SODD : Another MAD-X module

E. T. d’Amico

Abstract

The new madx command sodd implements the functionalities of the stand-alone program SODD (Second Order Detuning and Distortion function) written by F. Schmidt. It provides an easy interface to other MAD-X modules by automatically computing the required input parameters from the last TWISS command. The results are not only stored into files as it is done in the stand alone program but also inside internal MAD-X tables for the use in subsequent MAD-X commands. This note describes the coding implementation and documents the use of the command sodd.
1 Introduction

The stand-alone program SODD [1] has proved to be extremely useful. It calculates the detuning and distortion functions according to the analytical formulae of [2] extended to the second order by Frank Schmidt [3]. It consists of three parts:

Subroutine `detune` calculates the detuning terms in first and second order in the strength of the multipoles.

Subroutine `distort1` calculates the distortion function and the Hamiltonian terms in first order in the strength of the multipoles.

Subroutine `distort2` calculates the distortion function and Hamiltonian terms in second order in the strength of all combinations of the multipoles.

The interface with MAD-X was done through ASCII files which was not very convenient especially when the results were to be used by other MAD-X modules. Thus it was decided to create a new MAD-X module called `sodd` which will include inside MAD-X all the stand-alone functionalities. The input parameters become attributes of the `sodd` command and the data about the multipoles can be generated automatically taking advantage of the information provided by the previous MAD-X command TWISS. Actually an internal call to the program `sixtrack` is executed to build up the input file `fc.34`. The result files remain the same, albeit with more meaningful names and with an added header to label each column. Corresponding internal MAD-X tables are generated with the same names but without the first header line. The result data (all in double precision) can be accessed in a Fortran program by calls to the integer function `double_from_table` as in the following example:

```fortran
k = double_from_table('detune_1_end', 'mpor', n, multipole_order)
```

where the value at the crossing of the nth row and the column with name `mpor` in the table named `detune_1_end` is transferred into the variable `multipole_order`. The integer k is 0 for no error, -1 if the table does not exist, -2 if the column does not exist, -3 if the row does not exist. Reminder: each character string should end with a blank character.

The result data can also be accessed in a C program by calls to the function `double_from_table` as in the following example:

```c
int k,n;
double multipole_order;
....
k = double_from_table('detune_1_end', 'mpor', &n,&multipole_order);
....
```

On return, k has the same value as in the Fortran example.

The main points about the code implementation are given in the next section. Details about the input attributes and the description of the result files and internal tables can be found in the following sections **Input** and **Results**.
2 Code implementation

MAD-X offers a very large number of functionalities which are provided by software modules written partly in C and partly in Fortran. The modular structure is very flexible and an extended set of utilities facilitates the insertion of new modules. The module sodd is the first case study for such an insertion and it seems worthwhile to describe how the actual code implementation has been carried out. Adding a new module should start by informing MAD-X about the new command name and its related attributes. These are readily inserted in the file madxdict.h, its structure being self-explanatory. The following code has been added for sodd:

```
"sodd: control none 0 0 
"detune = [l, false, true], "
"distort1 = [l, false, true], "
"distort2 = [l, false, true], "
"start_stop = [r, {0.0,0.0}], "
"noprnt = [l, false, true], "
"nosixtrack = [l, false, true], "
"print_at_end = [l, false, true], "
"print_all = [l, false, true], "
"multipole_order_range = [i,{1,2}]; 
""
```

Next one needs to provide the required link in the main C program madxn.c by adding the line containing the new command name

```
extif (strcmp(toks[k], "sodd") == 0) exec_sodd(cmd);
```

to the routine control which evaluates the input stream and looks for known command names. It remains to write the new routine exec_sodd (also part of the main C program madxn.c) which can be done by referring to similar routines i.e. exec_plot. Attributes can be read either inside a C routine by using C-structures as in the following example for the logical attribute nosixtrack:

```
struct name_list* nl_sodd;
int pos,nosixtrack;
....
if (this_cmd != NULL & this_cmd->clone != NULL)
{
    nl_sodd = this_cmd->clone->par_names;
}
pos = name_list_pos("nosixtrack", nl_sodd);
nosixtrack = nl_sodd->inform[pos];
....
```

or inside a Fortran program by calls to the MAD-X routine comm_para an example of which is shown to acquire the attribute detune:

```
call comm_para('detune ', nint, ndble, k, int_arr, d_arr, char_a, char_l)
```

Since the program sodd was already written in Fortran, the most natural choice for the MAD-X module integration was to transform it in a Fortran routine named soddin. The read(5,*) commands where replaced by calls to comm_para. Of course the instruction

```
soddin_(&ierr);
```
had to be added to the C routine exec_sodd. Finally the declaration file madxd.h had to be updated with the following lines:

....
extern void soddin_(int*);
....
void exec_sodd(struct in_cmd*);
....

