4-DIMENSIONAL TABLE INTERPOLATION WITH MODELICA

Tobias Hirsch Markus Eck
German Aerospace Center (DLR)
Pfaffenwaldring 38-40, 70569 Stuttgart, Germany
tobias.hirsch@dlr.de, markus.eck@dlr.de

Abstract

The steady-state model for a solar field contains a large number of equations including conditional statements. For a yearly energy yield analysis the operational state (on duty, off duty) of the solar field may change from one time instant to the other. Due to the strongly varying boundary conditions a simulation run without convergence problems is not likely. For this reason a lookup-table model is designed to calculate the five output variables of the solar field depending on the four input variables. The interpolation model is based on the existing MODELICA model for 2D-interpolation and can be used for table interpolation tasks independent of the technical application. The structure of the model and a method for the automatic generation of the required interpolation data from the complex solar field model is described.

Keywords: solar power plant; look-up table; interpolation

1 Introduction

Solar thermal power plants are one of the most interesting options for renewable electricity production. For the calculation of the annual energy yield of these plants steady-state models are used. The calculation method which is based on mass and energy balances is called for every hour of the year with the corresponding weather data input and delivers an output of electric energy. This approach works well as long as transient effects in the plant can be neglected. When a thermal storage has to be considered an additional transient model has to be implemented. Since the solar field and the power block can still be represented as a steady-state block, the final plant model is composed of very complex steady-state models for the solar field and the power block and a rather simple transient model of the storage system. For an annual calculation on an hourly basis, the model is called 8760 times with input data that might be strongly varying from hour to hour. First tests with the complex steady-state models show that robustness of the simulation is not satisfying. Due to the large changes in input parameters and model dependencies it is very likely that an annual calculation might terminate before reaching the end time.

The reason for the complexity of the solar field model is the aspect that the model has to describe the operation in full load, part load and stand-by mode. While mass and energy balances are derived for regular field operation this is not the case for the stand-by mode. In order to determine the time instant with irradiation conditions sufficient for a switch from stand-by into part-load operation the set of balance equations has to be solved with a modified set of input parameters even if the field is shut-down. Implementing the equations within the MODELICA language yields a number of conditional statements that have to be operated by the solver. Robustness of the resulting system is hard to check and may differ from one field layout to the other.

A way to couple the complex steady-state field model with the simple transient thermal storage model is developed by replacing the equation-based solar field model by a table-based interpolation. When analyzing the system it is found that the solar field output is determined by just four independent inputs. Unfortunately, the existing interpolation model in MODELICA is limited to two independent variables. Within this paper, a MODELICA model is presented that allows a three dimensional interpolation using the MODELICA 2D-interpolation model. By an additional interpolation level the capability can easily be extended to an interpolation in four dimensions.

2 Solar field model characteristics

The solar field is composed of a large number of parabolic trough collector rows arranged in parallel. The water fed into the field at high pressure is preheated, evaporated and superheated by the solar irra-

diation. This kind of system is called a Direct Steam Generation parabolic trough power plant [1]. Apart from general parameters of the field, the output of the solar field is determined by the following input variables:

- Direct normal irradiation, *DNI*
- Ambient temperature, *T_amb*
- Feed water specific enthalpy, *h_in*
- Operating pressure of the field, *p_out*

All of these are a function of time with the first two taken from the weather data file and the last two being determined by the whole plant model. In addition to the generated mass flow, four more outputs have to be provided by the model, so the list of output variables reads:

- Steam mass flow, *m_out*
- Field inlet pressure, *p_in*
- Field outlet temperature, *T_out*
- Recirculation pump power, *P_rec*
- "Field in operation"-indicator, *FIO*

A MODELICA solar field model is available that describes the relation between input- and output parameters based on the physical equations. The model allows changes in the solar field configuration in an easy way by simply changing some parameters that e.g. determine the number or arrangement of collector rows. It is therefore suited for the design of a solar field but is not suited for annual energy yield analysis.

