PRADIS

THE DESCRIPTION OF LANGUAGE Python PRADIS Language (PPL)

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

VERSION 4.3

Contents

Agreements on the notation

Following agreements on selection of elements of the text are used:

Semiboldface italic type	Labels of the basic concepts
`Courier`	Instances of command lines and texts of programs in language *PPL*
The semiboldface	"Pay attention"

At the description of syntax of an input language following labels are accepted:

[xxx]	The information is not obligatory
Italic type	The user should substitute a concrete parametre value

Program structure

PPL, per se, is the Python in whom subfunctions and class-rooms of library of a PRADIS are accessible. Their assignment – the description of structure of a prototype process.

Generally the program in language *PPL* can be dissected into three main bodies conditionally:

- The description of modelled installation
- The description of the image of installation
- The description of the job for calculation and representation of outcomes

According to it almost all functions and class-rooms of library of a PRADIS serve one of these purposes.

During the description of structure the installation model is represented in the form of a population of models of the elements which have been switched on in libraries of a complex. Elements incorporate among themselves in certain points of physical space. The condition of these points is characterised by a quantity (or ***a degree of freedom***) and their derivatives. The amount of a degree of freedom in each point of physical space depends generally on dimensions of a quantity of the space, used models of elements and a mode of the joint of these elements. Further in this deed concept ***the node*** is used for a label of a point of installation with the gang of a degree of freedom (from one to six).

At the description of each of elements the certain amount of nodes to which this element incorporates is set. When it is a question of a degree of freedom of a separate element, conveniently to operate with concept ***an element*** . So, further in this deed concept the element branch is used as a synonym of concept ***an element degree of freedom***.

During calculation *PPL* operates with the variables named further the interior. The magnitudes gained immediately during the solution refer to to ***interior variables***, for example, in the mechanic of force, migration, a velocity, acceleration etc., i.e. The amount of such variables even at insignificant sizes of model is great, and in many cases can attain hundreds and thousand. Therefore is inexpedient to save the information on each interior variable. Immediately to representation the magnitudes computed on various algorithms with use of interior variables and presented as ***output variables*** are accessible**.** The information on output variables is saved after a solving. Output variables are presented by means of special class-rooms**.**

In the program text in language *PPL* there is a possibility to define the image of installation during calculation. It becomes as by means of class-rooms specially intended for it. However, if the element has the graphical image on-default it it is not necessary presents separately, it will be automatically generated.

Instance of the elementary program in language *PPL*:

```
# Connection of library of a PRADIS
from include import *

# Tracing of errors in the program with the help
#  Exclusions
try:
      # The description of nodes
      p1 = Point2d ()
      p2 = p1.x
      pbase = DOF1 ()

      # The job of base nodes
      Base ([pbase])

      # The description of elements
      body = MD ([p1], [1, 1])
      d = Diagram (0, 0)
      d. Add (2, 10)
      Go = STABL0 ([p2, pbase], [1e6, d])

      # Calculation of output variables
       displ_x = SUM ([p2, p2.V (), p2.A (), body. W (1),
body. I (1), body. S (1)], [1])

      # The image job
      rb = pXY (0, 0)
      img = RECTD (body, [rb, 2, 4], LayerParams)

      # The calculation description
      rng = Range (displ_x,-10, 10)
      slv = SHTERM ([rng], end = 2.0, outvar = 3)

      # Printing of output variables
      prn = Print ([rng], frm = 1)

      # Command of generation of a file of the job in
language PSL
      $ RUN :

except af. LVPS_TException, e:
```

```
print e. GetError ()
```

Apparently from the reduced program, any special description of data it is not required (unlike PSL). The description of data transmitted in models, an OVP, the GIP becomes to meanses of language Python.

Modules

In library PRADIS the assemblage of models, the GIP and an OVP contains. Each of these elements is used for simulation of processes of this or that area: the mechanics, hydraulics etc. Accordingly each of elements belongs to this or that area. Elements from one area are merged in *modules*. Therefore to use in the program elements, for example, from hydraulics, it is necessary to connect the hydraulic index:

```
from hydro import *
```

For use of mechanical elements, anything to connect it is not necessary. They are accessible by default, as their majority.

