

PRADIS

ОПИСАНИЕ ЯЗЫКА Python PRADIS Language (PPL)

**ПРОГРАММНЫЙ КОМПЛЕКС ДЛЯ АВТОМАТИЗАЦИИ
МОДЕЛИРОВАНИЯ НЕСТАЦИОНАРНЫХ ПРОЦЕССОВ В
МЕХАНИЧЕСКИХ СИСТЕМАХ И СИСТЕМАХ ИНОЙ
ФИЗИЧЕСКОЙ ПРИРОДЫ**

ВЕРСИЯ 4.3

Содержание

1. Соглашения о нотации.....	3
2. Структура программы.....	4
3. Модули.....	7
4. ОПИСАНИЕ СХЕМЫ	8
4.1. Описание узлов схемы	10
4.2. Описание моделей элементов	12
4.3. Описание выходных переменных.....	14
5. ИЗОБРАЖЕНИЕ ОБЪЕКТА	18
6. ОПИСАНИЕ ЗАДАНИЯ	20
6.1. Выполнение расчета	20
6.2. Отображение результатов	22
7. ИСПОЛЬЗОВАНИЕ КОМАНДЫ import.....	23
8. ПРИМЕРЫ ПРОГРАММ НА ЯЗЫКЕ PPL.....	24
8.1. Программа, содержащая описание объекта и описание задания на расчет и отображение результатов.....	24
9. ТЕСТОВЫЕ ЗАДАНИЯ НА ЯЗЫКЕ PPL.....	27
10. PPL vs. PSL.....	29

1. Соглашения о нотации

Используются следующие соглашения о выделении элементов текста:

полужирный курсив

обозначения основных понятий

Courier

примеры командных строк и текстов
программ на языке *PPL*

полужирный

“обратите внимание”

При описании синтаксиса входного языка приняты следующие обозначения:

[xxx]

информация не является обязательной

курсив

пользователь должен подставить
конкретное значение параметра

2. Структура программы

PPL, по сути, является Питоном, в котором доступны дополнительные функции и классы библиотеки Прадиса. Их назначение – описание структуры моделируемого процесса.

В общем случае программу на языке *PPL* можно условно разбить на три основные части:

- Описание моделируемого объекта
- Описание изображения объекта
- Описание задания на расчет и отображение результатов

В соответствии с этим почти все функции и классы библиотеки Прадиса служат одной из этих целей.

В ходе описания структуры модель объекта представляется в виде совокупности моделей элементов, включенных в библиотеки комплекса. Элементы соединяются между собой в определенных точках физического пространства. Состояние этих точек характеризуется некоторым количеством *обобщенных координат* (или *степеней свободы*) и их производных. Количество степеней свободы в каждой точке физического пространства зависит в общем случае от размерности пространства, используемых моделей элементов и способа соединения этих элементов. В дальнейшем в этом документе понятие *узел* используется для обозначения точки объекта со своим набором степеней свободы (от одной до шести).

При описании каждого из элементов задается определенное количество узлов, с которыми этот элемент соединяется. Когда речь идет о степенях свободы отдельного элемента, удобно оперировать понятием *ветвь элемента*. Итак, в дальнейшем в этом документе понятие *ветвь элемента* используется как синоним понятия *степень свободы элемента*.

В ходе расчета *PPL* оперирует переменными, называемыми далее внутренними. Ко *внутренним переменным* относятся, например, в механике силы, перемещения, скорости, ускорения и т.д., т.е. величины, получаемые непосредственно в ходе решения. Количество таких переменных даже при незначительных размерах модели велико, а во многих случаях может достигать сотен и тысяч. Поэтому нецелесообразно сохранять информацию о каждой внутренней переменной. Непосредственно доступными для отображения являются величины, вычисляемые по различным алгоритмам с использованием внутренних переменных и описанные как

выходные переменные. Информация о выходных переменных сохраняется после проведения расчетов. Выходные переменные описываются при помощи специальных классов.

В тексте программы на языке *PPL* есть возможность определить изображение объекта в ходе расчета. Это делается так же при помощи специально предназначенных для этого классов. Однако, если элемент имеет свой графический образ по-умолчанию, то его не нужно описывать отдельно, он будет сгенерирован автоматически.

