point on the helix are given by the following set of parametric equations:

$$\begin{aligned} x &= a\cos(t) \\ y &= a\sin(t) \\ z &= bt \end{aligned} \tag{7.1}$$

in which a is the radius of the helix as it is projected onto the x-y plane, $2\pi b$ is the distance in the z-direction of one full turn, and t is the independent parameter. For this purpose, two *Macro Files* are written, with names ***HELIX1.MAC*** and ***HELIX2.MAC***, as given below:

```
! HELIX1.MAC
! MACRO FOR HELIX GENERATION
! ARG1  : COEFFICIENT A IN EQ. 7.1
! ARG2  : COEFFICIENT B IN EQ. 7.1
! ARG3  : NUMBER OF SEGMENTS TO BE USED FOR QUARTER CIRCLE
! ARG4  : NUMBER OF HELIX STEPS
/PREP7 ! ENTER PREPROCESSOR
PI=4*ATAN(1)          ! DEFINE PI
T=0                   ! INITIAL VALUE OF T
DT=2*PI/(4*ARG3-1)    ! INCREMENT OF T
*DO,I,1,4*ARG3
K,,ARG1*COS(T),ARG1*SIN(T),ARG2*T
T=T+DT
*ENDDO
HELIX2,ARG3           ! CALL MACRO HELIX2
LGEN,ARG4,ALL,,,,,2*PI*ARG2
/EOF                  ! MARK END OF FILE

! HELIX2.MAC
! CALLED BY HELIX1.MAC
! ARG1 : NUMBER OF SEGMENTS TO BE USED FOR QUARTER CIRCLE

KSEL,S,KP,,1,ARG1+1
BSPLIN,ALL
KSEL,S,KP,,ARG1+1,2*ARG1+1
BSPLIN,ALL
KSEL,S,KP,,2*ARG1+1,3*ARG1+1
BSPLIN,ALL
KSEL,S,KP,,3*ARG1+1,4*ARG1+1
BSPLIN,ALL
ALLSEL
LGLUE,ALL

/EOF
```

As long as these files are located in the *Working Directory*, issuing the following command produces the helix shown in Fig. 7.9 (oblique view):

```
HELIX1,1,0.1,4,4
```

Note that the arguments are specified as $a = 1$ and $b = 0.1$, and four segments are used in creating a quarter circle. Finally, the geometry possesses a total of 4 helix steps. When the radius is modified to be $a = 0.5$, and the number of helix steps is increased to 10 using

```
HELIX1,0.5,0.1,4,10
```

the geometry shown in Fig. 7.10 is obtained.

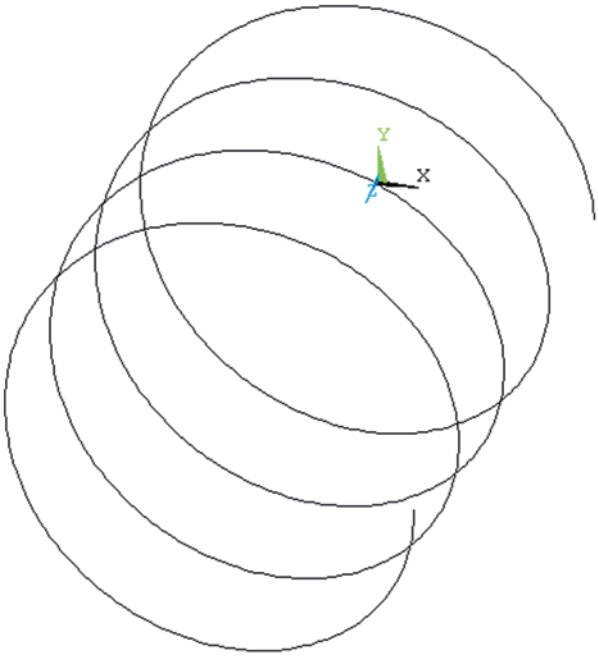

Fig. 7.9 Helix created upon execution of user-defined macro **`HELIX1`** with arguments ***1***, ***0.1***, ***4***, and ***4***

7.7 Useful Resources

There are three main resources for help in enhancing and accelerating knowledge of and skill in programming with APDL:

ANSYS Help System
Log File
ANSYS Verification Manual

The first topic is discussed in sufficient detail in Sect. 2.7. The following subsections briefly discuss the second and third topics.