3 General approach

The physically based solar field model is replaced by a table interpolation model that calculates one output variable (e.g. *m_out*) based on a set of interpolation data and the three input variables (*h_in, p_out, DNI*). Extension to the forth input variable is done by linear interpolation in the ambient temperature (*T_amb*). For each of the five output variables the same interpolation model can be used with an individual set of interpolation data. The interpolation data are automatically generated by calling the physical solar field model from a MATLAB script for all nodes of the interpolation data. The outputs of the solar field are stored in MATLAB .mat files and can directly be read by the MOCELICA interpolation model. Within the following sections the automatic generation of the interpolation data and the structure of the interpolation model will be described.

4 Generation of interpolation data

Since a large number of solar field configurations, each described by one set of interpolation data, is to be analysed for the yearly output, an efficient method is needed to generate the interpolation data. For the interpolation routines in MODELICA one look-up table in three dimensions (variation of input variables *p_out, h_in, DNI*) has to be provided for each of the five output variables (*m_out, p_in, T_out, P_rec, FIO*).

This is realized by a MATLB script file that calls the MODELICA executable for all combinations of input variables. By use of the DYMOLA-MATLAB interface the output variables are then stored by the MATLAB script in a ".mat"-file. For each output variable a separate file is generated that stores the three vectors of parameter variations

```
p_steps =[p_start : dp_: p_end]
h_steps =[h_start : dh_: h_end] ;
I_steps =[I_start : dI_: I_end] ;
```

and the three-dimensional result matrix containing the results at the nodes defined by the vectors above. The procedure is illustrated in figure 1.

Due to the complexity of the solar field model it is initialized with a fixed set of parameters. The desired operating point for each input parameter combination

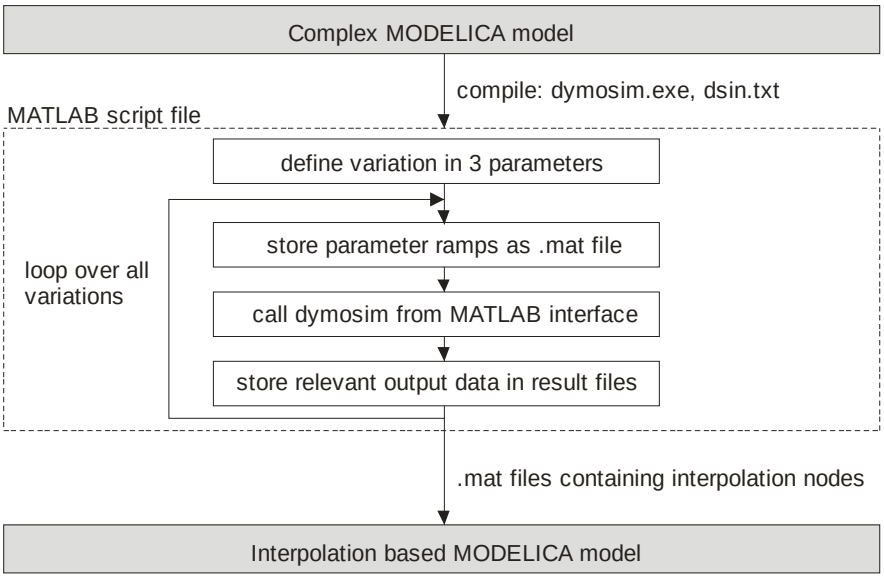

Figure 1: Procedure for generation of interpolation data

is then reached by a ramp in the three input variables. The final state of the ramp (values of the input variables for the actual combination) is stored by the MATLAB script in a .mat file before the executable is called. The data are then read by the executable to define ramps in the input variables that lead from the fixed initialization state to the desired final state. This approach has the advantage that no problems with the initialization occur during the parameter variations due to the stable initialization state. One separate call of the executable for each parameter variation is chosen, although the ramps might have been defined to generate a number of results points in one simulation run. The advantages for the implementation chosen are:

- only one data point is lost if the simulation does not converge
- high flexibility in the definition of the parameter variations (e.g. no need for equidistant grids) .