Пример простейшей программы на языке *PPL*:

```
# Подключение библиотеки Прадиса
from include import *

# Отслеживание ошибок в программе при помощи
# исключений
try:
    # Описание узлов
    p1 = Point2d()
    p2 = p1.x
    pbase = DOF1()

    # Задание базовых узлов
    Base ([pbase])

    # Описание элементов
    body = MD ([p1], [1, 1])
    d = Diagram (0, 0)
    d.Add (2, 10)
    Go = STABL0 ([p2, pbase], [1e6, d])

    # Расчёт выходных переменных
    displ_x = SUM ([p2, p2.V(), p2.A(), body.W(1),
body.I(1), body.S(1)], [1])

    # Задание изображения
    rb = pXY(0, 0)
    img = RECTD (body, [rb, 2, 4], LayerParams)

    # Описание расчёта
    rng = Range (displ_x, -10, 10)
    slv = SHTERM ([rng], end = 2.0, outvar = 3)

    # Печать выходных переменных
    prn = Print ([rng], frm = 1)

    # Команда генерации файла задания на языке PSL
    Run()
```

```
except af.LVPS_TException,e:  
  
    print e.GetError()
```

Как видно из приведённой программы, никакого особого описания данных не требуется (в отличие от PSL). Описание данных передаваемых в модели, ПРВП, ПГО делается посредством языка Python.

3. Модули

В библиотеке PRADIS содержится множество моделей, ПГО и ПРВП. Каждый из этих элементов используется для моделирования процессов из той или иной области: механика, гидравлика и т.д. Соответственно каждый из элементов принадлежит к той или иной области. Элементы из одной области объединены в *модули*. Поэтому чтобы использовать в программе элементы, например, из гидравлики, надо подключить гидравлический модуль:

```
from hydro import *
```

Для использования механических элементов, ничего подключать не нужно. Они доступны по умолчанию, так как их большинство.

Примеры модулей:

1. **electronics**: модуль электрических объектов.
2. **hydro**: гидравлический модуль.

4. ОПИСАНИЕ СХЕМЫ

Схема – важное понятие в PPL. Её аналогом в PSL является фрагмент. По сути, она описывает всю структуру, которую мы моделируем в среде Прадис. При подключении в программу модуля “include” (см. выше) создаётся корневая (главная) схема, в которой будет происходить моделирование. Корневая схема может в себя включать дочерние схемы. Работа всегда происходит в одной текущей схеме. Пользователь может переключаться между схемами, делая текущей ту, которую пожелает. Рассмотрим пример:

```
from include import *
# Текущая схема - корневая

sch1 = Scheme ()
# sch1 создана в корневой схеме
# sch1 - текущая

sch2 = Scheme ()
# sch2 создана в схеме sch1
# sch2 - текущая

sch3 = Scheme ()
# sch3 создана в схеме sch2
# sch3 - текущая

sch1.begin()
# текущая схема - sch1

glb.sch.end()
# текущая схема - sch3

glb.sch.end()
# текущая схема - sch2

glb.sch.end()
# текущая схема - sch1
```



```
glb.sch.end()  
# текущая схема – корневая
```

Scheme – это класс схемы из библиотеки Прадиса. Как видно из примера, какая-то схема всегда является текущей и обозначается именем “glb.sch”. При создании новой схемы, она создаётся в текущей схеме и становится текущей. Для выбора текущей схемы существует два метода класса “Scheme”: “begin” и “end”. Метод “begin” назначает схему текущей, а метод “end” выбирает схему, предшествовавшую текущей. Работа обоих методов хорошо показана в примере.

Степень вложенности схем друг в друга не ограничена. Каждая схема может содержать следующие объекты:

- Узлы.
- Модели элементов.
- Выходные переменные (ПРВП).
- Изображения элементов (ПГО).
- Программы расчёта.
- Программы печати ПРВП.

