

PRADIS

РАБОТА С DAT ФАЙЛОМ

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

ВЕРСИЯ 4.2

СОДЕРЖАНИЕ

1. Описание DAT-файла	3
2. Структура DAT файла.....	4
3. Работа с читателем DAT файла.....	7

1. Описание DAT-файла

DAT файл создается в процессе работы решателя PRADIS и содержит информацию из внутреннего массива A решателя PRADIS. Имя файла состоит из имени файла задания плюс расширение “.DAT”. Кроме DAT файла, для корректной работы Постпроцессора, в процессе работы решателя создаются еще два дополнительных файла, с расширениями “.DIS” и “.PNM”, содержащие дополнительную информацию о выходных переменных ПРБП. Например, если мы запустим на выполнение расчет задания с именем файла SWING командой “SLANG SWING”, то в результате расчета будут созданы три файла с именами:

SWING.DAT
SWING.DIS
SWING.PNM

Режим выдачи DAT файла задается параметром **OUTVAR**, который задается в операторе RUN в списке параметров программы интегрирования. В текущей версии решателя параметр OUTVAR может принимать следующие значения:

OUTVAR=0 - не создавать DAT файл;
OUTVAR=1 - выводить только обязательные параметры (режим по умолчанию)
OUTVAR=3 - выводить обязательные параметры плюс глобальный якобиан;
OUTVAR=4 – выдавать только информацию из ПРБП. В таком режиме DAT файл получается значительно меньше по размеру, но в нем отсутствует возможность работать с трехмерной графикой.

Если параметр **OUTVAR** отсутствует в списке, то по умолчанию принимается значение **OUTVAR=1**.

Период записи информации в DAT файл задается параметром OUT, задающим интервал времени между записями в файл.

Например, если в задании записано следующее:

\$ RUN :

Расчет пружинного маятника 'NEWMARK(END=3, SCALE=1,
OUTVAR=3, OUT=0.1;

Перемещение т.А по оси X = (-1.7,0),
Скорость т.А по оси X = (-2.6, 2.6),
Ускорение т.А по оси X = (-12.7, 12.7))

то в DAT файл будут записываться обязательные параметры и глобальный якобиан с периодом в 0.1 секунды.

2. Структура DAT файла

Файл состоит из блока постоянных параметров и блока переменных параметров для каждого шага работы решателя. Вывод происходит в бинарном виде. Переменные делятся на постоянные (по времени интегрирования) и переменные (по времени интегрирования). Поскольку переменная не обязательно может быть в файле, то необходимо знать какие переменные есть и в каком порядке они идут. Для этого каждой переменной назначается свой код, который определяет тип переменной.

Структура файла следующая:

Стартовый блок:

- Версия формата, integer4
- Дата создания, integer4
- Описание задания, char(1024)
- Размер таблицы кодов постоянных параметров, integer4
- Размер таблицы кодов переменных параметров, integer4

Таблица кодов постоянных параметров:

1. Код 1
2. ...
3. Код N

Таблица кодов переменных параметров:

4. Код 1
5. ...
6. Код N

Блок постоянных параметров:

7. Значение 1
8. ...
9. Значение N

Блок переменных параметров слой 1:

10. Значение 1
11. ...
12. Значение N

...

Блок переменных параметров слой T:

13. Значение 1
14. ...
15. Значение N

В начале значения каждого постоянного и переменного параметра записывается его длина в формате INTEGER*4, а потом массив соответствующей длины переменных в формате REAL*8 или INTEGER*4.

Файл содержит следующие массивы постоянных параметров с соответствующими кодами, приведенными в начале каждой строки:

- 1 - Вектор параметров программы интегрирования (REAL*8)
- 2 - Вектор параметров PAR (REAL*8)
- 3 - Вектор смещений для ПГО (INTEGER*4)

- 4 - Вектор смещений для моделей (INTEGER*4)
- 5 - Вектор стыковки элементов системы по I (INTEGER*4)
- 6 - Вектор смещений для программ расчета выходных парам. (INTEGER*4)
- 7 - Вектор-указатель системных параметров (INTEGER*4)
- 8 – Вектор структуры модели (INTEGER*4)
- 9 – Вектор структуры ПГО (INTEGER*4)

Файл содержит следующие массивы переменных параметров с соответствующими кодами, приведенными в начале каждой строки:

Обязательные параметры:

- 21 - Вектор X, V, A (REAL*8)
- 22 - Вектор нового состояния (REAL*8)
- 23 - Вектор старого состояния (REAL*8)
- 24 - Рабочие вектора WRK (REAL*8)
- 25 - Выходные значения ПРВП (REAL*8)
- 26 - Данные для ПГО (REAL*8)
- 27 - Токи моделей локальные (вектор I) (REAL*8)
- 28 - Непоименованная COMMON-область (смешанный формат, соответствующий формату непоименованной области COMMON решателя. Порядок и тип переменных описан в разделе 2 этого документа в описании структуры блока COMMON).
- 29 - Вектор COUNT (INTEGER*4)

Необязательные параметры:

- 41 - Глобальный якобиан

Вектор структуры моделей имеет следующую структуру:

- 1. Длина всей структуры (INTEGER*4)
- 2. Количество моделей (INTEGER*4)
- 3. Имя модели (CHARACTER*8)
- 4. ID модели (INTEGER*4)
- 5. Количество узлов модели (INTEGER*4)
- 6. Список номеров узлов модели (INTEGER*4)

Этот вектор является списком моделей, содержащим вышеописанную структуру для каждой модели.

Вектор структуры ПГО имеет следующую структуру:

- 1. Количество ПГО
- 2. Номер модели соответствующей данной ПГО (если = 0, то без модели) (INTEGER*4)
- 3. Имя ПГО (CHARACTER*8)
- 4. Количество параметров PARIMD (INTEGER*4)
- 5. Начало параметров PARIMD в массиве A (INTEGER*4)
- 6. Количество параметров WRKIMD (INTEGER*4)
- 7. Начало параметров WRKIMD в массиве A (INTEGER*4)
- 8. Количество параметров PARLR2 (INTEGER*4)
- 9. Начало параметров PARLR2 в массиве A (INTEGER*4)
- 10. ID ПГО (INTEGER*4)
- 11. Значение параметра UNV (INTEGER*4)

Этот вектор является списком ПГО, содержащим вышеописанную структуру для каждой ПГО.

Вектор параметров программы интегрирования (код параметра 1) состоит из следующих параметров: 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

3. Работа с читателем DAT файла

Для работы с файлом DAT разработан соответствующий пакет классов контекстов и структур. Пакет состоит из следующих классов и структур:

SolverContext – основной класс, управляющий чтением файла, позиционированием на нужный временной срез (Layer) файла. От него инициализируются все остальные классы. Объект данного класса позволяет осуществлять навигацию по временным слоям файла и получать постоянные и переменные величины. Для навигации используется метод `SetLayer(int)`, который устанавливает контекст на соответствующий слой. Для получения значений постоянных параметров используются методы с соответствующими именами, а для получения переменных параметров используются методы `GetVariableInt(int)` и `GetVariableDouble(int)`. Входным параметром этих методов является код переменной, значения которых приведены выше в списке параметров.

ModelContext – контекст моделей. Предназначен для получения информации касающейся моделей. Объект `ModelContext` содержит метод `SetModelNumber(int)` позволяющий осуществлять навигацию по моделям. Входным параметром этого метода является порядковый номер модели, который находится в пределах от 1 до N, где N равно количеству моделей и получается из метода `int GetModelSize()`.

DOFContext - контекст переменных типа X, V, A. Предназначен для получения значений X, V, A выбранного узла на текущем временном срезе.

PGOContext - контекст ПГО. Предназначен для получения информации касающейся ПГО. Объект `PGOContext` содержит метод `SetPGONumber(int)` позволяющий осуществлять навигацию по ПГО. Входным параметром этого метода является порядковый номер ПГО, который находится в пределах от 1 до N, где N равно количеству ПГО и получается из метода `int GetPGOSize()`.

Так как `PGOContext` содержит номер модели, к которой данная ПГО относится, то используя этот номер можно создать объект `ModelContext` из которого можно получить всю информацию, касающуюся модели с данным номером, в том числе и список узлов. Если номер модели равен нулю, то это означает, что нет соответствующей модели и соответственно нет степеней свободы и это образ статический.