Instances of modules:

1. **electronics**: the module of electric installations.
2. **hydro**: the hydraulic index.

THE CIRCUIT DESIGN DESCRIPTION

The circuit design – the important concept in PPL. Its analogue in PSL is the fragment. Per se, it presents all structure which we model a PRADIS in medium. At connection in the module program “include” (see above) there is a rooted (principal) circuit design in which there will be a simulation. The rooted circuit design can switch on in itself daughter circuit designs. Operation always happens in one current circuit design. The user can be switched between circuit designs, doing leaking that which will wish. We will observe an instance:

```
from include import *
# The current circuit design - rooted

sch1 = Scheme ()
# sch1 It is created in the rooted circuit design
# sch1 - Leaking

sch2 = Scheme ()
# sch2 It is created in the circuit design sch1
# sch2 - Leaking

sch3 = Scheme ()
# sch3 It is created in the circuit design sch2
# sch3 - Leaking

sch1.begin ()
# The current circuit design - sch1

glb.sch.end ()
# The current circuit design - sch3

glb.sch.end ()
# The current circuit design - sch2

glb.sch.end ()
# The current circuit design - sch1

glb.sch.end ()
```

```
# The current circuit design - rooted
```

Scheme is a class-room of the circuit design from PRADIS library. Apparently from an instance, any circuit design always is leaking and is marked out by a name “glb.sch”. At creation of the new circuit design, it forms in the current circuit design and becomes leaking. For sampling of the current circuit design exists two methods of class-room "Scheme": “begin” and “end”. The method “begin” assigns the circuit design leaking, and the method “end” chooses the circuit design preceding the leaking. Operation of both methods is well displayed in an instance.

The extent of an enclosure of circuit designs each other is not restricted. Each circuit design can contain following installations:

- Nodes.
- Models of elements.
- Output variables (OVP).
- Images of elements (GIP).
- Calculation programs.
- Print routines an OVP.

The description of nodes of the circuit design

Nodes are presented by creation of installations of class-rooms of various types of the nodes which have been switched on in library of a PRADIS. Here the list of available class-rooms:

- DOF. Simplis one degree of freedom.
- DOF1. A node with one degree of freedom. Has a field: “x”.
- XY. A node with two translational degree of freedom. Has fields: “x”, “y”.
- Point2d. A node with two translational and one rotational degree of freedom. Has fields: “x”, “y”, “r”.
- XYZ. A node with three translational degree of freedom. Has fields: “x”, “y”, “z”.
- RotatXYZ. A node with three rotational degree of freedom. Has fields: “rx”, “ry”, “rz”.
- Point. A node with three translational and three rotational degree of freedom. Has fields: “x”, “y”, “z”, “rx”, “ry”, “rz”.

All fields of the class-rooms set forth above are installations of class-room DOF. Instances of use of these class-rooms can be discovered in test jobs from folder DINAMA/test/PPL.

Often it is required, that fields of installations of nodes specified in the same degree of freedom if it goes into in some улов various type. For example, point migration on axis X can simultaneously go into type nodes: Point (a field ”x”), XYZ (a field ”x”), XY (a field ”x”), Point2d (a field ”x”) and DOF1 (a field ”x”). We Will instance. Let you have created two nodes – XYZ and XY, and want, that fields “y” at them specified in physically same degree of freedom. For this purpose it is necessary to write:

```
xyz = XYZ ()
xy = XY ()
xyz.y.Copy (xy, "y")
```

Or, that the same:

```
xyz = XYZ ()
xy = XY ()
xy.y.Copy (xyz, "y")
```

Other instances of use of a method “Copy” as can be discovered in catalogue DINAMA/test/PPL.