The output variable *FIO* is very important for the following interpretation of the interpolated data since it determines if a data point calculated by interpolation is valid. The value is set to false if the solar field can not be operated for the combination of input variables or if the simulation has not converged. In both cases, the data points obtained from the interpolation do not represent a physical state of the solar field.

In order to allow direct access to the interpolation data from the MODELICA 2D-interpolation model *CombiTable2D* the data a stored in the following way. For each value of input variable x_3, e.g. 70 bar, 80 bar, 90 bar, 100 bar, 110 bar, a set of 2D-interpolation data are stored in one separate matrix. In our example, these matrices are named *data1* to *data5*. The matrix contains in the first row the vector of nodes in variable x_2 and in the first column the vector of nodes in variable x_1. The matrix is then filled with the output data at the corresponding nodes:

0	$x_2(1)$	$x_2(2)$	...	$x_2(ih)$
$x_1(1)$	dat(1,1)	dat(1,2)...		dat(1,4)
$x_1(2)$	dat(2,1)	dat(2,2)...		dat(2,4)
...	...	...	...	...
$x_1(iI)$	...	...	...	dat(iI,ih)

All data matrices together are stored in one single .mat-file. This file holds all data necessary for the 3D-interpolation in variables x_1, x_2 and x_3. For each output variable that has to be described by 3D-interpolation a separate file is generated. This allows, in principle, an arbitrary number of output variables. In our example, five output variables are used with the data stored in the files *FIO.mat*, *m_flow.mat*, *p_in.mat*, *P_rec.mat*, *T_out.mat*.

5 3D interpolation model

The three-dimensional table interpolation used in the yearly analyzer is based on the two-dimensional table interpolation model available in the MODELICA standard library. This model is very efficient since the search for the interpolation interval starts at the result found in the last time instant. The two dimensional interpolation model is used to interpolate in the variables x_1 (*DNI*) and x_2 (*h_in*) for a fixed value of variable x_3 (p_out). For each value of the variable x_3 defined in the vector *p_steps* one value u_i (i=1:n) for the output variable is calculated. The final output value is then generated by a 1-D interpolation in the n results u_i. The procedure is illustrated in figure 2. The model that holds the following equations is named *Kennlinie3D* (german word for Characteristic3D). In the following, the code of this model is described. The model contains three inputs

```
Modelica.Blocks.Interfaces.RealInput x1;
Modelica.Blocks.Interfaces.RealInput x2;
Modelica.Blocks.Interfaces.RealInput x3;
```

for variables x_1, x_2 and x_3. In the solar field example these inputs correspond to *h_in*, *DNI*, *p_out*. The result is delivered via output

```
Modelica.Blocks.Interfaces.RealOutput y;
```

A data structure is defined to provide information on the upper and lower limits of x_1 and x_2 as well as the matrix name in the interpolation file that holds the interpolation data.

```
encapsulated record interpolation_source
      Real   x3;
      Real   min_x1;
      Real   max_x1;
      Real   min_x2;
      Real   max_x2;
      String table_name;
end interpolation_source;
```

In the model n instances of this data structure are created as parameters by:

```
parameter interpolation_source[:]
      IP_source;
```

In Dymola, the data can be entered via the graphical user interface which is shown in figure 3. In this example, 2-D-interpolation in x_1 and x_2 data have been generated for five pressure levels from 70 bar up to

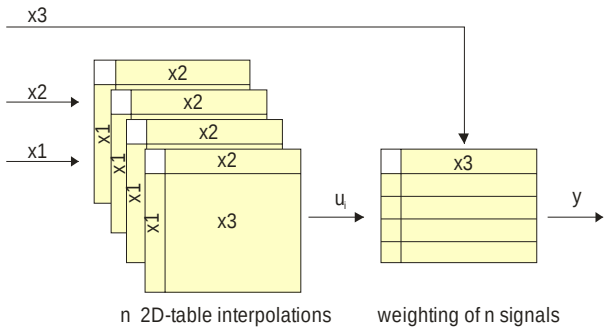

Figure 2: Structure of the 3D interpolation model

110 bar. The interpolation data are found in matrices *data1* to *data5* in the interpolation data file defined by `parameter String SourceFile= "p_in".`

