ACHIEVEMENTS IN RESULT VISUALIZATION WITH THE COMPUTER NUMERIC E-LEARNING SYSTEM MMT

Irene Hafner[(a)], Martin Bicher[(b)], Thomas Peterseil[(c)], Stefanie Winkler[(d)], Ursula Fitsch[(e)], Nicole Nagele-Wild[(f)], Wolfgang Wild[(g)], Felix Breitenecker[(h)]

[(a)] - (b), (d) - (h) Vienna University of Technology, Institute of Analysis and Scientific Computing
[(c)] dwh Simulation Services
[(a)] irene.hafner@tuwien.ac.at, [(b)] martin.bicher@tuwien.ac.at, [(c)] thomas.peterseil@drahtwarenhandlung.at,
[(d)] stefanie.winkler@tuwien.ac.at, [(e)] ursula.fitsch@tuwien.ac.at, [(f)] nicole.nagele@tuwien.ac.at,
[(g)] wolfgang.wild@tuwien.ac.at, [(h)] felix.breitenecker@tuwien.ac.at

ABSTRACT

This paper discusses the latest developments of the MMT system. MMT, which stands for Mathematics, Modelling and Tools, represents an e-learning system for teaching basic mathematics as well as modelling and simulation. Since the MMT system got its new interface in summer 2010, the range of examples has reached an impressive amount. Apart from examples concerning Linear Algebra and Analysis which are used in lectures of basic mathematics for students of electrical engineering and geodesy and geomatics engineering, the focus of the MMT system has recently been laid on the extension of examples for teaching modelling and simulation to students of Technical Mathematics. Lecturers use this web-interface to explain the current topic by showing examples on the MMT server. In addition, students receive an account to be able to also access these examples on the server from home to train their newly gained knowledge. Recently the data transfer from MATLAB to the MMT server has been renewed completely. The new render files enable a simple exchange of various output data.

Keywords: E-Learning, MATLAB, Simulink, MMT, rendering

1. STRUCTURE OF THE MMT SERVER

The interface of the MMT server is shown in Fig.1.

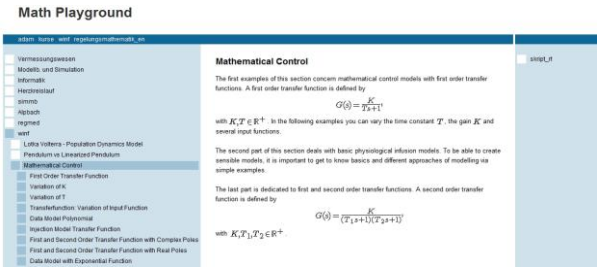

Figure 1: Overview of the MMT Interface

The left section consists of the content tree. Depending on the lecture they are attending, students see a certain selection of examples. The middle section contains a description of the current section or example and the parameter section. On the right side, all files offered for download can be found.

2. EXAMPLES

2.1. Introductory Example

Each example on the MMT server starts with a description providing information about the current topic. In Addition to the short description in the middle section, pdf documentations and pictures can be uploaded from the lecturers. These documents as well as the underlying source code for an example can be downloaded by following the links on the right side.

All parameters for a simulation or calculation can be defined in the section below the description. By clicking the *ok* button, the MMT parameters are transferred to the referring example's source code which is further executed. Fig.2 shows a MMT MATLAB example consisting of one m-file.

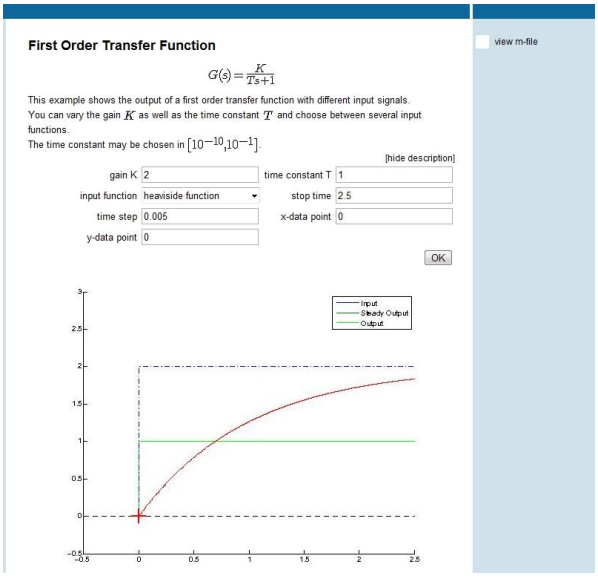

Figure 2: Picture of a MMT Example Accessing One MATLAB File – Description, Parameters and Output

