PDEase2D Reference Manual

The pre-defined sections and their descriptions are:

Section Type	Description
Title	Text title to identify the problem in output files and plots.
Select	Option settings which control various optional features of PDEase2D.
Coordinates	Independent spatial variables and coordinate systems.
Variables	Dependent field variables.
Definitions	Definition of terms and functions which appear in the equations, constraints, boundary specifications and so forth.
Initial Values	Initial conditions in time-dependent problems.
Equations	The differential equations to be solved.
Constraints	Constraint equations which are imposed on the solution.
Boundaries	Geometric boundaries and regions and boundary conditions.
Time	Time coordinate, lower and upper limit and number of time steps in time-dependent problems.
Monitors	Plots which are displayed during the progress of the computation.
Plots	Plots which are displayed upon completion of the computation.
Histories	Time history of the values of one or more dependent variables at a point during a time-dependent problem.
End	Signifies the end of the problem descriptor file.

Each section must begin with its own pre-defined (reserved) name to inform PDEase2D of the nature of the statements which it contains. While PDEase2D permits considerable flexibility in the order and number of sections used in a problem descriptor file, it processes a problem descriptor file from top to bottom and cannot make forward references. Variables can be used in a Definitions section provided they have been declared in a previous Variables section. Definitions declared in a Definitions section can be used in any subsequent definitions or expressions.

With the exception of the Boundaries and End sections, the use of any particular section in a problem descriptor file is optional. All problem descriptor files must contain at least a Boundaries section and an End section.

You should use each section in an input file only once and in the order listed in the PDease2D.MFE template.

While PDEase2D does not require any formal structure, it is good practice to format the sections and statements using indentations to indicate levels in the hierarchical structure of the file as follows:

```
section_1

statement

statement

section_2

statement

statement
```

This format is easy for both the user and others to read and understand.

Example

```
{SIMPLE.PDE}

{*****************************************************************

This sample demonstrates the simplest application of PDEase2D to heat flow problems.

The heat flow equation is

div(K*grad(Temp)) + Source = 0.

The function

Temp = Const - x**2 - y**2

satisfies the heat equation if K is constant and  Source = 4*K.

We define a square region of material of conductivity K = 1, with a uniform heat source of 4 heat units per unit
     area. We further specify the boundary value

Temp = 1 - x**2 - y**2.

Since we know the analytic solution, we can compare the accuracy of the PDEase2D solution.

************************************************************}
```

```
Title
"Simple Heatflow"

Variables
Temp(range=0,1)                                    {Identify "Temp" as the system
     variable with approximate range o to 1}

Definitions
K = 1                                                {declare and define the
     conductivity}
source = 4                                           {declare and define the source}
Texact = 1-x**2-y**2                                 {for convenience, define the
     exact solution}

initial values
Temp = 1                                             {optional in linear steady state
     problems}

Equations
div(K*grad(Temp)) + source = 0                      {define the heatflow equation}
```

```
boundaries                                  {define the problem domain}

Region 1                                    {... only one region}

value(Temp)=Texact                {specify Dirichlet boundary at exact solution}

start(-1,-1)                      {specify the starting point}
line to (1,-1)                    {walk the boundary}
to (1,1)
to (-1,1)
finish                            {bring boundary back to starting point}

monitors
contour(Temp)                     {show the Temperature during solution}
plots                             {write these hardcopy files at completion}
contour(Temp
surface(Temp)
contour(Temp-Texact) as "Error"
vector(-dx(Temp),-dy(Temp)) as "Heat Flow"

end {end of descriptor file }
```

- You can prepare problems for PDEase2D using PDEase2D's easy-to-learn natural language and a text editor to create a problem descriptor file which describes mathematically the problem to be solved. Or you can edit the text inside a PDEase2D command section inside a Macsyma notebook.

PDEase2D supports use of the following continuous functions:

Function	Description
ABS(arg)	Absolute value.
ARCCOS(arg)	Arc cosine in radians.
ARCSIN(arg)	Aarc sine in radians.
ARCTAN(arg)	Arc tangent in radians.
ATAN2 (arg1,arg2)	Returns the angle in radians whose cosine and sine are proportional to arg1 and arg2. Same as ATAN(arg1/arg2), except when arg2=0.
COS(arg)	
COSH(arg)	
ERF(arg)	Error function.
ERFC(arg)	Error function complement.
EXP(arg)	Exponential function.
LOG (arg)	Natural log; synonym for LN(arg).
LOG10(arg)	Log base 10.
LN(arg)	Natural log.
SIN(arg)	
SINH(arg)	
SQRT(arg)	Square root.
TAN(arg)	
TANH(arg)	

PDEase2D supports use of the following piecewise continuous functions:

- Function Description
- USTEP(arg) The unit step function. USTEP requires one argument and is 0 for negative values of its argument and 1 for positive values of its argument.
- UPULSE(arg1,arg2) The unit pulse function. UPULSE requires two arguments and is 1 when argument1 is positive and argument2 is negative and is 0 everywhere else.
- URAMP(arg1,arg2) The unit ramp function. URAMP requires two arguments and is 0 when argument1 and argument 2 are both negative, ramps linearly from 0 to 1 when argument1 turns positive while argument2 is negative, and is 1 when argument1 and argument2 are both positive. If arg1 is greater than arg2 URAMP evaluates to USTEP(arg1).