Frequently it is necessary to switch on the description of nodes in the program, the kinematic or other which potential performances are accepted for 0 (a potential, temperature, pressure etc.). For example, in the mechanic are timbered nodes, in pneumatics - the nodes connected to an aerosphere etc. Further such nodes we will name .

Structure of the description of base nodes of a fragment:

`Base ([`*a node* $_1$ [*a node* $_2$ [... *узел*$_n$]]])

узел$_j$ *number* of a node of the fragment presented as base.

Apparently, function “Base” has in parametre the list of nodes. It is possible and not to specify base nodes, if in them there is no necessity.

Instance .

```
pnt = Point ()
xyz = XYZ ()
dof = DOF ()
Base ([dof, pnt.x, pnt.ry, xyz.z])
```

Syntax of the description ***of exterior nodes*** of the circuit design (i.e. nodes of the circuit design which serve for turning on of the presented circuit design in circuit designs of higher level) is defined in the same way, only by means of function “External”. For example:

```
pnt = Point2d ()
xy = XY ()
dof = DOF ()
External ([dof, pnt.r, xy.x])
```

The description of models of elements

The description of models of the elements which are switched on in the circuit design, looks like the following:

The identifier = a name ([*узел*$_1$, *узел*$_2$..., *узел*$_n$], [*the list*])

The identifier THE IDENTIFIER of model of an element.

Name A NAME of model of an element from among the models which have been switched on in libraries of a complex.

узел$_j$ INSTALLATION of one of class-rooms of nodes from PRADIS library.

The list AN ARGUMENT LIST of model of an element.

In an argument list of model of an element, except numbers, there can be installations of various special class-rooms from PRADIS library. Behind each model of a complex the certain aspect of an argument list is fixed. In case of transfer of parametre of irregular type exclusion with explains that for an error has been admitted will be rejected. Here the list of these special class-rooms:

- Diagram. The class-room presenting table association. It is initialized by the first point of association (two numbers), adding of points is carried out by means of a method “Add”.
- pXY. The class-room setting co-ordinates of a 2-dimensional point. It is initialized by two numbers.
- Begin2d. The class-room setting co-ordinates of the origin of a 2-dimensional point with twirl. It is initialized by three numbers.
- pXYZ. The class-room setting co-ordinates of a 3-dimensional point. It is initialized by three numbers.
- TrapeziumData. The class-room setting parametres трапециидального of a pulse. It is initialized by six numbers.
- InertiaMoment. The class-room setting moments of inertia round three principal axises. It is initialized by three numbers.
- SimpleMaterial. The class-room setting property of a material. It is initialized by three numbers.

- ContactForceModel. The class-room defining forces of contact interacting. It is initialized by the list of numbers, first of which sets force type, and remaining - force parametres.

Instances of use of all these special class-rooms can be seen in test jobs from folder DINAMA\test\PPL.

Instance.

```
pntO = pXY (0, 0)
pntA = pXY (0.5,-0.5)
M = 1
J = 1

nd_2d_1 = Point2d ()
nd_2d_2 = Point2d ()
nd_2d_3 = Point2d ()

mayatn1 = BALKA ([nd_2d_1, nd_2d_2], [pntO, pntA, 1, 0.5,
                    1E-6, 1E-4, 2E11])
mass = MD ([nd_2d_3], [M, J])
```

There is still an alternative alternative of the description of models of elements.

The identifier = `AddModel ("name"`, [*узел*$_1$, *узел*$_2$..., *узел*$_n$], [*the list*])

As you can see, it differs only that it is necessary to write a command “AddModel” and a model name to write in ковычках. Here there is one nuance, the model name in this case should be written together with a module name in which this model is switched on, through a point. In remaining all the same. In the core this method is intended for a plug-in of models for which it is not included class-rooms in PPL. The description from the previous instance would look so:

```
mayatn1 = AddModel (“mechanics. BALKA”, [nd_2d_1, nd_2d_2],
                    [pntO, pntA, 1, 0.5, 1E-6, 1E-4, 2E11])
mass = AddModel (“mechanics. MD”, [nd_2d_3], [M, J])
```

The description of output variables

The description of an output variable (the description of a ringing of the matching program of calculation of an output variable) looks like the following:

The identifier = a name ([*переменная*$_1$ [*переменная*$_2$ [..., *переменная*$_n$]]], [*the list*])

The identifier THE IDENTIFIER of an output variable.