The variable x_1 (*h_in*) may vary between 500 kJ/kg and 1100 kJ/kg and the variable x_2 (*DNI*) between 0 and 1000 W/m^2. The 2-dimensional interpolation is done in *n* MODELICA interpolation blocks which are instantiated by

```
Modelica.Blocks.Tables.CombiTable2D
     IP_table[n](
         each tableOnFile=true,
         each fileName=SourceFile,
         tableName={IP_source[i].table_name
                    for i in 1:n}
                );
```

The inputs x_1 and x_2 and connected to the corresponding inputs u_1 and u_2 of the *n* interpolation blocks, taking into account the variable range limitations defined in `IP_source`.

```
for i in 1:n loop
   IP_table[i].u1=
      max(IP_source[i].min_x1,
          min( IP_source[i].max_x1, x1 )
          );
   IP_table[i].u2=
      max(IP_source[i].min_x2,
          min( IP_source[i].max_x2, x2 )
          );
end for;
```

The final result is calculated by weighting the *n* outputs of the 2D-interpolation blocks

```
y = sum(  IP_table[i].y*weight[i]
          for i in 1:n );
```

The weighting factors are calculated from a linear interpolation in the variable x_3. For example, a value of x_3=82e5 Pa would lead to a vector of weighting factors *weight* =[0 0.8 0.2 0 0]. The Dymola routine *dymTableIpo1* is used for the interpolation. This routine has to be initialized by

```
when initial() then
   Weight_tableID=dymTableInit(
           1.0,
           smoothness,
           "NoName",
           "NoName",
           Weight_matrix,
           0.0);
end when;
```

and called with the command

```
for i in 1:n loop
   weight[i] =
      min(1.0,
          max(0.0,dymTableIpo1(
                  Weight_tableID,
                  Weight_columns[i],
                  x3))  );
end for;
```

with the corresponding declarations

```
parameter Real[:,:]   Weight_matrix =
    [IP_source.x3, diagonal(ones(n))];

parameter Integer     Weight_columns[:]=
    2:size(Weight_matrix, 2);

Real     Weight_tableID;
Real[n]  weight;
parameter
   Modelica.Blocks.Types.Smoothness.
   Temp   smoothness =
   Modelica.Blocks.Types.Smoothness.
        LinearSegments;
```

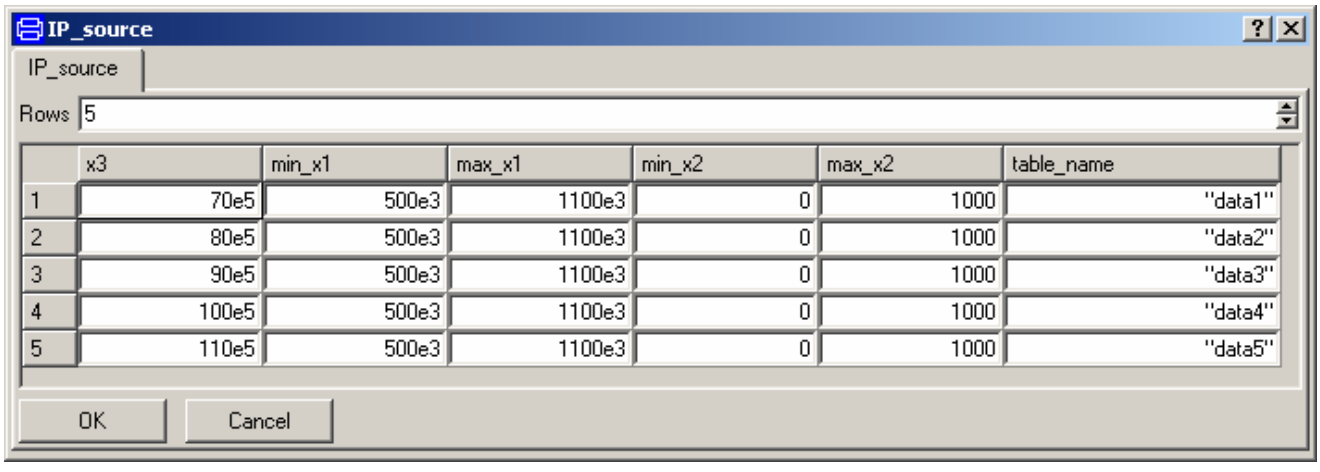

Figure 3: Screenshot of the Dymola graphical user interface for `IP_source` with five pressure levels