4.1. Описание узлов схемы

Узлы описываются путём создания объектов классов различных типов узлов, включённых в библиотеку Прадиса. Вот список имеющихся классов:

- DOF. Просто одна степень свободы.
- DOF1. Узел с одной степенью свободы. Имеет поле: "x".
- XY. Узел с двумя поступательными степенями свободы. Имеет поля: "x", "y".
- Point2d. Узел с двумя поступательными и одной вращательной степенью свободы. Имеет поля: "x", "y", "r".
- XYZ. Узел с тремя поступательными степенями свободы. Имеет поля: "x", "y", "z".
- RotatXYZ. Узел с тремя вращательными степенями свободы. Имеет поля: "rx", "ry", "rz".
- Point. Узел с тремя поступательными и тремя вращательными степенями свободы. Имеет поля: "x", "y", "z", "rx", "ry", "rz".

Все поля вышеперечисленных классов являются объектами класса DOF. Примеры использования этих классов можно найти в тестовых заданиях из папки DINAMA/test/PPL.

Часто требуется, чтобы поля объектов узлов указывали на одну и ту же степень свободы, если она входит в несколько узлов различного типа. Например, перемещение точки по оси X может одновременно входить в узлы типа: Point (поле "x"), XYZ (поле "x"), XY (поле "x"), Point2d (поле "x") и DOF1 (поле "x"). Приведём пример. Пусть Вы создали два узла – XYZ и XY, и хотите, чтобы поля "y" у них указывали на физически одну и ту же степень свободы. Для этого надо написать:

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

Или, что то же самое:

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

Другие примеры использования метода “Cory” так же можно найти в каталоге DINAMA/test/PPL.

Зачастую в программу надо включать описание узлов, кинематические либо другие потенциальные характеристики которых принимаются за 0 (потенциал, температура, давление и т.д.). Например, в механике - это закрепленные узлы, в пневматике - узлы, соединенные с атмосферой и т.д. В дальнейшем такие узлы будем называть **базовыми**.

Структура описания базовых узлов фрагмента:

```
Base ([узел1 [узел2 [, ... узелn ]]])
```

узел_j Номер узла фрагмента, описываемого как базовый.

Как видно, функция “Base” имеет параметром список узлов. Базовые узлы можно и не указывать, если в них нет необходимости.

Пример .

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

Синтаксис описания **внешних узлов** схемы (т.е. узлов схемы, которые служат для включения описываемой схемы в схемы более высокого уровня) определяется точно так же, только при помощи функции “External”. Например:

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

4.2. Описание моделей элементов

Описание моделей элементов, включаемых в схему, имеет следующий вид :

идентификатор = *имя* ([*узел₁*, *узел₂* ... *узел_n*], [*список*])

<i>идентификатор</i>	Идентификатор модели элемента.
<i>имя</i>	Имя модели элемента из числа моделей, включенных в библиотеки комплекса.
<i>узел_j</i>	Объект одного из классов узлов из библиотеки Прадиса.
<i>список</i>	Список параметров модели элемента.

В списке параметров модели элемента, кроме чисел, могут быть объекты различных специальных классов из библиотеки Прадиса. За каждой моделью комплекса закреплён определённый вид списка параметров. В случае передачи параметра неправильного типа будет выброшено исключение с пояснениями о том, что за ошибка была допущена. Вот список этих специальных классов:

- **Diagram.** Класс, описывающий табличную зависимость. Инициализируется первой точкой зависимости (двумя числами), добавление точек осуществляется с помощью метода “Add”.
- **pXY.** Класс, задающий координаты 2-мерной точки. Инициализируется двумя числами.
- **Begin2d.** Класс, задающий начальные координаты 2-мерной точки с вращением. Инициализируется тремя числами.
- **pXYZ.** Класс, задающий координаты 3-мерной точки. Инициализируется тремя числами.
- **TrapeziumData.** Класс, задающий параметры трапециидального импульса. Инициализируется шестью числами.
- **InertiaMoment.** Класс, задающий моменты инерции вокруг трёх главных осей. Инициализируется тремя числами.
- **SimpleMaterial.** Класс, задающий свойство материала. Инициализируется тремя числами.

- **ContactForceModel.** Класс, определяющий силы контактного взаимодействия. Инициализируется списком чисел, первое из которых задаёт тип силы, а остальные - параметры силы.