The general differential operator DIFF can be use for all scalar differentiation operations. Specify the Select option "Macsyma" to use the DIFF operator.

PDEase2D Notation	Action
DIFF(v,var,n)	nth order (partial) derivative of f(var) with respect to spatial or time coordinate var.
DT(v)	Partial derivative of v with respect to time
DX(v)	First order partial derivative of v with respect to coordinate x
DXX(v)	Second order partial derivative of v with respect to coordinate x
DY(v)	First order partial derivative of v with respect to coordinate y
DYY(v)	Second order partial derivative of v with respect to coordinate y.
DEL2(v)	Two dimensional Laplacian of v.

PDEase2D contains differential operators which make it much easier to write vector calculus operations than by using the scalar differential operators.

- PDEase2D allows the syntax GRAD(V)**2 and treats this as DOT(GRAD(V),GRAD(V)).
- For more information about vector operators, see Discussion: Div, Curl, and Natural Boundary Conditions.

• DIV(F)	Div(F)	Divergence of a vector .
• GRAD(v)	Grad(v)	Gradient of a scalar. The result is a vector.
• CURL(F)	Curl(F)	Curl of a vector . The result is the scalar magnitude of a vector normal to the computational plane.
• CURL(V)	Curl(v)	Curl of the scalar component of a vector perpendicular to the computational plane. The result is a vector.

Vector Algebra Operations

- CROSS(arg1,arg2) The cross product. CROSS requires two vector arguments and returns a scalar value equal to the component of the vector cross product of arg1 and arg2 normal to the computational plane.
- DOT(arg1,arg2) The dot product. DOT requires two vector arguments and returns a scalar value equal to the magnitude of the vector dot product of arg1 and arg2.
- MAGNITUDE(arg) The magnitude function. MAGNITUDE requires one vector argument and returns a value equal to the magnitude of the vector arg.
- NORMAL(arg) The normal-component function. NORMAL requires one vector argument and returns a scalar value equal to the normal component of the vector argument. NORMAL only has meaning with respect to a vector argument on a defined external or internal boundary.
- TANGENTIAL(arg1) The tangential-component function. TANGENTIAL requires one vector argument and returns a scalar value equal to the tangential component of the vector argument. TANGENTIAL only has meaning with respect to a vector argument on a defined external or internal.
- VECTOR(arg1,arg2) The vector function. VECTOR requires two scalar arguments and constructs a vector whose components are arg1 and arg2.

Integral Operators

- INTEGRAL(arg1,["arg2"]) The area integral function. INTEGRAL requires one or two arguments. When only one argument is used, the function value is the area integral of the argument evaluated over the entire spatial domain of the problem. When two arguments are used, the second argument must be a named region defined in the BOUNDARIES section of a problem descriptor file.

- BINTEGRAL(arg1,["arg2"]) The boundary integral function. BINTEGRAL requires one or two arguments. When only one argument is used the function value is the line integral of the argument evaluated over all boundaries of the problem. When two arguments are used, the second argument must be a named path defined in the BOUNDARIES section of a problem descriptor file.

Conditional Expressions

PDEase2D follows the conventional mathematical standards of evaluation arithmetic operators in conditionals. You can force a particular order of evaluation by using parentheses ( ) or [ ].

You can nest conditional expressions. When you use nesting, the else clauses are associated with the if clauses by the rule that the last else clause goes with the first if, etc.