6 Solar field model with 3 inputs

The solar field model *SolarField_Characteristic* based on the interpolation is assembled from five 3D-interpolation blocks of type *Kennlinie3D* as shown in figure 4. The three input connectors for *h_in* (red lines), *DNI* (blue lines) and *p_out* (green lines) are connected to the corresponding inputs of the 3D-interpolation blocks. Based on the interpolation data provided in files *FIO.mat*, *m_flow.mat*, *p_in.mat*, *P_rec.mat*, *T_out.mat* the outputs *FIO*, *m_flow*, *p_in*, *P_rec* and *T_out* are calculated. The values are only valid if the indicator *FIO* is 1. In case this value is smaller than 1, a default value, e.g. 70 bar for *p_in*, is used instead of the calculated value.

7 Extension to four dimensions

As mentioned in the beginning of this text the solar field output depends on one more variable namely the ambient temperature. Since the dependence on this variable is nearly linear three nodes in ambient temperature (0 °C, 20 °C, 40 °C) are sufficient for the model. For each of the three temperature levels a separate set of interpolation data is generated. Three instances of the solar field model *SolarField_Characteristic* are created with the outputs linearly weighted with the actual ambient temperature *T_amb*. The weighting is realized by the same approach as in the 3D-interpolation model using the Dymola function *dymTableIpo1*. For reusability a model called *WeightedSignals* is defined. Figure 5 shows a screenshot of the final solar field model with the three *SolarField_Characteristic* models each representing one level of ambient temperature and five *WeightedSignals* models that are responsible for weighting obtained from the three interpolation models.

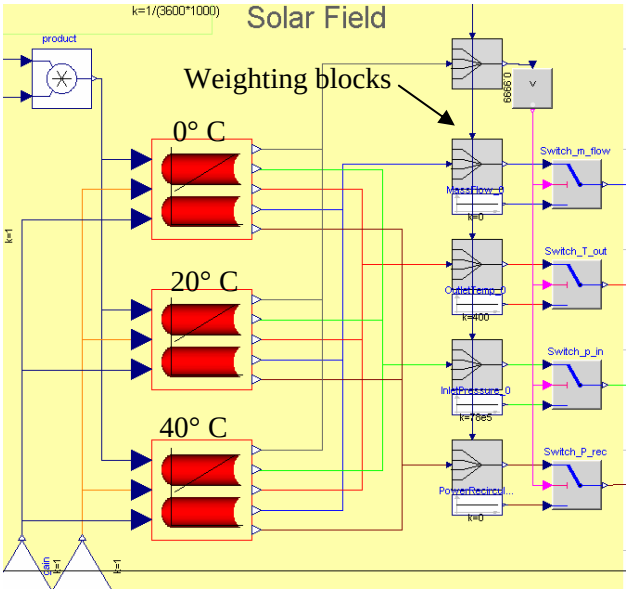

Figure 5: Solar field model with three instances of the *SolarField_Characteristic* model representing three levels of ambient temperatures

