PRADIS

OPERATION WITH DAT-FILE

THE SOFTWARE FOR SIMULATION OF NON-STATIONARY PROCESSES IN MECHANICAL SYSTEMS AND SYSTEMS OF OTHER PHYSICAL NATURE

VERSION 4.2

Contents

The DAT-file description

DAT the file forms in the course of operation решателя PRADIS and contains the information from interior array A решателя PRADIS. The Filename consists of a filename of the representation a plus ".DAT extension. Except DAT a file, for correct operation of the Postprocessor, in the course of operation решателя there are two more additional files, with".DIS extensions and '.PNM ", containing the additional information on output variables an OVP. For example, if we start on execution calculation of the representation with filename SWING a command" **SLANG SWING** "as a result of calculation three files with names will be created:

SWING.DAT
SWING.DIS
SWING.PNM

The condition of output DAT of a file is set by parameter **OUTVAR** which is set in operator RUN in an argument list of the program of integration. In the current solver version parameter OUTVAR can accept following values:

OUTVAR=0 - Not to create DAT a file;
OUTVAR=1 - To deduce only required parameters (a condition by default)
OUTVAR=3 - To deduce required parameters a plus a global Jacobian;
OUTVAR=4 – To produce only the information from an OVP. In such condition DAT the file is gained much less on a size, but in it there is no possibility to work with a three-dimensional drawing.

If parameter **OUTVAR** is absent in the list value **OUTVAR=1** by default starts**.**

Data recording phase in DAT a file is set by parameter OUT setting a time slice between filings.

For example, if in the representation the following is noted:

$ $ RUN :

Calculation of a spring pendulum ' NEWMARK (END=3, SCALE=1,
OUTVAR=3, OUT=0.1;
Transition т.A on a X-axis = (-1.7,0),
Velocity т.A on a X-axis = (-2.6, 2.6),
Expedition т.A on a X-axis = (-12.7, 12.7))

That in DAT a file required parameters and a global Jacobian with phase in 0.1 seconds will be noted.

Structure DAT-file

The file consists of the block of fixed parameters and the block of variable parameters for each pitch of operation решателя. The leading-out happens in a binary aspect. Variables share on constants (on an integration time) and variables (on an integration time). As a variable be not mandatory can in a file it is necessary to know what variables is and in what order they go. For this purpose each variable assigns the code which defines a variable type.

File structure the following:

The start block:

- The format version, integer4
- Creation date, integer4
- The representation description, char (1024)
- Size of the table of codes of fixed parameters, integer4
- Size of the table of codes of variable parameters, integer4

The table of codes of fixed parameters:

1. A code 1
2. …
3. Code N

The table of codes of variable parameters:

4. A code 1
5. …
6. Code N

The block of fixed parameters:

7. Value 1
8. …
9. Value N

The block of variable parameters a stratum 1:

10. Value 1
11. …
12. Value N

…

The block of variable parameters stratum T:

13. Value 1
14. …
15. Value N

In the beginning of value of each fixed and variable parameter its length format INTEGER*4, and then an array of appropriate length of variables in format REAL*8 or INTEGER*4 is noted.

The file contains following arrays of fixed parameters with the appropriate codes reduced in the beginning of each string:

1 - the Vector of parameters of the program of integration (REAL*8)
2 - the Vector of parameters PAR (REAL*8)
3 - the Vector of offsets for the GIP (INTEGER*4)
4 - the Vector of offsets for models (INTEGER*4)

5 - the Vector of joining of units of system on I (INTEGER*4)
6 - the Vector of offsets for programs of calculation of days off to steams. (INTEGER*4)
7 - the Vector-index of system parameters (INTEGER*4)
8 – the Vector of structure of model (INTEGER*4)
9 – the Vector of structure of the GIP (INTEGER*4)

The file contains following arrays of variable parameters with the appropriate codes reduced in the beginning of each string:

Required parameters:
21 - the Vector X, V, A (REAL*8)
22 - the Vector of a new state (REAL*8)
23 - the Vector of an old state (REAL*8)
24 - Workers of vector WRK (REAL*8)
25 - Output values an OVP (REAL*8)
26 - Data for the GIP (REAL*8)
27 - Currents of models local (a vector I) (REAL*8)
28 - not Named COMMON-area (the mixed format corresponding to a format of not named area COMMON решателя. The order and type of variables is presented in section 2 of this deed in the description of structure of block COMMON).
29 - Vector COUNT (INTEGER*4)