PDEase2D Symbol	Definition
=	equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
<>	not equal to

Example

```
{ IFPARAM.PDE
This problem is a nonlinear test, which solves a modified steady-state Burger's equation. The conductivity is
      defined by a conditional expression. }
Title
"Nonlinear Heatflow, Conditional Conductivity“
Variables
u
Definitions
nl = u
a = if u<0.5 then 1+2*u else 2
Initial values
u = 1 - (x-1)**2 - (y-1)**2
Equations
div(a*grad(u)) + nl*dx(u) +4 = 0
Boundaries
Region 1
value(u)=0
start(0,0) line to (2,0) to (2,2) to (0,2) finish

Monitors
contour(u)
Plots
surface(u) dataviewer
contour(u)
contour(a) as "Conductivity“
elevation(a,u) from (0,1) to (2,1)
End
```

Closed Boundary conditions

You can use the Boundaries section to inform PDEase2D fully of complex physical geometries. The Boundaries section uses a hierarchical structure consisting of one subsection which must begin with the reserved word region, and as many additional subsections as needed to fully describe the physical geometry of the problem, each of which must begin with the reserved word REGION, feature, or exclude.

```
Boundaries

region [integer1]["region1_name"]
start ["path1_name"] (xcoord1,ycoord1)
geometric statement
geometric statement
geometric statement
finish

region [integer2]["region2_name"]
start [path2_name](xcoord2,ycoord2)
geometric statement
geometric statement
finish
```

```
feature [interger3]
start [path3_name](xcoord3,ycoord3)
geometric statement
geometric statement
Finish

exclude [interger5]
start [path5_name](xcoord5,ycoord5)
geometric statement
geometric statement
```

Feature subsections (when used) must follow all Region subsections, and Exclude subsections (when used) must follow all Region subsections and any Feature subsections.

Open Boundaries

Feature subsections are used to describe non-closed geometric entities. They are formed in exactly the same manner as Region subsections except that they do not have the terminating reserve word finish, and they do not enclose a subdomain with definable parameters. While not strictly enforced by PDEase2D, it is recommended that all regions be described prior to any features.

Example

```
{describe a square with an internal line feature}
Boundaries
Region
start (0,0)
line to (3,0) to (3,3) to (0,3) to finish
Feature
start (0.5,1.5)
line to (2.5,1.5)
```

Two dimensional regions

- Region subsections are used to describe closed geometric entities which are included in the problem. They are formed by beginning with the reserved word start followed by a starting coordinate point, "walking" a continuous boundary using line and arc segments, and terminating with the reserved word finish.
- The line and arc segments used in forming Region subsections are:
- line to (X,Y)
- arc to (X,Y) to (X1,Y1)
- arc (radius=R) to (X,Y)
- arc (center=X,Y) to (X1,Y1)
- arc (center=X,Y) angle=
- where, is an angle measured in degrees and follows the convention that positive angles rotate counter-clockwise while negative angles rotate clockwise, and the end point is equivalent to the coordinate specification determined by the radius swept out by the angle.
- When successive segments of the same kind (either line or arc) are used, the segment name does not have to be repeated.

Example

```
{describe a simple square}
Boundaries
Region
start (0,0)
line to (3,0) to (3,3) to (0,3) to finish

{describe a hollow square}
Boundaries
Region
start (0,0)
line to (3,0) to (3,3) to (0,3) to finish
start (1.5,0.5)
line to (2.5,1.5) to (1.5,2.5) to (0.5,1.5) to finish
```

Specifying Segment Boundary Conditions

- Segment boundary conditions are specified by statements of the form:
- value(variable) = expression
- natural(variable) = expression
- load(variable) = expression

- In each case, the identifier (variable) assigns the designated boundary condition to the partial differential equation associated with variable as described in EQUATIONS Sections.

- Natural and load are synonymous. They represent generalized derivative boundary conditions, including Neumann boundary conditions for the Poisson equation and surface loading in stress problems.

- Boundary condition specifications must immediately precede either the reserved word start, line or arc and cannot precede the reserved word to. If a boundary condition changes within a series of similar boundary segments the boundary segment type (line or arc) must be repeated.

Example

```
{this problem represents a semiconductor diode with a thin active region on top of a passive region which is then bonded to
        a metal heatsink. The entire structure is convectively coupled to the surrounding ambient}

Boundaries
region 1                                                          {Defines the maximum extent of the system}
start(1,0)
natural(Temp) = B2*(Ta - Temp)
line to (1,.375)
natural(Temp) = B3*(Ta - Temp)
line to (1,.5) to (-1,.5) to (-1,.375)
natural(Temp) = B2*(Ta - Temp)
line to (-1,0)
natural(Temp) = B1*(Ta - Temp)
line to (-2.5,0) to (-2.5,-.5) to (-2.0,-.5)
line to (-2.0,-1.25) to (-1.75,-1.25) to (-1.75,-.5)
line to (-.75,-.5) to (-.75,-1.5) to (-.5,-1.5) to (-.5,-.5)
line to (.5,-.5) to (.5,-1.5) to (.75,-1.5) to (.75,-.5)
line to (1.75,-.5) to (1.75,-1.25) to (2,-1.25) to (2,-.5)
line to (2.5,-.5) to (2.5,0) to finish
{passive region of the diode}
region 2 "passive"
K = .1
{does not need boundary conditions since region 2 is entirely within region 1}
start(1,0)
line to (1,.375) to (-1,.375) to (-1,0) to finish
{active region of the diode}
region 3 "active"
K = .1
A = 1                                                                                  {turns source on in this region}
start(1,.375)
line to (1,.5) to (-1,.5) to (-1,.375) to finish
```