7.7.1 Using the Log File for Programming

Every time ANSYS is used interactively (using GUI), a *Log File* is created in the *Working Directory* with the name ***jobname.log***. If *Jobname* is not specified, the default for the *Jobname* is ***file*** and the *Log File* is named ***file.log***. This file records every single action the user takes when using ANSYS, including the ones that are not directly related to the finite element method, such as graphics (zoom in/out, turning on/off entity numbering in the *Graphics Window*, etc.). Although it may appear to be a little "messy," it is extremely useful in learning which commands are used for certain actions when using GUI.

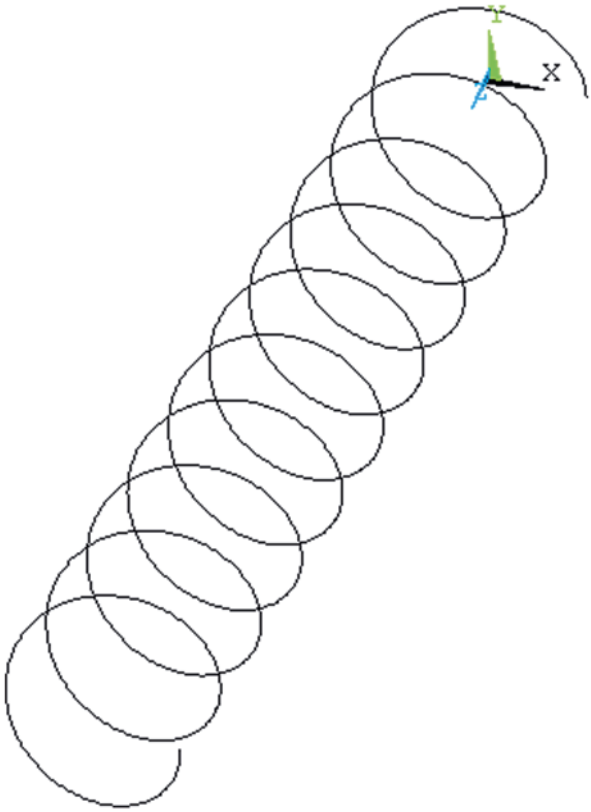

Fig. 7.10 Helix created upon execution of user-defined macro **`HELIX1`** with arguments *0.5*, *0.1*, *4*, and *10*

The following example illustrates how the user can utilize the *Log File* for learning certain commands. Suppose the domain is the same as the one considered in Sect. 7.2 and the user is required to write an input file to create the solid model. The model generation includes:

Creation of a square with side length of 2.
Creation of a circle with radius 1.
Subtraction of the circle from the square.

The details on how to perform these simple operations in GUI are left out as they should be clear based on the coverage in Chap. 4 and several examples throughout this book. Once the user performs these operations using the GUI, the *Log File* can be viewed from within ANSYS using

Utility Menu > List > Files > Log File

which should appear in a separate *Output Window* with contents as given below:

```
/BATCH
/COM,ANSYS RELEASE 8.0 UP20030930 14:49:33 06/16/2004
/PREP7
RECTNG,,2,,2,
PCIRC,1, ,0,360,
ASBA, 1, 2
```

Creation of the square and the circle is achieved by using the commands **RECTNG** and **PCIRC**, respectively. Finally, area subtraction is performed using the **ASBA** command. Once the command names are identified, *Help Pages* for these commands must be read to learn the correct and most efficient usage. The *Log File* may not always be as clear-cut as the one shown above, especially when the GUI action involves graphical picking. However, as the user becomes more accustomed to the use of ANSYS, both in *Batch* and *Interactive Modes*, the *Log File* becomes easier to follow.

7.7.2 Using the Verification Problems for Programming

As mentioned in Sect. 2.7, the *Verification Manual* under the ANSYS *Help System* provides an excellent platform for programming in APDL. In order to demonstrate this point, *Verification Problem 2* (one of the 238 problems) is selected. In this problem, a simply supported I-beam with known properties is subjected to a uniformly distributed transverse loading, as shown in Fig. 7.11. The help page for this problem can be viewed by using the following menu path ***within*** the ANSYS *Help System*, which appears on the left side of the *Help Window* (heading shown in Fig. 7.12):

Mechanical APDL > Verification Test Case Descriptions > VM2

The problem description, a sketch of the problem and corresponding FEA representation, the reference from which the problem is taken, and analysis assumptions are found at the top of the page. Included further down is a table (see Fig. 7.13) showing the results obtained by both analytical methods and the ANSYS software. As observed in Fig. 7.12, there is a hyperlink to the text file **vm2.dat**, which includes the input commands for the solution of this problem using the *Batch Mode*. Upon clicking on this hyperlink, the file appears as partially shown in Fig. 7.14. The user can go through this file line by line, referring frequently to the help pages of unfamiliar commands in order to learn the correct usage of commands.