Optional parameters:
41 - the Global Jacobian

The vector of structure of models has the following structure:

1. Length of all structure (INTEGER*4)
2. An amount of models (INTEGER*4)
3. A model name (CHARACTER*8)
4. ID models (INTEGER*4)
5. An amount of sites of model (INTEGER*4)
6. The list of numbers of sites of model (INTEGER*4)

This vector is the list of models containing the above described structure for each model.
The vector of structure of the GIP has the following structure:

1. An amount of the GIP
2. Number of model appropriate given to the GIP (if = 0, without model) (INTEGER*4)
3. A name of the GIP (CHARACTER*8)
4. An amount of parameters PARIMD (INTEGER*4)
5. The beginning of parameters PARIMD in array A (INTEGER*4)
6. An amount of parameters WRKIMD (INTEGER*4)
7. The beginning of parameters WRKIMD in array A (INTEGER*4)
8. An amount of parameters PARLR2 (INTEGER*4)
9. The beginning of parameters PARLR2 in array A (INTEGER*4)
10. ID THE GIP (INTEGER*4)
11. Parameter value UNV (INTEGER*4)

This vector is the list of the GIP containing the above described structure for each GIP.

The vector of parameters of the program of integration (a code of parameter 1) consists of following parameters: SAVE, END, OUT, SMAX, SMIN, DRLTI, DABSI, DRLTU, DABSI, DABSU, DRLTX, DABSX, FLAG, ITR, DEBUG, TIMER, CONTRL, MODE, CHANGE, PROGNZ, SCHEME, WEIGHT, SECOND, IGNORE, ATM, SCALE, CHECKM, PRTTIME, OUTPER, OUTVAR

Operation with reader of DAT-file

The appropriate package of class-rooms of contexts and structures is developed for operation with file DAT. The package consists of following class-rooms and structures:

SolverContext – the basic class-room controlling reading of a file, positioning on necessary a temporal shearing (Layer) a file. From it all remaining class-rooms are initialized. The plant of the given class-room allows to realise navigating on temporary stratums of a file and to gain constants and variable quantities. For navigating method SetLayer (int) which instal a context on an appropriate stratum is used. For obtaining of values of fixed parameters methods with appropriate names are used, and for obtaining of variable parameters methods GetVariableInt (int) and GetVariableDouble (int) are used. The Data-in of these methods is the code of the variable which values are reduced above in an argument list.

ModelContext – a context of models. It is intended for obtaining of the information concerning models. Plant ModelContext contains method SetModelNumber (int) allowing to realise navigating on models. A data-in of this method is the serial number of model which is in limits from 1 to N where N it is equal to an amount of models and is gained from a method int GetModelSize ().

DOFContext - a context of variables of type X, V, A. It is intended for obtaining of values X, V, A the chosen site on a current temporary shearing.

PGOContext - a context of the GIP. It is intended for obtaining of the information of concerning GIP. Plant PGOContext contains method SetPGONumber (int) allowing to realise navigating on the GIP. A data-in of this method is the serial number of the GIP which is in limits from 1 to N where N it is equal to an amount of the GIP and is gained from a method int GetPGOSize ().

As PGOContext contains model number to which given with the GIP concerns using this number it is possible to create plant ModelContext from which it is possible to gain all information concerning model with given number, including the list of sites. If model number is equal to null it means that there is no appropriate model and accordingly there are no degree of freedoms and it is an image static.

Structure for block COMMON:

```
struct COMMON
{
        double TIME, STEP, STEP01, STEP02, DT,
                 DABSI, DRLTI, STEPMD, TIMEND;
        char NAME [8];
        int NSTEP, SYSPRN, NITER, ITR;
        short CODE, NUMINT, NUMPRV, CODSTP, CODGRF,
                         NEWINT, MINSTP;
};
```