Specifying Boundary Point Values

You can fix boundary values at specific points on a boundary, by following a coordinate specification with the statement:

```
point value(var) = expression
```

Example

```
boundaries                                      {define the problem domain}
Region 1                                        {... only one region }
start(-1,-1)
natural(Temp)=0
line to (0,-1) point value(Temp) = 0
to (1,-1) to (1,1) point value(Temp) = stage
to (-1,1) to (-1,0) point value(Temp) = 0
line to finish
```

Describing Steady-State Problems

- TITLE Sections [optional]
- MFESWITCHES Sections [optional]
- SELECT Sections
- COORDINATES Sections [optional]
- VARIABLES Sections
- DEFINITIONS Sections
- INITIAL VALUES Sections [optional]
- EQUATIONS Sections
- CONSTRAINTS Sections [optional]
- BOUNDARIES Sections
- MONITORS Sections [optional]
- PLOTS Sections [optional]
- END Sections
- File and DataViewer Output Summary

An optional MFE_SWITCHES section may be included at the start of any PDEase script section in the format shown below:

Example

{# MFE_Switches

Keep_Notifications   {Keep notifications section on reexecution}

Root_Name(foo)   {Root name of PDEase variables for this script (default PDZ)}

Large_Engine   {Uses the large node limit engine. Default is to use 64000 node limit model}

PDEase_File_Path(X:₩foo₩bar #}   {- path for ancillary files (e.g., .tbl files)

The default location is 1) the directory where Pdease2D opened the .pde or .mfe file; or 2) the ₩Macsyma2₩PDEase2D directory}

The MFE_SWITCHES section must be the first section of any PDEase2D script.

The switches are optional, case insensitive, and may be listed any order. Note the switches are actually enclosed inside a PDEase2D comment section {?. You must use{# ?#} to distinguish an MFE_SWITCHES section from a comment.

COORDINATES Sections [optional]

Example

```
title "Axi-symmetric Heatflow "

Coordinates
ycylinder("R","Z")                    {select a cylindrical coordinate system, with the rotational axis along
      the "Y" direction and the coordinates named "R" and "Z" }

Variables
Temp(range=50)                        {Define Temp as the system variable, with approximate value range 0
      to 50 }
```

When either xcylinder("name_x","name_y") or ycylinder("name_x","name_y") is specified, PDEase2D automatically expands any DIV, GRAD, and CURL operators used in equations and forms the Galerkin integrals with the volume element radius*d(radius)*d(axis). When the normally cylindrical coordinates r and z are used the volume element is rdrdz).

You should use a COORDINATES section for axisymmetric problems and write equations using the operators DIV, GRAD and CURL if possible.

PDEase2D defaults to CARTESIAN("x","y") in the absence of a Coordinates section.

VARIABLES Sections

- PDEase2D uses the VARIABLES section to declare all the dependent variables for the problem. Also, you can assign names to dependent variables.

- You must begin names with an alphabetic character; you cannot use separators or reserved symbols. Variable names can be of any length. Dependent variable names may be a single character other than those reserved for independent variables, a word other than a reserved word, or a compound word like heat_flux. PDEase2D always treats hyphens (-) as a minus sign. You cannot use them to form compound names.

Variables statements can include an optional range clause. When a range clause is used it takes one of the following forms and applies only to the variable with which it is associated:

- variable_name(range=min,max)
- variable_name(range=max)
- variable_name(min,max)
- variable_name(max)

where:

variable_name	Name assigned to the dependent variable.
range	Reserve word signifying that range values are to follow.
=	Reserved symbol.
min	Minimum expected value of the variable.
max	Maximum expected value of the variable.

DEFINITIONS Sections

PDEase2D uses the Definitions section to declare numerical constants, functions, and expressions, to assign names to them for later use in the problem descriptor file, and optionally to assign global (default) values to them. In general the individual statements declared in the Definitions section relate to some physical entity such as material parameters.