Примеры использования всех этих специальных классов можно увидеть в тестовых заданиях из папки DINAMA\test\PPL.

Пример.

```
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])
```

Есть ещё альтернативный вариант описания моделей элементов.

```
идентификатор = AddModel ("имя",[узел1,узел2 ... ,узелn ], [список])
```

Как видите, он отличается только тем, что нужно писать команду “AddModel” и имя модели писать в кавычках. Здесь есть один нюанс, имя модели в данном случае должно писаться вместе с именем модуля, в который включена эта модель, через точку. В остальном всё то же самое. В основном этот метод предназначен для плагинов моделей, для которых не заложено классов в PPL. Описание из предыдущего примера выглядело бы так:

```
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])
```

4.3. Описание выходных переменных

Описание выходной переменной (описание вызова соответствующей программы расчета выходной переменной) имеет следующий вид:

идентификатор = *имя* ([*переменная*₁ [,*переменная*₂ [... ,*переменная*_{*n*}]]], [*список*]
)

<i>идентификатор</i>	Идентификатор выходной переменной.
<i>имя</i>	Имя программы расчета выходных переменных из числа программ, включенных в библиотеки комплекса.
<i>переменная</i> _{<i>j</i>}	Указатель на <i>j</i> -ю внутреннюю переменную, передаваемую в программу расчета выходных переменных. Количество и порядок указателей в описании вызова должны соответствовать их количеству и порядку следования в документации для программы расчета выходных переменных. Синтаксис описания указателя на внутреннюю переменную определен ниже.
<i>список</i>	Список параметров программы расчета выходных переменных. Количество и последовательность параметров должны полностью соответствовать количеству и порядку следования параметров, определенных в документации для программы расчета выходных переменных.

Допустимыми указателями на внутренние переменные являются:

<i>узел</i> _{<i>j</i>}	Узел описываемого фрагмента, потенциальные характеристики которого передаются в программу расчета выходных переменных (основная потенциальная переменная и две ее производных по времени). Под потенциальными характеристиками узла подразумевается :
---------------------------------	--

в механике

перемещение узла *узел*_{*j*} и его производные по времени;

в гидравлике, пневматике	интеграл по времени от давления, давление и его производная по времени для узла _j ;
в термодинамике	интеграл от температуры, температура и ее производная по времени для узла _j ;
в общем случае	значение переменной (в терминах которой ищется решение системы дифференциальных уравнений), характеризующей состояние узла, и значения ее производных по времени.
узел _j . V ()	Первая производная по времени основной потенциальной переменной узла _j (соответственно скорость, давление, температура, потенциал).
узел _j . A ()	Вторая производная по времени основной потенциальной переменной узла _j (ускорение, производная от давления, производная от температуры, производная от потенциала).
Модель. I (ветвь _j)	Указатель на потокową переменную , передаваемую в программу расчета выходных переменных. Для элемента с идентификатором <i>Модель</i> характеризует состояние ветви _j этого элемента. Необходимо заметить, что нумерация ветвей является локальной для элемента, поэтому значение ветвь _j не должно превышать количества узлов этого элемента. Так, для двухузлового элемента ветвь _j может принимать значения 1 или 2. Под потоковой переменной подразумевается:
в механике	сила (момент), с которыми система действует на элемент. Для определения признака указателя на механическую потоковую переменную предпочтительно использовать первую или вторую форму записи указателя (с признаками I или F);

в гидравлике, пневматике расход или эквивалентная ему величина, направленная от системы к элементу. Для определения признака указателя на такую потоковую переменную предпочтительно использовать первую или третью форму записи указателя (с признаком I или Q);

в термодинамике тепловой поток, направленный от системы к элементу. Как и в предыдущем случае, предпочтительна первая или третья форма записи (с признаком I или Q);

в общем случае значение переменной (относительно которой сформулирован закон сохранения - 3-й закон Ньютона, закон Кирхгофа для токов и т.д.), характеризующее поток этой величины от системы к элементу. Предпочтительна первая форма записи потоковой переменной (с признаком I).

Модель.W(N) Указатель на *N*-ю компоненту **рабочего вектора** для элемента с идентификатором *Модель*, значение которой требуется передать в программу расчета выходных переменных.