Structure for DOF:

```
struct DOF
{
double X, V, A;
};
```

SolverContext – contains following methods:

int GetVersion ();	To gain the file version
int GetCreationDate ();	To gain a file creation date
char* GetDescription ();	To gain the file description
int GetPermanentCodeTableLength ();	To gain length of the table of fixed parameters
int GetVariableCodeTableLength ();	To gain length of the table of variable parameters
int* GetPermanentCodeTable ();	To gain the table of fixed parameters
int* GetVariableCodeTable ();	To gain the table of variable parameters
int GetIntegrationParamLength ();	To gain a modulus of a vector of parameters of integration
double* GetIntegrationParam ();	To gain a vector of parameters of integration
int GetParLength ();	To gain a modulus of a vector of parameters PAR
double* GetPar ();	To gain a vector of parameters PAR
int GetASMILength ();	To gain modulus of a vector ASMI
int* GetASMI ();	To gain vector ASMI
int GetASMMLength ();	To gain modulus of a vector ASMM
int* GetASMM ();	To gain vector ASMM
int GetANUZLLength ();	To gain modulus of a vector ANUZL
int* GetANUZL ();	To gain vector ANUZL
int GetASMOLength ();	To gain modulus of a vector ASMO
int* GetASMO ();	To gain vector ASMO
int GetASYSOLength ();	To gain modulus of a vector ASYSO
int* GetASYSO ();	To gain vector ASYSO
int GetModelStructureLength ();	Size of a vector of structure of models
int * GetModelStructure ();	Vector of structure of models
int GetModelSize ();	Amount of models
char * GetModelNames ();	Names of models
int GetPRVPSize ();	Amount an OVP
int GetPGOSize ();	Amount of the GIP
char * GetPGONames ();	Names of the GIP

int * GetPGOStructure ();	Vector of structure of the GIP
int GetLayerSize ();	To gain an amount of stratums
int Refresh ();	State renewal
int SetLayer (int);	Installation of a current stratum
int* GetVariableInt (int);	To gain integer parameters
double* GetVariableDouble (int);	To gain real parameters
int GetVariableLength (int);	To gain length of parameter
COMMON GetCOMMON ();	To gain COMMON area
double GetTime ();	To gain a time of a current stratum
int GetCurrLayer ();	To gain a current stratum

ModelContext – contains following methods:

SetLayer (int Layer);	To instal a stratum
int SetModelNumber (int);	To instal number of current model
char * GetModelName ();	To gain a name of current model
int GetModelCode ();	To gain a code in a database of current model
int GetModelNodeSize ();	To gain an amount of degree of freedoms of current model
int * GetModelNodes ();	To gain numbers of degree of freedoms of current model
double * GetLocal_I ();	To gain currents of models local (a vector I)
int GetParLength ();	To gain an amount of parameters of model (vector PAR)
double * GetPar ();	To gain model parameters (vector PAR)
int GetWRKLength ();	To gain length WRK an array
int GetNewLength ();	To gain length New an array
int GetOldLength ();	To gain length Old an array
double * GetWRK ();	To gain array WRK
double * GetNew ();	To gain array New
double * GetOld ();	To gain array Old
int GetModelSize ();	To gain an amount of models

DOFContext – contains following methods:

SetLayer (int);	To instal a stratum

int GetLength ();	To gain an amount of sites
struct DOF GetDOF (int n);	To gain X, V, A for a site n

PGOContext – contains following methods:

SetLayer (int Layer);	To instal a stratum
int SetPGONumber (int);	To instal number of the current GIP
int GetPGOSize ();	To gain an amount of the GIP
char * GetPGOName ();	To gain a name of the current GIP
int GetModelNumber ();	To gain number of model of the appropriate current GIP (if номер=0 to any model does not belong)
int GetPGOCode ();	To gain a code in a database of the current GIP
int GetUNV ();	To gain parameter UNV
int GetPARIMDLength ();	To gain an amount of parameters PARIMD
double * GetPARIMD ();	To gain parameters PARIMD
int GetWRKIMDLength ();	To gain an amount of parameters WRKIMD
double * GetWRKIMD ();	To gain parameters WRKIMD
int GetPARLR2Length ();	To gain an amount of parameters PARLR2
double * GetPARLR2 ();	To gain parameters PARLR2
int GetColor ();	To gain colour