Example

```
Variables
U
V
Definitions
conductivity = 1.03e5 + 1.0/sin(34*x)
resistivity = 1/conductivity
scale_length = 2
rho = 1.25 + sqrt(u^2 + v^2)
```

INITIAL VALUES Sections [optional]

- Pdeases2D uses the Initial Values section, which is optional for steady state problems, to initialize the dependent variables.
- Dependent variables can be initialized to constants, expressions, functions and previously defined definitions. By default any dependent variable not initialized with an initial value statement is initialized to zero. Example

Example

```
Variables
Temp

Definitions
Temp0 = ((x-1)**2 + (y-1)**2)

initial values
Temp = Temp0
```

EQUATIONS Sections

- The Equations section is used to list the partial differential equations to be solved. Equations are written in the same way they would be written on paper with the minor concession that the partial differential operators are specified with the reserved symbols DX and DY for the first order partial derivative with respect to x and y, DXX and DYY for the second order partial with respect to x and y.
- While not required by PDEase2D, steady state equations should always be written with all variable dependent terms to the left of the equal sign and all non-variable dependent terms to the right of the equal sign. Placing certain differential terms of an equation to the right of the equal sign can adversely affect the sign of natural (load) boundary conditions.

```
{thermal elastic expansions of a free body}

variables

Tp                  {temp}
up                  {x - displacement}
vp                  (y - displacement}

Equations
{equation defining temperature dependence}
dx(kp33*dx(Tp)+kp13*dy(Tp)) +dy(kp13*dx(Tp)
+kp11*dy(Tp)) + Qs= 0.

{equation defining x directed displacement}
dx(C31*dy(vp) + C33*dx(up) +2*C35*0.5*(dy(up)+dx(vp)) - apxx*Tp)
 +dy(C51*dy(vp) + C53*dx(up) +2*C55*0.5*(dy(up)+dx(vp)) - apxy*쎄)= 0.

{equation defining y directed displacement}
dx(C51*dy(vp) + C53*dx(up) + 2*C55*0.5*(dy(up)+dx(vp)) - apxy*Tp)
 +dy(C11*dy(vp) + C13*dx(up) +2*C15*0.5*(dy(up)+dx(vp)) - apyy*쎄)= 0.
```

CONSTRAINTS Sections [optional]

{A closed reactor system in which no material is added or removed. Monovalent species 'a' reacts with monovalent species 'b' to produce 'c' releasing heat while 'c' decomposes into 'a' and 'b' absorbing heat}

variables

Temp

a

b

c

Definitions

ktemp = 0.1

ka = 0.05

kb = 0.05

kc = 0.01

heat = 0

eabs = 1.0

a0 = 0.1*(1-r*r) {remember in cartesian coordinates PDEase2D defines r = sqrt(x^2 + y^2) }

b0 = 0.1*(1-r*r)

c = c0

Equations

div(ktemp*grad(temp) - eabs*c*temp + eabs*a*b + heat = 0

div(ka*grad(a)) - a*b + c*temp = 0

div(kb*grad(b)) - a*b + c*temp = 0

div(kc*grad(c)) + a*b - c*temp = 0

constraints {total mass is conserved}

integral(a + b + c) = integral(a0 + b0+ c0)

BOUNDARIES Sections

```
boundaries

region 1 "outer"

start "outer name" (-1,-1)

line to (1,-1) to (1,1) to (-1,1) to finish

region 2 "lower"

start(-1,-0.1)

line to (1,-0.1) to (1,1) to (-1,1) to finish

feature 3

start "feature name" (0,-0.4) to (0,0.4)

exclude 4

start(-0.5,0) arc(center=-0.2,0) angle=360 finish
```

MONITORS Sections [optional]

- contour(arg) Two dimensional display requiring one argument which displays an equal contour map.
- surface(arg) Three dimensional perspective display requiring one argument which displays its argument in elevation.
- elevation(arg1,[arg2,..]) Two dimensional display (sometimes called a line out) requiring at least one argument which displays the value of its argument(s) vertically. Elevation monitors must be followed by either a from (X,Y) to (X1,Y1) line specification or an on "boundary_name" specification. Also draws a thumbnail sketch.
- vector(arg1,arg2) Two dimensional display requiring two arguments which displays the sum of vector components as a directed arrow.
- grid(arg1,arg2) Two dimensional display requiring two arguments. Grid displays are particularly useful for displaying displacements in stress/strain problems.