Name A NAME of the program of calculation of output variables from among the programs which have been switched on in libraries of a complex.

переменная$_j$ THE INDEX on *j*-ю the interior variable transmitted in the program of calculation of output variables. The amount and an order of indexes in the ringing description should match to their amount and a sequence in the documentation for the program of calculation of output variables. Syntax of the description of the index on an interior variable is defined more low.

The list AN ARGUMENT LIST of the program of calculation of output variables. The amount and sequence of parametres should match completely to amount and a sequence of the parametres defined in the documentation for the program of calculation of output variables.

Admissible indexes on interior variables are:

узел$_j$ ***the Node*** of the presented fragment, which potential performances are transmitted in the program of calculation of output variables (the basic potential variable and two its derivatives on a time). It is meant potential performances of a node:

In the mechanic node migration *узел*$_j$ and its derivatives on a time;

In hydraulics, a pneumatics integral on a time from pressure, pressure and its derivative on a time for *узла*$_j$;

In thermodynamics integral from temperature, temperature and its derivative on a time for *узла*$_j$;

Generally value of a variable (in which terms the solution of system of the differential equations is searched),

characterising a node condition, and values of its derivatives on a time.

узел$_j$`.V()` THE FIRST derivative on a time of the basic potential variable *узла*$_j$ *(*accordingly a velocity, pressure, temperature, a potential).

узел$_j$`.A()` A FLEXON on a time of the basic potential variable *узла*$_j$ *(*acceleration, a derivative from pressure, a derivative from temperature, a derivative from a potential).

Модель.I *(ветвь*$_j$*)* THE INDEX on ***the data-flow variable*** transmitted in the program of calculation of output variables. For an element with the identifier *the Model* characterises a condition *ветви*$_j$ this element. It is necessary to notice, that foliation of branches is local for an element, therefore value *ветвь*$_j$ should not exceed amount of nodes of this element. So, for a two-nodal element *ветвь*$_j$ can accept values 1 or 2. It is meant a data-flow variable:

In the mechanic force (moment) with which the system acts on an element. For definition of an indication of the index on a mechanical data-flow variable it is preferable to use the first or second notation of the index (with indications I or F);

In hydraulics, a pneumatics the charge or the magnitude equivalent to it directed from system to an element. For definition of an indication of the index on such data-flow variable it is preferable to use the first or third notation of the index (with an indication I or Q);

In thermodynamics the thermal stream directed from system to an element. As well as in the previous case, the first or third notation (with an indication I or Q) is preferable;

Generally value of a variable (concerning which the conservation relations - 3rd Newton's law, a Kirchhoff's law for currents etc. are formulated), characterising a

stream of this magnitude from system to an element. The first notation of a data-flow variable (with an indication I) is preferable.

Модель.W (*N*) THE INDEX on *N*-ю for an element with the identifier *the Model,* which value is required to be transmitted a component ***of a working vector*** in the program of calculation of output variables.

Модель.S (*N*) THE INDEX on *N*-ю for an element with the identifier is required to be transmitted *the Model*, which value to a component of a vector ***of a condition*** in the program of calculation of output variables.

Accessible components of a working vector of each model of an element are resulted in its description.

Instance. The description of ringings of programs of calculation of output variables can look as follows:

```
    rsA=ROUT ([kuzov. I (1), kuzov. I (2), kuzov. I (3)],
[1.5])
    vsk=X ([xyz11.z], [0.5])
    NXX=X ([kard. W (46)], [35.345])
```

In a reduced instance it is supposed, that elements with identifiers earlier have been defined:

```
    kuzov, kard
```

And as a node with the identifier:

```
    xyz11
```

THE INSTALLATION IMAGE

The description of a separate element looks as follows:

~идентификатор = the identifier (*a name,* [*список*$_1$], [*~список*])

~ The identifier THE IDENTIFIER the image.