Модель.S(N) Указатель на *N*-ю компоненту **вектора состояния** для элемента с идентификатором *Модель*, значение которой требуется передать в программу расчета выходных переменных.

Доступные компоненты рабочего вектора каждой модели элемента приводятся в ее описании.

Пример. Описание вызовов программ расчета выходных переменных может выглядеть следующим образом :

```
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])
```

В приведенном примере предполагается, что ранее были определены элементы с идентификаторами:

kuzov, kard

а так же узел с идентификатором :

xyz11

5. ИЗОБРАЖЕНИЕ ОБЪЕКТА

Описание отдельного элемента выглядит следующим образом :

\sim идентификатор = идентификатор(имя, [список₁], [\sim список])

\sim идентификатор	Идентификатор изображение.
имя	Идентификатор модели элемента, графический образ которой задаётся. Если при описании какого-либо графического образа идентификатор модели не указывается (указывается что-либо кроме модели, например, “None”), считается, что графический образ неподвижен (связан с неподвижной системой координат). Если элемент имеет графический образ по умолчанию, то изображение этого элемента строится автоматически с использованием графического образа, принятого для этого элемента по умолчанию (“стандартный” графический образ).
идентификатор	Имя графического образа, если для изображения объекта используется не стандартный графический образ или графический образ пользователя.
список ₁	Список параметров нестандартного графического образа. Количество и последовательность параметров должны полностью соответствовать количеству и порядку следования параметров, определенных в документации по программе реализации нестандартного графического образа.
\sim список	Список параметров программы реализации изображения элемента. Он содержит только три элемента, которые должны следовать в порядке «цвет, материал, прозрачность». Например, «[‘YELLOW’, ‘plactic’, 1.0]». Это параметры по умолчанию. Они будут приняты, если вы оставите список пустым.

Описание изображения объекта может выглядеть как в приведенных ниже примерах.

Пример

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])

6. ОПИСАНИЕ ЗАДАНИЯ

6.1. Выполнение расчета

Описание каждого из вызовов программы интегрирования характеризуется соответствующим рассчитываемым интервалом времени, параметрами точности и режимами отображения информации по ходу расчета. Для каждой программы интегрирования может быть задан список оперативно отображаемых в ходе расчета выходных переменных из определенных ранее. Считается, что каждая последующая программа интегрирования продолжает расчет с того момента времени, на котором он был прерван предыдущей программой интегрирования. В комплексе *PPL* имеется возможность интерактивно управлять завершением программы интегрирования. В случае интерактивного прерывания соглашение о продолжении расчета последующими программами интегрирования остается в силе.

Сохранение текущего состояния расчета происходит с временным шагом, задаваемым пользователем и, автоматически, по окончании интегрирования.

В настоящее время в состав комплекса входят программы интегрирования системы дифференциальных уравнений II-го порядка неявным методом Штермера и методом Ньюмарка. Описание вызова программы интегрирования выглядит следующим образом :

$\sim \text{идентификатор} = \sim \text{имя} (\text{список}, \text{имя}_1 = \text{число} [, \text{имя}_2 = \text{число} [, \dots \text{имя}_n = \text{число}]])$

$\sim \text{идентификатор}$ Идентификатор программы интегрирования.

$\sim \text{имя}$ Имя программы интегрирования (“SHTERM”, “NEWMARK”).

имя_j Имя j -го ключевого параметра из списка ключевых параметров, определенных в паспорте программы интегрирования.

число Действительное число, задающее значение ключевого параметра .

список Список интервалов, оперативно отображающихся при расчёте. Интервал – это класс “Range”, инициализируемый выходной переменной и двумя числами, минимальным и максимальным значениями.

Примечание. Значения ключевых параметров, не встречающихся в описании вызова программы интегрирования, принимаются по умолчанию равными значениям этих параметров, указанным в паспорте программы интегрирования.

Пример. Описания программы интегрирования могут выглядеть следующим образом :

```
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)
```

6.2. Отображение результатов

Описание вызова программы отображения выглядит следующим образом:

`DISP (список, имя1 = число [, имя2 = число [...имяn = число])`

имя_j Имя *j*-го ключевого параметра из списка ключевых параметров, определенных в документации по программе отображения.