PLOTS Sections [optional]

```
{ TWOMATL.PDE }
{ *****************************************************************

This sample demonstrates the application of PDEase2D to heatflow problems with differing material parameters.
      We define a square region of material with a conductivity of 5. Imbedded in this square is a diamond-shaped
      region of material with a uniform heat source of 1, and a conductivity of 1.

*****************************************************************}
title "Hot Diamond“
Variables
Temp(range=0,1)            {declare "Temp" to be the system variable, with approximate value range 0 to 1}
Select
paint = off
Definitions
K              {declare the conductivity as a parameter, but leave its value definition until later}
Heat                                              {similarly for the Heat source}
Flux = -K*grad(Temp)
initial values
Temp = 0                                          {unimportant in linear steady-state problems}
Equations
div(K*grad(Temp)) + Heat = 0            {define the heatflow equation}
boundaries                                        {define the problem domain
Region 1                                          {Define the outer boundary}
K=5                                     {Define the conductivity in the outer region}
Heat=0                                  {Define the Heat source in the outer region}
value(Temp)=0              {Prescribe the boundary temperature}
start "outer" (0,0)        {Start the outer boundary at the lower left}
```

PLOTS Sections [optional]

```
line to (3,0)              {Walk the boundary counterclockwise}
to (3,3)
to (0,3)
finish                                 {Return to start}
Region 2                               {Define the imbedded diamond}
K=1                                    {Define the Conductivity}
Heat=1                                 {Define the Source}
start "inner" (1.5,0.5)    {Start at the bottom vertex}
line to (2.5,1.5)                      {Walk the boundary counterclockwise}
to (1.5,2.5)
to (0.5,1.5)
finish                                 {Return to start}

Monitors
contour(Temp)                          {show the Temperature during solution}
plots                                  {write these hardcopy files at completion}
grid(x,y)                              {show the final grid}
contour(Temp) as "Temperature"         {show the solution }
contour(Temp) zoom(2,1,1,1)            {Zoom in on diamond corner}
as "Temperature Zoom“
elevation(Temp)                        {show a diagonal line-out of temperature}
from (0,0) to (3,3) as "Temperature Zoom" print
surface(Temp)  interactive
vector(-dx(Temp),-dy(Temp)) as "Heat Flow“
elevation(normal(flux)) on "Outer" range(0,1)
elevation(normal(flux)) on "Inner" range(0,1)
end
```

Describing Time-Dependent Problems

- Time Range
- Variables Section Range Clause [optional]
- Initial Values
- Equations
- Time Dependent Monitors and Plots
- HISTORIES Section [optional]
- File and DataViewer Output

TIME Section Time Range Specification [optional]

Direct specification of the problem time domain is most easily made by including in the input file a TIME section. When used, the TIME section must contain a single statement of the form:

[from] [=] time1 to [=] time2 [by [=] increment]

where:

from	Reserved word which is optional.
=	Equal sign which is optional.
time1	Start of the time domain.
time2	End of the time domain.
by	Reserved word which may be replaced by reserved word deltat
increment	Optional and when not used defaults to a value of 10e-4*(time1-time2), is the value of the initial time step. PDEase2D will adjust subsequent time steps up or down as required by the problem.

Example

```
Title
'Masked Diffusion'
Select
errlim = 0.001
Variables
u(range=0,1)
Definitions
concs = 1.8e8                    {surface concentration}
D = 1.1e-2                                          {diffusivity}
conc = concs*u
cexact = concs*erfc(x/(2*sqrt(D*t)))
uexact = erfc(x/(2*sqrt(D*t)))
{masked surface flux multiplier}
M = 10*upulse(y-0.3,y-0.7)
initial values
u = 0
Equations
div(D*grad(u)) = dt(u)
Boundaries
region 1
start(0,0)
natural(u) = 0      line to (1,0)
natural(u) = 0      line to (1,1)
natural(u) = 0      line to (0,1)
natural(u) = M*(1-u)  line to finish
feature                                                                       {a "gridding feature" to help localize the activity}
start (0.02,0.3) line to (0.02,0.7)
Time
0 to 1 by 0.001
```

Equations

- For PDEase2D to treat a problem as a time dependent problem, the equation(s) must contain time derivative term(s) of the form:
- dt(variable)
- Example

```
{LASERXTT.PDE}

Initial values

temp = 0

equations

div(k*grad(temp)) + source) = cp*dt(temp)
```

- PDEase2D recognizes only first order partial derivatives with respect to the temporal independent variable. You must rewrite systems which involve higher order partial derivatives by using intermediate dependent variables to reduce the system to first order in time.

For.By.To Time Dependent Monitors and Plots Specification

- You can specify monitors and plots using of any of the following statements:
- for t [=] ts [by t1 to] t1 [by t1 to] ..... ti

where:

for	Reserved word.
t	Reserved symbol.
=	Optional relational operator.
ts	Number or the reserved word starttime.
by	Reserved word.
t1	Time increment.
t1	Number.
ti	Number or the reserved word endtime.