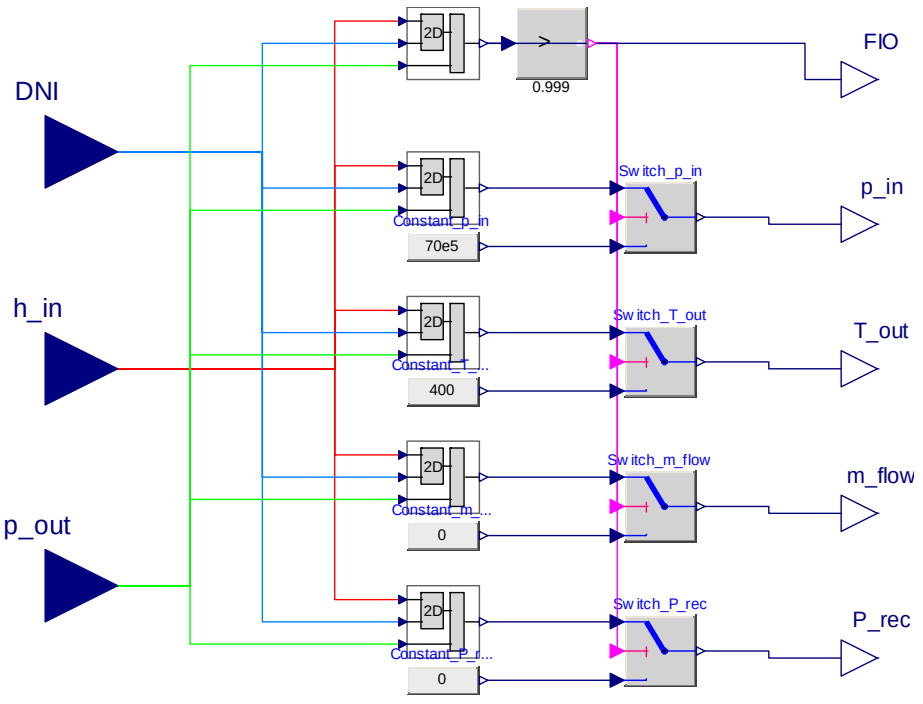

Figure 4: The *SolarField_Characteristic* model composed of five 3D-interpolation blocks of type *Kennlinie3D*

8 Conclusions

A MODELICA model *Kennlinie3D* for table interpolation in three dimensions is developed. The model is based on the MODELICA 2D-interpolation model *CombiTable2D* which gives access to an efficient interpolation routine provided by Dymola. Interpolation to four dimensions is possible with an additional interpolation level supported by the developed model *WeightedSignals*. In order to allow a large number of parameter studies a method is developed that automatically generates the required interpolation data from a complex solar field model. Due to the universal design of the models they can also be used apart from the solar field application.

Acknowledgements

The authors would like to thank the German Ministry for the Environment, Nature Conservation and Nuclear Safety for the financial support given to the ITES project under contract No. 16UM0064.

References

[1] Eck M., Zarza E., Eickhoff M., Rheinländer J., Valenzuela L. Applied research concerning the direct steam generation in parabolic troughs. Solar Energy, Vol. 74, 2003, pp. 341-351

Apendix: Source code of model *WeightedSignals*

```
model WeightedSignals

  Modelica.Blocks.Interfaces.RealInput  x      "actual value of x";
  Modelica.Blocks.Interfaces.RealInput  u[n]   "values at nodes x_param";
  Modelica.Blocks.Interfaces.RealOutput y      "interpolation result";

  parameter Real    x_param[:] "interpolation nodes"
                           // (here [0°C, 20°C, 40°C] )
  parameter Integer n=size(x_param,1) "Dimension of signal vector";
  parameter Modelica.Blocks.Types.Smoothness.Temp
      smoothness=Modelica.Blocks.Types.Smoothness.LinearSegments
      "smoothness of table interpolation";

  parameter Real[:,:] Weight_matrix     = [x_param, diagonal(ones(n))];
  parameter Integer   Weight_columns[:] =  2:size(Weight_matrix, 2);
  Real                Weight_tableID;
  Real[n]             weight             "weights of the values u[i]";

equation
  for i in 1:n loop
     weight[i] = dymTableIpo1( Weight_tableID, Weight_columns[i], x);
  end for;
  y = sum(  u[i] * weight[i]    for i in 1:n);

when initial() then
   // Initialize Weighting functionality
   Weight_tableID=dymTableInit(1.0,smoothness,"NoName","NoName",Weight_matrix, 0.0);
end when;

end WeightedSignals;
```