число Действительное число, задающее новое значение ключевого параметра.

список Список интервалов. Интервал – это класс “Range”, инициализируемый выходной переменной и двумя числами, минимальным и максимальным значениями.

Пример. Описания вызовов программ отображения могут выглядеть следующим образом:

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

7. ИСПОЛЬЗОВАНИЕ КОМАНДЫ `import`

Иногда бывает необходимым задавать часть данных в другом файле из-за большого количества параметров. Для этого удобно пользоваться командой питона “`import`”. Механизм использования хорошо показан в тестовых заданиях “`kn3ef.py`” и “`kn3ff.py`” из каталога `DINAMA/test/PPL`.

Пример.

Покажем два отрывка из файлов. Первый из вспомогательного файла, второй из главного.

```
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]
```

8. ПРИМЕРЫ ПРОГРАММ НА ЯЗЫКЕ *PPL*

8.1. Программа, содержащая описание объекта и описание задания на расчет и отображение результатов

Математическая модель стержневого маятника с упругой опорой и сосредоточенными на концах маятника точечными инерционными элементами. С осью маятника через редуктор соединяется двигатель с линейной механической характеристикой.

```
# Подключаем библиотеку Прадиса
from include import *

# Ловим исключения
try:
    # Описываем узел 2-мерного пространства
    p1 = Point2d()

    # Компоненту x 2-мерного узла обозначаем отдельно для
    # дальнейшего использования
    p2 = p1.x

    # Описываем степень свободы, которая будет базовой
    pbase = DOF1()

    # Задаём базовый узел
    Base ([pbase])

    # Описываем модель MD с одним узлом и двумя параметрами
    body = MD ([p1], [1, 1])

    # Задаём табличную зависимость специальным классом
    # Инициализируем первой точкой зависимости
    d = Diagram (0, 0)

    # Добавляем точку в зависимость
    d.Add (2, 10)

    # Описываем модель STABL0 с двумя узлами и двумя
    # параметрами
    Go = STABL0 ([p2, pbase], [1e6, d])

    # Описываем выходную переменную с шестью внутренними
```



```

# переменными и одним параметром
displ_x = SUM ([p2, p2.V(), p2.A(), body.W(1), body.I(1),
body.S(1)], [1])

# Задаём параметры изображения
color = 'gold'
transp = 1.0
material = 'plastic'

LayerParams = color, material, transp

# Задаём начальные координаты изображения специальным
# классом
rb = pXY(0, 0)

# Описываем изображения элемента body с тремя
# параметрами
img = RECTD (body, [rb, 2, 4], LayerParams)

# Описываем интервал переменной displ_x
rng = Range (displ_x, -10, 10)

# Описываем метод интегрирования
slv = SHTERM ([rng], end = 2.0, outvar = 3)

# Описываем печать результатов
prn = DISP ([rng], frm = 1)

# Команда на генерацию задания на PSL
Run()

except af.LVPS_TException,e:
    # Если произошла ошибка, печатаем её
    print e.GetError()

```

Вот что получится после генерации задания на PSL из этого примера:

```
$ DATA:

$ FRAGMENT:
#BASE: 11
#STRUCTURE:

{Вот две модели, которые мы описывали}
Model_1 'MD(8,9,10;1,1)
Model_2 'STABL0(8,11;1e+06,0,0,2,10)

#EXTERNAL:

#OUTPUT:

{Вот ПРВП, которую мы описывали}
OutVariable_1
'SUM(8,8',8",W:Model_1(1),I:Model_1(1),S:Model_1(1);1)

#MAP
$ SHOW:

{Вот изображение}
Image_1 'LAYER(Model_1(RECTD;0,0,2,4); 1,7,0,0, 0,0,0,
0,0,0,14)

$ RUN:

{Описание метода интегрирования}
Solver_1 'SHTERM(END=2,OUTVAR=3;OutVariable_1 = (-10,10))

$ PRINT:

{Описание задания на печать}
Print_1 'DISP( FROM=1;OutVariable_1 = (-10,10))

$ END
```

9. ТЕСТОВЫЕ ЗАДАНИЯ НА ЯЗЫКЕ *PPL*

После установки Прадиса в каталоге DINAMA/test/PPL вы сможете найти тестовые задания на PPL. Они уже упоминались в этом документе выше. В этом разделе мы дадим краткое описание каждому из тех заданий.