The name THE IDENTIFIER of model of the element, which graphical image is set. If at the description of any graphical image the model identifier is not underlined (something is underlined except model, for example, “None”), it is considered, that the graphical image is motionless (is linked with motionless axes). If the element has a graphical image by default the image of this element is under construction automatically with use of the graphical image accepted for this element by default (a "standard" graphical image).

The identifier A NAME of a graphical image if for the installation image the standard graphical image or a graphical image of the user is used not.

список$_1$ AN ARGUMENT LIST of a non-standard graphical image. The amount and sequence of parametres should match completely to amount and a sequence of the parametres defined in the documentation under the program of implementation of a non-standard graphical image.

~список AN ARGUMENT LIST of the program of implementation of the image of an element. It contains only three elements which should follow as it should be «*colour, a material, a transparency*». For example, «[‘ YELLOW ’, ‘ plactic ’, 1.0]». These are parametres by default. They will be accepted, if you leave the list empty.

The description of the image of installation can look as in the instances resulted more low.

Instance

A2 = pXYZ (0.5, 0.5, 0)

C2m = pXYZ (0.5, 0.5, 1)
D2m = pXYZ (1.5, 0.5, 0)
LSK3D (T2, [A2, C2m, D2m, 0.2], [])
KN3EFV (CN, [33, 5, 5], [‘ YELLOW ’, ‘ plactic ’, 1.0])

THE JOB DESCRIPTION

Performance of calculation:

The description of each of ringings of the program of integration is characterised by a matching counted slice of time, parametres of an exactitude and regimes of representation of the information on a calculation course. For each program of integration the list of output variables operatively represented during calculation from defined earlier can be set. It is considered, that each subsequent program of integration continues calculation from that instant on which it has been discontinued by the previous program of integration. In complex *PPL* there is a possibility online to operate end of the program of integration. In case of interactive interruption the agreement on calculation prolongation by the subsequent programs of integration remains in force.

Saving of a current condition of calculation happens to the temporary pitch set by the user and, automatically, upon termination of integration.

Now the complex composition includes programs of integration of system of the differential equations of II th order an implicit method of Stormer and method Ньюмарка. The description of a ringing of the program of integration looks as follows:

~идентификатор = ~имя (*the list*, *имя*$_1$ = *number* [*имя*$_2$ = *number* [... *имя*$_n$ = *number*]])

~ The identifier THE IDENTIFIER of the program of integration.

~имя A NAME of the program of integration (“SHTERM”, “NEWMARK”).

имя$_j$ A NAME *of j th* key parametre from the list of the key parametres defined in the certificate of the program of integration.

Number A REAL NUMBER setting value of key parametre.

The list the List of the intervals which are operatively represented at calculation. The interval is the class-room "Range" initialized by an output variable and two numbers, minimum and maximum values.

Note Values of the key parametres which are not meeting in the description a ringing of the program of integration, are accepted by default equal to the values of these parametres specified in the certificate of the program of integration.

Instance. Descriptions of the program of integration can look as follows:

```
Nm=NEWMARK ([Range (ZA1,-11,1)], outvar=1, end=15, scale=1)
```

```
rx = Range (x,-2,2)
ry = Range (y,-2,2)
rz = Range (z,-1,1)
SHTERM ([rx, ry, rz], outvar=1, end = 4, smax = 1E-2, drlti
        = 0.0001, dabsi = 0.001, drltu = 0.0001, dabsu =
        0.001, drltx = 0.0001, dabsx = 0.001)
```

Representation of outcomes

The description of a ringing of the program of representation looks as follows:

`DISP` (*the list*, *имя*$_1$ = *number* [*имя*$_2$ = *number* [... *имя*$_n$ = *number*]])

имя$_j$ A NAME *of j th* key parametre from the list of the key parametres defined in the documentation under the program of representation.