Структура для блока 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;
};
```

Структура для DOF:

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

```
};
```

SolverContext – содержит следующие методы:

int GetVersion();	Получить версию файла
int GetCreationDate();	Получить дату создания файла
char* GetDescription();	Получить описание файла
int GetPermanentCodeTableLength();	Получить длину таблицы постоянных параметров
int GetVariableCodeTableLength();	Получить длину таблицы переменных параметров
int* GetPermanentCodeTable();	Получить таблицу постоянных параметров
int* GetVariableCodeTable();	Получить таблицу переменных параметров
int GetIntegrationParamLength();	Получить длину вектора параметров интеграции
double* GetIntegrationParam();	Получить вектор параметров интеграции
int GetParLength();	Получить длину вектора параметров PAR
double* GetPar();	Получить вектор параметров PAR
int GetASMILength();	Получить длину вектора ASMI
int* GetASMI();	Получить вектор ASMI
int GetASMMLength();	Получить длину вектора ASMM
int* GetASMM();	Получить вектор ASMM
int GetANUZLLength();	Получить длину вектора ANUZL
int* GetANUZL();	Получить вектор ANUZL
int GetASMOLength();	Получить длину вектора ASMO
int* GetASMO();	Получить вектор ASMO
int GetASYSOLength();	Получить длину вектора ASYSO
int* GetASYSO();	Получить вектор ASYSO
int GetModelStructureLength();	Размер вектора структуры моделей
int * GetModelStructure();	вектора структуры моделей
int GetModelSize();	Количество моделей
char * GetModelNames();	Имена моделей
int GetPRVPSize();	Количество ПРВП
int GetPGOSize();	Количество ПГО
char * GetPGONames();	Имена ПГО

int * GetPGOStructure();	Вектор структуры ПГО
int GetLayerSize();	Получить количество слоев
int Refresh();	Обновление состояния
int SetLayer(int);	Установка текущего слоя
int* GetVariableInt(int);	Получить целочисленные параметры
double* GetVariableDouble(int);	Получить реальные параметры
int GetVariableLength(int);	Получить длину параметра
COMMON GetCOMMON();	Получить COMMON область
double GetTime();	Получить время текущего слоя
int GetCurrLayer();	Получить текущий слой

ModelContext – содержит следующие методы:

SetLayer(int Layer);	Установить слой
int SetModelNumber(int);	Установить номер текущей модели
char * GetModelName();	Получить имя текущей модели
int GetModelCode();	Получить код в базе данных текущей модели
int GetModelNodeSize();	Получить количество степеней свободы текущей модели
int * GetModelNodes();	Получить номера степеней свободы текущей модели
double * GetLocal_I();	Получить токи моделей локальные (вектор I)
int GetParLength();	Получить количество параметров модели (вектор PAR)
double * GetPar();	Получить параметры модели (вектор PAR)
int GetWRKLength();	Получить длину WRK массива
int GetNewLength();	Получить длину New массива
int GetOldLength();	Получить длину Old массива
double * GetWRK();	Получить массив WRK
double * GetNew();	Получить массив New
double * GetOld();	Получить массив Old
int GetModelSize();	Получить количество моделей

DOFContext – содержит следующие методы:

SetLayer(int);	Установить слой
----------------	-----------------

int GetLength();	Получить количество узлов
struct DOF GetDOF(int n);	Получить X,V,A для узла n

PGOContext – содержит следующие методы:

SetLayer(int Layer);	Установить слой
int SetPGONumber(int);	Установить номер текущей ПГО
int GetPGOSize();	Получить количество ПГО
char * GetPGOName();	Получить имя текущей ПГО
int GetModelNumber();	Получить номер модели соответствующей текущей ПГО (если номер=0, то никакой модели не принадлежит)
int GetPGOCode();	Получить код в базе данных текущей ПГО
int GetUNV();	Получить параметр UNV
int GetPARIMDLength();	Получить количество параметров PARIMD
double * GetPARIMD();	Получить параметры PARIMD
int GetWRKIMDLength();	Получить количество параметров WRKIMD
double * GetWRKIMD();	Получить параметры WRKIMD
int GetPARLR2Length();	Получить количество параметров PARLR2
double * GetPARLR2();	Получить параметры PARLR2
int GetColor();	Получить цвет

Ниже приведен текст программы, иллюстрирующей работу со всеми вышеперечисленными методами классов:

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

// ----- Стартовый блок -----
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;

// ----- Постоянные параметры -----
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;

    // ----- Переменные параметры -----
    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;

```