The text of the program illustrating operation with all methods set forth above of class-rooms is more low reduced:

```
//----------------SolverContext-----------------------
SolverContext SC ("C: \\dinama \\pradis32 \\swing.dat");

//---------------the Start block----------------------
cout <<"FileName =" <<SC.FileName <<endl;

cout <<"Version =" <<SC.GetVersion () <<endl;
if (SC.GetVersion ()! = 1) return-1;
cout <<"CreationDate =" <<SC.GetCreationDate () <<endl;
cout <<"Description =" <<SC.GetDescription () <<endl;
cout <<"PermanentCodeTableLength ="
<<SC.GetPermanentCodeTableLength () <<endl;
cout <<"VariableCodeTableLength =" <<SC.GetVariableCodeTableLength
() <<endl;
cout <<endl;

cout <<"PermanentCodeTable =";
for (int i=0; i <SC.GetPermanentCodeTableLength (); i ++)
   cout <<SC.GetPermanentCodeTable () [i] <<"";
cout <<endl;
cout <<endl;
```

```
cout <<"VariableCodeTable =";
for (i=0; i <SC.GetVariableCodeTableLength (); i ++)
   cout <<SC.GetVariableCodeTable () [i] <<"";
cout <<endl;
cout <<endl;

//----------Fixed parameters----------------------
cout <<"IntegrationParamLength =" <<SC.GetIntegrationParamLength ()
<<endl;
cout <<"IntegrationParam =";
for (i=0; i <SC.GetIntegrationParamLength (); i ++)
   cout <<SC.GetIntegrationParam () [i] <<"";
cout <<endl;
cout <<endl;

cout <<"ParLength =" <<SC.GetParLength () <<endl;
cout <<"Par =";
for (i=0; i <SC.GetParLength (); i ++)
   cout <<SC.GetPar () [i] <<"";
cout <<endl;
cout <<endl;


cout <<"ASMILength =" <<SC.GetASMILength () <<endl;
cout <<"ASMI =";
for (i=0; i <SC.GetASMILength (); i ++)
   cout <<SC.GetASMI () [i] <<"";
cout <<endl;
cout <<endl;

cout <<"ASMMLength =" <<SC.GetASMMLength () <<endl;
cout <<"ASMM =";
for (i=0; i <SC.GetASMMLength (); i ++)
   cout <<SC.GetASMM () [i] <<"";
cout <<endl;
cout <<endl;

cout <<"ANUZLLength =" <<SC.GetANUZLLength () <<endl;
cout <<"ANUZL =";
for (i=0; i <SC.GetANUZLLength (); i ++)
   cout <<SC.GetANUZL () [i] <<"";
cout <<endl;
cout <<endl;

cout <<"ASMOLength =" <<SC.GetASMOLength () <<endl;
cout <<"ASMO =";
for (i=0; i <SC.GetASMOLength (); i ++)
   cout <<SC.GetASMO () [i] <<"";
cout <<endl;
cout <<endl;

cout <<"ASYSOLength =" <<SC.GetASYSOLength () <<endl;
cout <<"ASYSO =";
for (i=0; i <SC.GetASYSOLength (); i ++)
   cout <<SC.GetASYSO () [i] <<"";
cout <<endl;
cout <<endl;

cout <<"ModelStructureLength =" <<SC.GetModelStructureLength ()
<<endl;
cout <<"ModelSize =" <<SC.GetModelSize () <<endl;
cout <<"ModelNames =" <<SC.GetModelNames () <<endl;
for (i=0; i <SC.GetModelStructureLength (); i ++)
```

```
   cout <<SC.GetModelStructure () [i] <<"";
cout <<endl;
cout <<endl;

   cout <<"PGOSize =" <<SC.GetPGOSize () <<endl;
cout <<"PGONames =" <<SC.GetPGONames () <<endl;
for (i=0; i <SC.GetPGOSize () * 7; i ++)
   cout <<SC.GetPGOStructure () [i] <<"";
cout <<endl;
cout <<endl;

//----------Variable parameters-----------------------
cout <<"LayerSize =" <<SC.GetLayerSize () <<endl;

cout <<"SetLayer =" <<SC.SetLayer (2) <<endl;
cout <<"SetLayer =" <<SC.SetLayer (500) <<endl;

cout <<"Refresh =" <<SC.Refresh () <<endl;
cout <<"LayerSize =" <<SC.GetLayerSize () <<endl;

cout <<"SetLayer =" <<SC.SetLayer (5) <<endl;