Number A REAL NUMBER setting new value of key parametre.

The list the List of intervals. The interval is the class-room "Range" initialized by an output variable and two numbers, minimum and maximum values.

Instance. Descriptions of ringings of programs of representation can look as follows:

```
DISP ([Range (FIZG,-90,90),
      Range (DZ,-1e10,1e10),
      Range (DR,-1e10,1e10),
      Range (FZ,-1e10,1e10),
      Range (FR,-1e10,1e10),],
      start=0.0)
```

COMMAND USE import

Sometimes happens necessary to set a part of data in other file because of a great many of parametres. For this purpose it is convenient to use a command of a python “import”. Use mechanism is well displayed in test jobs “kn3ef.py” and “kn3ff.py” from catalogue DINAMA/test/PPL.

Instance.

Let's display two fragments from files. First from an optional file, second from the main thing.

```
include_1 = [280, 496,
pXYZ (0.9375, 0, 0),
pXYZ (1, 0, 0),
...


from kn3ef_1 import *
FORM2 = [A2, B2f, C2f, D2f, include_1, include_2]
```

INSTANCES OF PROGRAMS IN LANGUAGE *PPL*

The program containing the description of installation and the description of the job for calculation and representation of outcomes

Mathematical model of a rod pendulum with an elastic leg and concentrated to the pendulum extremities pointwise inertia elements. With a pendulum axis through the reductor the propeller incorporates to a linear speed-torque characteristic.

```
# We connect PRADIS library
from include import *

# We catch exclusions
try:
     # We present a node of 2-dimensional space
     p1 = Point2d ()

     # To component x a 2-dimensional node it is marked out
separately for
     # Further use
     p2 = p1.x

     # We present a degree of freedom which will be base
     pbase = DOF1 ()

     # We set a base node
     Base ([pbase])

     # We present model MD with one node and two parametres
     body = MD ([p1], [1, 1])

     # We set table association the special class-room
     # We initialize the first point of association
     d = Diagram (0, 0)


     # We add a point in association
     d. Add (2, 10)

     # We present model STABL0 with two nodes and two
     # In parametres
     Go = STABL0 ([p2, pbase], [1e6, d])

     # We present an output variable with six interior
```

```
     # Variables and in one parametre
     displ_x = SUM ([p2, p2.V (), p2.A (), body. W (1), body. I
     (1), body. S (1)], [1])

     # We set image parametres
     colour = ' gold '
     transp = 1.0
     material = ' plastic '

     LayerParams = colour, material, transp

     # We set image co-ordinates of the origin the special
     # The class-room
     rb = pXY (0, 0)

     # We present element images body with three
     # In parametres
     img = RECTD (body, [rb, 2, 4], LayerParams)

     # We present a variable interval displ_x
     rng = Range (displ_x,-10, 10)

     # We present an integration method
     slv = SHTERM ([rng], end = 2.0, outvar = 3)

     # We present printing of outcomes
     prn = DISP ([rng], frm = 1)

     # Command on job generation on PSL
     $ RUN :

except af. LVPS_TException, e:
     # If there was an error, type it
     print e. GetError ()
```

Here that will be gained after job generation on PSL from this instance:

```
$ DATA:

 $ FRAGMENT:
# BASE : 11
#STRUCTURE:

{Here two models which we presented}
Model_1 ' MD (8,9,10; 1,1)
Model_2 ' STABL0 (8,11; 1e+06,0,0,2,10)

#EXTERNAL:

# OUTPUT :

{Here an OVP which we presented}
OutVariable_1 ' SUM (8,8 ', 8 ", W:Model_1 (1), I:Model_1 (1),
S:Model_1 (1); 1)


#MAP
 $ SHOW:

{Here the image}
Image_1 ' LAYER (Model_1 (RECTD; 0,0,2,4); 1,7,0,0, 0,0,0,
0,0,0,14)


 $ $ RUN :

{The description of a method of integration}
Solver_1 ' SHTERM (END=2, OUTVAR=3; OutVariable_1 = (-10,10))



 $ $ PRINT :

{The description of the job for printing}
Print_1 ' DISP (FROM=1; OutVariable_1 = (-10,10))


 $ END
```

TEST JOBS IN LANGUAGE *PPL*

After PRADIS installation in catalogue DINAMA/test/PPL you can discover test jobs on PPL. They were already mentioned in this deed above. In this section we will give the short description to each of those jobs.