Example

```
time

from 0 to 1 by 0.1

plots                                                           {write these hardcopy files at specified times}

for t=0 by 0.01 to 0.1 by 0.1 to endtime

grid(x,y)                                                       {show the final grid }

contour(Temp) as "Temperature"  range=(0,0.2) {show the solution }

contour(Temp) zoom(2,1,1,1)      range=(0,0.2)          {Zoom in on diamond

corner } as "Temperature Zoom"

elevation(Temp)                                         {show a diagonal line-out of temperature}

from (0,0) to (3,3) range=(0,0.2) as "Temperature Zoom" print

surface(Temp) range=(0,0.2)

vector(-dx(Temp),-dy(Temp)) range=(0,0.5) as "Heat Flow"

elevation(normal(flux)) on "Outer" range=(0,0.5)

elevation(normal(flux)) on "Inner" range=(0,0.5)
```

Multiple Time Dependent Monitors and Plots Specification

- You can use multiple time sequence statements for MONITORS and PLOTS. Each time sequence statement stays in effect until you declare a new one.
- Example

```
{OPENTUBE.PDE}

plots

for cycle=10                        {watch the asked events by cycle}

grid(x,y)

elevation(Temp) from (0,0) to (xev,yev)

elevation(C) from (0,0) to (xev,yev)

for t= 0 by 0.1 to 1       {also print at some fixed times}

contour(Temp)

contour(C)
```

HISTORIES Section [optional]

- You can include a HISTORIES section in time-dependent problems. You must begin this section immediately after the PLOTS section and immediately before the END section.
- You can use the HISTORIES section to display results as a function of time. Optionally, you can place this information in a file or DataViewer.
- PDEase2D supports one type of history statement:
- history(arg) at (X,Y) [(X1,Y1? {file | fileonly } {dataviewer | datavieweronly | dv | dvonly}

Example

```
{Modified OPENTUBE.PDE}

plots

for cycle=10                                {watch the fast events by cycle}

grid(x,y)

elevation(Temp) from (0,0) to (xev,yev)

elevation(C) from (0,0) to (xev,yev)

for t= 0 by 0.1 to 1          {also print at some fixed times}

contour(Temp)

contour(C)

elevation(Temp) from (0,0) to (xev,yev)

elevation(C) from (0,0) to (xev,yev)

histories

history(Temp) at (0,0) (xev/2,yev/2) (xev,yev) as "Temperature"

history(C) at (0,0)  (xev/2,yev/2) (xev,yev) as "Reaction Completion"

end
```

Staging

- The counter established by the SELECT statement stages=integer can be used in subsequent expressions in PDEase2D input files by use of the reserved word stage. Each time PDEase2D runs the problem (stages the problem) the reserved word stage is incremented by one starting with an initial value of one.

```
{ Modified VISCOUS.PDE }

  { ************************************************************
Steady Viscous Flow in a Channel with Obstruction
      This example shows the application of PDEase2D in viscous flow. The Navier-
     Stokes equations for steady incompressible flow in two dimensions in Cartesian
     coordinates are

dens*(dt(U) + U*dx(U) + V*dy(U)) = visc*del2(U) - dx(P) + dens*Fx
dens*(dt(V) + U*dx(V) + V*dy(V)) = visc*del2(V) - dy(P) + dens*Fy

together with the continuity equation
div[U,V] = 0

Where
U and V are the X- and Y- components of the flow velocity
P is the fluid pressure
dens is the fluid density
visc is the fluid viscosity
Fx and Fy are the X- and Y- components of the body force.
}
```

In order to derive a third equation for the pressure variable, we differentiate the U-equation with respect to X and the V-equation with respect to Y and add the equations. Using the continuity equation to eliminate terms, we get

del2(P) = 2*dens*(dx(U)*dy(V) - dy(U)*dx(V))

Although this equation is consistent with the continuity equation, it does not enforce it. However, since div[U,V] = 0, we are free to add it at will to the pressure equation. A negative value of div[U,V] implies the destruction of material, so we need a positive pressure to oppose the flow. This implies a modified pressure equation

del2(P) = 2*dens*[dx(U)*dy(V) - dy(U)*dx(V)] + L*(dx(U)+dy(V))

where L is a "large" number chosen to enforce "sufficient" compliance with the material conservation equation. Setting U and V equal to zero in the U and V equations to reflect the conditions on a no-slip boundary, we get

dx(P) = visc*del2(U)

dy(P) = visc*del2(V)

These relations can be used to specify the natural boundary condition for the pressure equation. The normal component of the gradient of P is

n<dot>grad(P) = nx*dx(P) + ny*dy(P)

where nx and ny are the direction cosines of the surface normal. The problem posed here shows flow in a 2D channel blocked by a bar of square cross-section. We run the problem in three stages, with successively larger values of L. The user can identify from this sequence a value of L which is satisfactory for his application.

We have included three elevation plots of X-velocity, at the inlet, center and outlet of the channel. The integrals presented on these plots show the consistency of mass transport across the channel.