All internal tables used within the module Sodd have to be defined in the declaration file madxd.h by adding the following lines:

....
struct table* sodd_table_70; /* sodd output table detune_1_end */
struct table* sodd_table_71; /* sodd output table detune_1_all */
struct table* sodd_table_72; /* sodd output table detune_2_end */
struct table* sodd_table_73; /* sodd output table detune_2_all */
struct table* sodd_table_74; /* sodd output table distort_1_F_end */
struct table* sodd_table_75; /* sodd output table distort_1_H_end */
struct table* sodd_table_76; /* sodd output table distort_1_F_all */
struct table* sodd_table_77; /* sodd output table distort_1_H_all */
struct table* sodd_table_78; /* sodd output table distort_2_F_end */
struct table* sodd_table_79; /* sodd output table distort_2_F_all */
....

and the column names have to be defined in the declaration file madxl.h, to which the following lines have been added:

....
int sodd_detune_5_types[] = 
{
    1, 1, 2, 1, 1
};

char* sodd_detune_5_cols[] =
{
    "mpor", "plane/mpor2", "detune", "H_inv_order", "V_inv_order",
    " " /* blank terminates */
};

int sodd_distort1_8_types[] = 
{
    2, 2, 2, 2, 2, 2, 2, 2
};

char* sodd_distort1_8_cols[] =
{
    "mpor", "cos", "sin", "amp", "j", "k", "l", "m",
    " " /* blank terminates */
};

int sodd_distort1_11_types[] =
The actual creation of the tables is realised by the following code added to the routine exec_sodd inside the main C program madxn.c:

```c
sodd_table_70 = make_table("detune_1_end", "sodd_detune_5",
    sodd_detune_5_cols, sodd_detune_5_types, 2);
sodd_table_70->dynamic = 1;
add_to_table_list(sodd_table_70, table_register);
sodd_table_71 = make_table("detune_1_all", "sodd_detune_5",
    sodd_detune_5_cols, sodd_detune_5_types, 2);
sodd_table_71->dynamic = 1;
add_to_table_list(sodd_table_71, table_register);
sodd_table_72 = make_table("detune_2_end", "sodd_detune_5",
    sodd_detune_5_cols, sodd_detune_5_types, 2);
sodd_table_72->dynamic = 1;
add_to_table_list(sodd_table_72, table_register);
sodd_table_73 = make_table("detune_2_all", "sodd_detune_5",
    sodd_detune_5_cols, sodd_detune_5_types, 2);
sodd_table_73->dynamic = 1;
add_to_table_list(sodd_table_73, table_register);
sodd_table_74 = make_table("distort_1_F_end", "sodd_distort1_8",
    sodd_distort1_8_cols, sodd_distort1_8_types, 2);
sodd_table_74->dynamic = 1;
add_to_table_list(sodd_table_74, table_register);
sodd_table_75 = make_table("distort_1_H_end", "sodd_distort1_8",
    sodd_distort1_8_cols, sodd_distort1_8_types, 2);
sodd_table_75->dynamic = 1;
add_to_table_list(sodd_table_75, table_register);
sodd_table_76 = make_table("distort_1_F_all", "sodd_distort1_11",
    sodd_distort1_11_cols, sodd_distort1_11_types, 2);
```

...
sodd_table_76->dynamic = 1;
add_to_table_list(sodd_table_76, table_register);
sodd_table_77 = make_table("distort_1_H_all", "sodd_distort1_11",
    sodd_distort1_11_cols, sodd_distort1_11_types, 2);
sodd_table_77->dynamic = 1;
add_to_table_list(sodd_table_77, table_register);
sodd_table_78 = make_table("distort_2_F_end", "sodd_distort2_9",
    sodd_distort2_9_cols, sodd_distort2_9_types, 2);
sodd_table_78->dynamic = 1;
add_to_table_list(sodd_table_78, table_register);
sodd_table_79 = make_table("distort_2_H_end", "sodd_distort2_9",
    sodd_distort2_9_cols, sodd_distort2_9_types, 2);
sodd_table_79->dynamic = 1;
add_to_table_list(sodd_table_79, table_register);

Filling the tables is done inside the Fortran routine soddin with calls to the routine double_to_table as in the following example:

    call double_to_table(table_name,name_5(k),double_to_write(k))

where table_name is the name of the table as defined in the first argument of the routine make_table, name_5(k) is the column name and double_to_write(k) is the double precision value to be stored.