BAL.PY

The job models driving of four girders presented by different models. It is necessary to pay attention, that function “Base” is called some times in different places. This advantage of language PPL in comparison with PSL (there base nodes can appear only in one place). As it is necessary to notice, that the image is explicitly presented only for one girder while all are drawn four. Such it is is possible thanks to new possibilities PPL, it shapes all possible graphical images by default automatically.

BELT.PY

The job models filament driving. Special class-room "Diagram" is actively used by transfer of parametres. As at the description of a method of integration in the first parametre the empty list of intervals is transmitted. Pay attention, that the list can be and empty, but it should be, and necessarily first.

KARDAN1.PY

The job models cardan driving. It is necessary to pay attention to method use “Copy” for a label of an equivalent degree of freedom in different nodes.

KN3EF.PY

The job models collisions of an orb with a plane. Will pay special attention on command use “import” for transfer of a great many of parametres to model. And as on use of special class-room ContactForceModel.

KN3FF.PY

The job models collisions of two surfaces. The special attention should be converted on a mode of data read-out from additional files by means of function “GetXYZ”.

PSV3KT.PY

The complicated job with a great many of parametres, nodes and models. Special class-rooms and a method “Copy” for a label of an equivalent degree of freedom are very actively used. As at the description the OVP is used a method “I” for a label of data-flow variables. Special attention it is necessary to convert that at the image description “OPORAD” is transmitted by the first parametre “None” that means that the image is not linked with model, i.e. it is motionless. Besides, in all descriptions of images in last parametre the empty list (an image argument list) is transmitted. In such cases image parametres by default are used.

ROT3_2.PY

The job displays operation of model ROT3. Contains many descriptions various an OVP.

SWING.PY

The job with a good obvious picture.

TATABN3.PY

The job displays operation of model STRGN. At the description the OVP is used a method "W" for a label of components of a working vector.

TCYL_M.PY
The job displays operation of model CYLDR.

TROT6.PY
The job displays operation of model ROT1.

PPL vs. PSL

In this section it is displayed, that PPL on level of the interface and flexibility 10 times more, than PSL. We will display it on an instance of test job DINAMA/test/PPL/condit.py. That PPL is grounded on the Python, gives huge advantages in comparison with PSL. To the full it is visible in the job condit.py. Briefly we will present it.

The job models driving of system from cargoes connected by springs. It would Seem, anything complicated, however there are some singularities. Springs not absolutely usual, and with performance the force from a strain set is table. Pay attention to how the table is set. It is set in a scraper:

```
prug_tabl = Diagram (0, 0)
for i in range (1,11):
     prug_tabl. Add (i * 0.1, i * i * 0.01)
     prug_tabl. Add (-i * 0.1,-i * i * 0.01)
```

As you can see, square association is set not linear, but. Thus, it is possible to set easily any table associations, and with any exactitude, without special force. In a scraper it is possible to make 10 points, and it is possible 1000, the program is not becomes more complicated and more. That you will not tell about PSL. As pay attention, that points can be added not in ascending order argument, and in arbitrary – their classifying happens automatically.

One more important singularity of the program what even the prototype system topology can vary at change of one numeral. In the program there is a flag “FLAG”. At value 4 the system from four cargoes is presented, at any by the friend – from three. It is is possible thanks to the human controller of the Python “if”:

```
for i in range (3):
     pnt.append (XYZ ())
if FLAG == 4:
     pnt.append (XYZ ())
```

Track closely as it becomes some times in the program.

Thus, the big possibilities in the description of systems thanks to that PPL is grounded in programming language Python open.