cout <<endl;

cout <<"IntVariableLength =" <<SC.GetVariableLength (29) <<endl;
cout <<"VariableInt =";
for (i=0; i <SC.GetVariableLength (29); i ++)
   cout <<SC.GetVariableInt (29) [i] <<"";
cout <<endl;
cout <<endl;

cout <<"SetLayer =" <<SC.SetLayer (20) <<endl;
cout <<"DoubleVariableLength =" <<SC.GetVariableLength (21) <<endl;
cout <<"VariableDouble =";
for (i=0; i <SC.GetVariableLength (21); i ++)
   cout <<SC.GetVariableDouble (21) [i] <<"";
cout <<endl;
cout <<endl;

char Name [9];
for (i=0; i <8; i ++) Name [i] =SC.GetCOMMON ().NAME [i];
Name [8 =0;
cout <<"Common =" <<SC.GetCOMMON ().CODE <<""
    <<Name <<"" <<SC.GetCOMMON ().NSTEP <<""
    <<SC.GetCOMMON ().NUMINT <<"" <<SC.GetCOMMON ().TIME <<endl;
cout <<endl;


cout <<"Time =" <<SC.GetTime () <<endl;
cout <<endl;

cout <<"PRVPSize =" <<SC.GetPRVPSize () <<endl;
cout <<"PGOSize =" <<SC.GetPGOSize () <<endl;
cout <<endl;

//==== DOFContext ==================
DOFContext DC (&SC);
DC.SetLayer (5);
cout <<"DOFContextLength =" <<DC.GetLength () <<endl;
struct DOF d = DC.GetDOF (3);
cout <<"DOF_X =" <<d. X <<endl;
cout <<"DOF_V =" <<d. V <<endl;
cout <<"DOF_A =" <<d. A <<endl;
cout <<endl;
```

```
//======== ModelContext ===========
ModelContext MC (&SC);
MC.SetLayer (5);
cout <<"SetModelNumber =" <<MC.SetModelNumber (2) <<endl;
cout <<"ModelName =" <<MC.GetModelName () <<endl;
cout <<"ModelCode =" <<MC.GetModelCode () <<endl;
cout <<"ModelNodeSize =" <<MC.GetModelNodeSize () <<endl;
cout <<"ModelNodes =";
for (i=0; i <MC.GetModelNodeSize (); i ++)
   cout <<MC.GetModelNodes () [i] <<"";
cout <<endl;
cout <<"Local_I =";
for (i=0; i <MC.GetModelNodeSize (); i ++)
   cout <<MC.GetLocal_I () [i] <<"";
cout <<endl;
cout <<"ParLength =" <<MC.GetParLength () <<endl;
cout <<"Par =";
for (i=0; i <MC.GetParLength (); i ++)
   cout <<MC.GetPar () [i] <<"";
cout <<endl;
cout <<"WRKLength =" <<MC.GetWRKLength () <<endl;
cout <<"WRK =";
for (i=0; i <MC.GetWRKLength (); i ++)
   cout <<MC.GetWRK () [i] <<"";
cout <<endl;
cout <<"NewLength =" <<MC.GetNewLength () <<endl;
cout <<"New =";
for (i=0; i <MC.GetNewLength (); i ++)
   cout <<MC.GetNew () [i] <<"";
cout <<endl;
cout <<"OldLength =" <<MC.GetOldLength () <<endl;
cout <<"Old =";
for (i=0; i <MC.GetOldLength (); i ++)
   cout <<MC.GetOld () [i] <<"";
cout <<endl;
cout <<endl;

//======== PGOContext ===========
PGOContext PC (&SC);
PC.SetLayer (5);
cout <<"SetPGONumber =" <<PC.SetPGONumber (1) <<endl;
cout <<"GetPGOSize =" <<PC.GetPGOSize () <<endl;
cout <<"GetPGOName =" <<PC.GetPGOName () <<endl;
cout <<"GetModelNumber =" <<PC.GetModelNumber () <<endl;
cout <<"GetPGOCode =" <<PC.GetPGOCode () <<endl;
cout <<"GetUNV =" <<PC.GetUNV () <<endl;
cout <<"GetPARIMDLength =" <<PC.GetPARIMDLength () <<endl;
for (i=0; i <PC.GetPARIMDLength (); i ++)
   cout <<PC.GetPARIMD () [i] <<"";
cout <<endl;
cout <<"GetWRKIMDLength =" <<PC.GetWRKIMDLength () <<endl;
for (i=0; i <PC.GetWRKIMDLength (); i ++)
   cout <<PC.GetWRKIMD () [i] <<"";
cout <<endl;
cout <<"GetPARLR2Length =" <<PC.GetPARLR2Length () <<endl;
for (i=0; i <PC.GetPARLR2Length (); i ++)
   cout <<PC.GetPARLR2 () [i] <<"";
cout <<endl;
cout <<"GetColor =" <<PC.GetColor () <<endl;
cout <<endl;
```