3 Input data

The attributes which can be inserted in the sodd command are:

- detune : logical, the default value being false. If true, the detune subroutine is executed.
- distort1 : logical, the default value being false. If true, the distort1 subroutine is executed.
- distort2 : logical, the default value being false. If true, the distort2 subroutine is executed.
- start_stop : position range given as two real numbers separated by a comma. Only the multipoles located within this range will be considered in all calculations. Unit is m.
- multipole_order_range : specifies the low and high limit of orders to be studied. Erect and skew elements are denoted with positive and negative values respectively. The two integer values should be separated by a comma.
- noprint : logical, the default value being false. If true, no file or internal table will be created to keep the results. In this case the attributes print_all or print_at_end have no effect.
- print_all : logical, the default value being false. If true, the files and internal tables containing results at each multipole will be generated.
- print_at_end : logical, the default value being false. If true, the files and internal tables containing results at the end of the position range will be generated.
- nosixtrack : logical, the default value being false. If true, the input file fc.34 called fort.34 in [1] will not be generated internally by invoking the conversion routine of sixtrack and the user must provide it before the execution of the sodd command.

4 Results

4.1 attribute detune

If the attribute print_at_end has been set, the following two files (and the corresponding internal tables) are created at the end of the position range:
The corresponding column names in the associated internal table are:

1) mpor
2) plane/mpor2
3) detune
4) H_inv_order
5) V_inv_order

If the attribute print_all has been set, the following two files (and the corresponding internal tables) are created at each multipole in the position range:

\textit{detune.1.all} containing five columns which are the same as for the internal table \textit{detune.1.end}.

The corresponding column names in the associated internal table are the same as for the internal table \textit{detune.1.end}.

\textit{detune.2.all} containing five columns which are the same as for the internal table \textit{detune.2.end}.

The corresponding column names in the associated internal table are the same as for the internal table \textit{detune.1.end}.

\textbf{4.2 attribute distort1}

If the attribute print_at_end has been set, the following two files (and the corresponding internal tables) are created at the end of the position range:

\textit{distort.1.F.end} containing eight columns:

1) multipole order
2) cosine part of distortion
3) sine part of distortion
4) amplitude of distortion
5) j
6) k
7) l
8) m
The corresponding column names in the associated internal table are:
1) mpor
2) cos
3) sin
4) amp
5) j
6) k
7) l
8) m

`distort_1_H_end` containing eight columns:
1) multipole order
2) cosine part of Hamiltonian
3) sine part of Hamiltonian
4) amplitude of Hamiltonian
5) j
6) k
7) l
8) m

The corresponding column names in the associated internal table are the same as for the internal table `distort_1_H_end`.

If the attribute `print_all` has been set, the following two files (and the corresponding internal tables) are created at each multipole in the position range:

`distort_1_F_all` containing eleven columns:
1) multipole order
2) appearance number in position range
3) number of resonance
4) position
5) cosine part of distortion
6) sine part of distortion
7) amplitude of distortion
8) j
9) k
10) l
11) m

The corresponding column names in the associated internal table are:
1) mpor
2) app
3) res
4) pos
5) cos
6) sin
7) amp
8) j
9) k
10) l
11) m
\textit{distort1 H}\textsubscript{all} containing eleven columns:

1) multipole order
2) appearance number in position range
3) number of resonance
4) position
5) cosine part of Hamiltonian
6) sine part of Hamiltonian
7) amplitude of Hamiltonian
8) j
9) k
10) l
11) m

The corresponding column names in the associated internal table are the same as for the internal table \textit{distort1 F}\textsubscript{all}.

4.3 \textbf{attribute distort2}

Only the attribute \textit{print_at_end} is taken in account. If present, the following two files (and the corresponding internal tables) are created at the end of the position range:

\textit{distort2 F}\textsubscript{end} containing nine columns:

1) first multipole order
2) second multipole order
3) cosine part of distortion
4) sine part of distortion
5) amplitude of distortion
6) j
7) k
8) l
9) m

The corresponding column names in the associated internal table are:

1) mpor
2) mpor2
3) cos
4) sin
5) amp
6) j
7) k
8) l
9) m
containing nine columns:
1) first multipole order
2) second multipole order
3) cosine part of Hamiltonian
4) sine part of Hamiltonian
5) amplitude of Hamiltonian
6) \( j \)
7) \( k \)
8) \( l \)
9) \( m \)
The corresponding column names in the associated internal table are the same as for the internal table distort.2

5 Acknowledgments

I would like to thank O. Bruning and F. Schmidt for their support and encouragement to write this note. Their comments have greatly improved it.

References