****

```
Title
'Viscous flow in 2D channel, Re < 0.1'

SELECT    STAGES=3
Variables
u(0.1)
v(0.01)
p(1)

Definitions
Lx = 5        Ly = 1.5
Gx = 0        Gy = 0
p0 = 1
speed2 = u^2+v^2
speed = sqrt(speed2)
dens = 1
visc = 1
vxx = (p0/(2*visc*Lx))*(Ly-y)^2       { open-channel x-velocity }
rball=0.25
WEIGHT = 10^STAGE
```

initial values

```
u = 0.5*vxx  v = 0  p = p0*x/Lx
```

Equations

```
div(grad(u)) - dx(p)/visc = dens*(u*dx(u) + v*dy(u))/visc
div(grad(v)) - dy(p)/visc = dens*(u*dx(v) + v*dy(v))/visc
div(grad(p)) = 2*dens*[dx(U)*dy(V) - dy(U)*dx(V)] + WEIGHT*(dx(u)+dy(v))
```

Boundaries

```
region 1
start(0,0)
load(u) = 0  value(v) = 0 load(p) = 0
line to (Lx/2-rball,0)
value(u)=0 value(v)=0 load(p)= - visc*del2(u)
line to (Lx/2-rball,rball)
load(p)=visc*del2(v)
line to (Lx/2+rball,rball)
load(p)=visc*del2(u)
line to (Lx/2+rball,0)
load(u) = 0  value(v) = 0 load(p) = 0
line to (Lx,0)
load(u) = 0      value(p) = p0
line to (Lx,Ly)
value(u) = 0     load(p) = -visc*del2(v)
line to (0,Ly)
load(u) = 0      value(p) = 0
line to finish
```

```
monitors

contour(speed)

plots

grid(x,y)

contour(u)

contour(v)

contour(speed)

vector(u,v) as "flow"

contour(p) as "Pressure"

contour(dx(u)+dy(v)) as "Continuity Error"

elevation(u) from (0,0) to (0,Ly)

elevation(u) from (Lx/2,0) to (Lx/2,Ly)

elevation(u) from (Lx,0) to (Lx,Ly)

end
```

Using Staging to Initialize Non-Linear Problems

- Staged problems always use the solution and solution grid obtained during stage 1 to initialize stage 2, and that obtained during stage 2 to initialize stage, 3 etc. Because of this feature, staging can effectively be used to run a linearized version of a non-linear problem to obtain a good initial value before running the non-linear problem. Staging lists can also be used effectively to gradually approach a problem which might otherwise be unstable.

- Example

{Modified MAGSTAGE.PDE}

{This example demonstrates the use of parameter staging to stabilize a nonlinear problem. A linear solution is used as the initial condition for a nonlinear solution. The problem is that of MAGNET.PDE, and considers the determination of the magnetic vector potential A in a cyclotron magnet.}

Animated Time-Dependent Problems

- You can add the clause animate to any plot type in monitors or plot section in time dependent problems. This will produce a sequence of frames when the calculation ends. For example, in the sample problem opentub1.mfe, the plot section has the commands

```
for cycle=10                    { watch the fast events by cycle }
grid(x,y) animate
contour(Temp) animate as "Temperature - cycles"
contour(C) animate as "Reaction Completion - cycles"
for t= 0 by 0.1 to 1            { also plot at some fixed times }
contour(Temp) animate as "Temperature“
contour(C) animate as "Reaction Completion“
for t= 0.20 by 0.01 to 0.3        { show some surfaces during burn }
surface(Temp) animate as "Temperature“
surface(C) animate as "Reaction Completion"
```

File Output Formats and DataViewer Formats

- PDEase2D can produce the following kinds of file output
- Output from Eigenvalues (Pdz.eig)
- Output from Elevation plots
- Output from Contour plots
- Output from Surface plots
- Output from Histories
- By default, the root_name of this PDEase2D program is Pdz. You can change this using the MFE_SWITCHES.

```
Title "File output test"

coordinates
cartesian(x,y)

definitions
b = 1.0
variables
u
Initial values
u = 1-(x-0.5)**2-(y-0.5)**2
equations
del2(u)+  lambda*u = 0
boundaries
region 1
        start(0,0)
        value(u)=0 line to (1,0)
                                   to (1,0.25)
                                   to (0.5,0.47)
                                   to (0.25,0.47)
                                   to (0.25,0.50)
                                   to (0.5,0.50)
                                   to (1,0.75)
                                   to (1,1)
                                   to (0,1)
        finish
plots

surface(u) file
contour(u) file
elevation  (u) from (0,.5) to (.5,0) file

end
```