BAL.PY

Задание моделирует движение четырёх балок, описываемых разными моделями. Стоит обратить внимание, что функция “Base” вызывается несколько раз в разных местах. Это преимущество языка PPL по сравнению с PSL (там базовые узлы могут объявляться только в одном месте). Так же следует заметить, что изображение явно описано только для одной балки, в то время как рисуются все четыре. Такое возможно благодаря новым возможностям PPL, он формирует все возможные графические образы по умолчанию автоматически.

BELT.PY

Задание моделирует движение нити. Активно используется специальный класс “Diagram” при передаче параметров. Так же при описании метода интегрирования первым параметром передаётся пустой список интервалов. Обратите внимание, что список может быть и пустым, но он должен быть, и обязательно первым.

KARDANI.PY

Задание моделирует движение кардана. Стоит обратить внимание на использование метода “Cory” для обозначения эквивалентных степеней свободы в разных узлах.

KN3EF.PY

Задание моделирует соударения сферы с плоскостью. Обратите особое внимание на использование команды “import” для передачи большого количества параметров в модель. А так же на использование специального класса ContactForceModel.

KN3FF.PY

Задание моделирует соударения двух поверхностей. Особое внимание следует обратить на способ считывания данных из дополнительных файлов при помощи функции “GetXYZ”.

PSV3KT.PY

Сложное задание с большим количеством параметров, узлов и моделей. Очень активно используются специальные классы и метод “Cory” для обозначения эквивалентных степеней свободы. Так же при описании ПРВП используется метод “I” для обозначения потоковых переменных. Особое внимание следует обратить на то, что при описании изображения “OPORAD” первым параметром передаётся “None”, что означает, что образ не связан с моделью, т.е. неподвижен. Кроме того, во всех описаниях изображений последним параметром передаётся пустой список (список параметров изображения). В таких случаях используются параметры изображения по умолчанию.

ROT3_2.PY

Задание показывает работу модели ROT3. Содержит много описаний различных ПРВП.

SWING.PY

Задание с хорошей наглядной картинкой.

TATABN3.PY

Задание показывает работу модели STRGN. При описании ПРВП используется метод “W” для обозначения компонент рабочего вектора.

TCYL_M.PY

Задание показывает работу модели CYLDR.

TROT6.PY

Задание показывает работу модели ROT1.

10. PPL vs. PSL

В этом разделе показывается, что PPL по уровню интерфейса и гибкости на порядок выше, чем PSL. Мы покажем это на примере тестового задания DINAMA/test/PPL/condit.py. То, что PPL основан на Питоне, даёт огромные преимущества по сравнению с PSL. В полной мере это видно в задании condit.py. Вкратце опишем его.

Задание моделирует движение системы из грузов соединённых пружинами. Казалось бы, ничего сложного, однако есть некоторые особенности. Пружины не совсем обычные, а с характеристикой усилие от деформации заданной таблично. Обратите внимание на то, как задаётся таблица. Она задаётся в цикле:

```
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)
```

Как видите, задаётся не линейная, а квадратичная зависимость. Таким образом, можно легко задавать любые табличные зависимости, причём с любой точностью, без особых усилий. В цикле можно сделать 10 точек, а можно 1000, программа не становится сложнее и больше. Чего не скажешь о PSL. Так же обратите внимание, что точки могут добавляться не в порядке возрастания аргумента, а в произвольном – сортировка их происходит автоматически.

Ещё одна важная особенность программы в том, что даже топология моделируемой системы может меняться при смене одного символа. В программе есть флаг “FLAG”. При значении 4 описывается система из четырёх грузов, при любом другом – из трёх. Это возможно благодаря оператору Питона “if”:

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

Проследите внимательно, как это делается несколько раз в программе.

Таким образом, открываются большие возможности в описании систем благодаря тому, что PPL основан на языке программирования Python.