Another important benefit from the *Verification Manual* is that one can learn how to solve problems with certain properties. The *Verification Test Case Descriptions* help page, accessed through the menu path

Mechanical APDL > Verification Test Case Descriptions

is a good place to start. For example, if the problem at hand involves materials with viscoelastic behavior, it would be a good idea to scan the test case descriptions to

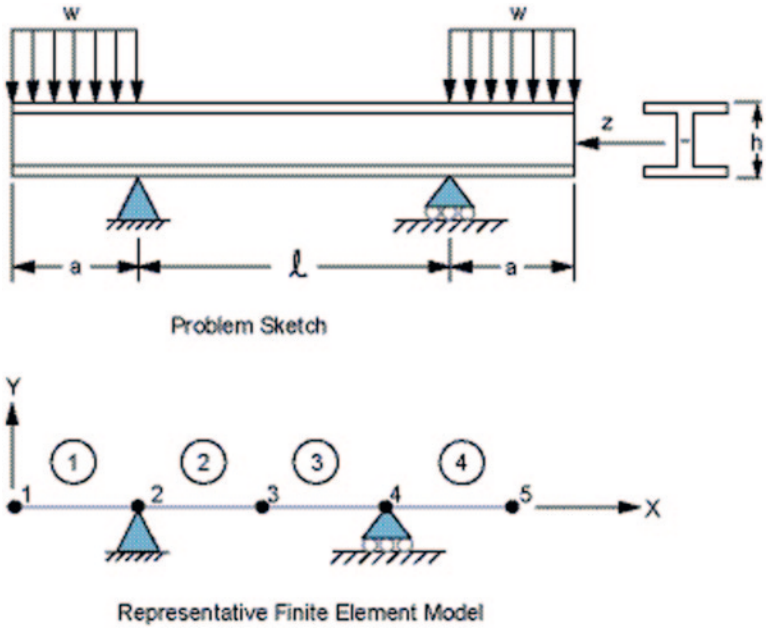

Fig. 7.11 Graphical description of *Verification Problem 2* as given in ANSYS *Verification Manual*

VM2

Beam Stresses and Deflections

Overview

Reference:	S. Timoshenko, *Strength of Material, Part I, Elementary Theory and Problems*, 3rd Edition, D. Van Nostrand Co., Inc., New York, NY, 1955, pg. 98, problem 4.
Analysis Type (s):	Static Analysis (ANTYPE = 0)
Element Type (s):	3-D 2 Node Beam (BEAM188)
Input Listing:	vm2.dat

Fig. 7.12 Heading of the help page for *Verification Problem 2* as given in ANSYS *Verification Manual*

Results Comparison

	Target	ANSYS	Ratio
Stress, psi	-11400.000	-11440.746	1.004
Deflection, in	0.182	0.182	1.003

Fig. 7.13 Comparison of results as given in ANSYS *Verification Manual*

VM2 Input Listing

```
/COM,ANSYS MEDIA REL. 8.0 (9-17-2003) REF. VERIF. MANUAL: REL. 8.0
/VERIFY,VM2
JPGPRF,500,100,1             ! MACRO TO SET PREFS FOR JPEG PLOTS
/SHOW,JPEG
/PREP7
MP,PRXY,,0.3
/TITLE, VM2, BEAM STRESSES AND DEFLECTIONS
C***        STR. OF MATL., TIMOSHENKO, PART 1, 3RD ED., PAGE 98, PROB. 4
ANTYPE,STATIC
ET,1,BEAM3
KEYOPT,1,9,9                  ! OUTPUT AT 9 INTERMEDIATE LOCATIONS
R,1,50.65,7892,30
MP,EX,1,30E6
N,1                           ! DEFINE NODES AND ELEMENTS
N,5,480
FILL
E,1,2
EGEN,4,1,1
D,2,UX,,,,,UY                 ! BOUNDARY CONDITIONS AND LOADING
D,4,UY
SFBEAM,1,1,PRES,(10000/12)
SFBEAM,4,1,PRES,(1E4/12)
FINISH
/SOLU
```

Fig. 7.14 VM2 input listing as given in ANSYS *Verification Manual*

find a solved problem with such materials. A quick glance at the list of test case descriptions reveals that *Verification Problem 200* involves a viscoelastic material and that the user may benefit from examining this file in order to see how the problem is treated before moving on to the problem at hand, which is likely to be more complicated.