The m-files for a MMT example basically look like usual m-functions consisting of MATLAB code for the

calculation of the example and additional lines to enable the communication with MMT. After clicking the ok – button, the parameters currently selected for the example are converted into strings and stored in a structure variable. This variable is used as input to the m-function. After explanatory comments concerning the topic of the example, authors and last modification date, the strings stored in the structure variable are converted into the desired data formats for further usage (see Fig.3).

```
K = str2double(instruct.var1);
T = str2double(instruct.var2);
number = str2double(instruct.var3);
tend = str2double(instruct.var4);
dt = str2double(instruct.var5);
xxx = str2double(instruct.var6);
yyy = str2double(instruct.var7);
```

Figure 3: Extraction of Input Parameters from the MMT Server in MATLAB

The ensuing code represents the actual code for the respective example. At the end of the m-file, all results are again returned to the MMT server. An overview of the output possibilities on the MMT system is given in section 3.

A MATLAB example for the MMT server can also exist of several m-files. Only the main m-file which has to be the first one to be found in the list of attached m-files contains the additional code for the data exchange with the MMT server. All other m-files needed for the corresponding example are called from the main one and don't have to be adapted in any way.

All MATLAB examples on the MMT server are executed accessing the MATLAB version on the server itself, so it's not necessary to possess a local MATLAB licence.

2.2. Examples Accessing Simulink

The possibility of including examples using several m-files has also enabled the inclusion of Simulink models into the MMT system. Simulink models actually consist of MATLAB code which can also be created graphically using block diagrams. Hence the Simulink model is treated like an additional m-file and called from a MATLAB function which only contains the extraction of input variables, the execution of the Simulink file and the return of the simulation results (see Fig. 4 for the extraction of input parameters from the MMT server and the execution of the mdl-file).

Students can download the source code of the MATLAB function as well as the Simulink model. Since the mdl-file is treated as text file from the browser, it has to be stored locally on the downloader's computer and re-opened with MATLAB to show the corresponding graph model. The successful inclusion of Simulink files also encourages the usage of Simscape on the MMT server, which will open the possibilities of teaching other modelling approaches like Physical Modelling via MMT.

```
phi0 = str2num(instruct.var1);
w0 = str2num(instruct.var2);
k = str2num(instruct.var3);
l = str2num(instruct.var4);
m = str2num(instruct.var5);
t_end = str2num(instruct.var6);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% CALCULATIONS, FUNCTION CALLS AND MAIN PROGRAM %%%%
% All general calculations and operations come here.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% set the workspace to give simulink access to the
% variables
option=simset('SrcWorkspace', 'current');
% starting simulation:
[t,x] = sim('pendulumtrial',t_end,option);

%% GRAPHICAL OUTPUT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Graphical output is generated here.
```

Figure 4: MATLAB Function Extracting Values from the MMT Server and Simulating a mdl-File

3. NEW RENDER FILES AND CREATION OF MULTIPLE OUTPUT

One of the very recent developments for the MMT server has been the enlargement of possibilities to create output from MATLAB files on the MMT system. Before developing the new rendering routines, returning the output from MATLAB to the MMT system was very unintuitive and rather complicated. Up to March 2012 one could only choose between simple textual output written in html-code or one MATLAB figure. Hence the only possibility to show more than one output plot was using subplots, which of course lead to a very unclear picture for a certain amount of subplots. To improve these options, completely new rendering routines have been developed. These routines allow every MMT example to have as much layers of outputs as needed. For each of these layers a number, written in brackets, occurs in the upper right output corner to enable the navigation between them, see Fig. 5. Every layer consists either of textual output, a static figure or an animated gif with optional title.

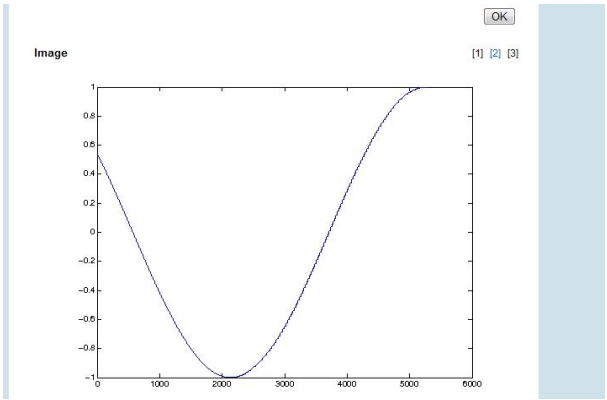

Figure 5: Switching between Multiple Output Layers by Clicking on the Numbers in Brackets

The return variable for the MMT server has to be a string containing html code with the information on the creation of all desired layers. To prevent the manual creation of this string for every single example, a variable *r* of type cell array is used. This array contains cells of strings with information about every layer

which has to be created. To initialize the cell array, the first function to be called in the output section of the MATLAB function is *adamInitialize*. Conventionally the cell array is called *r*, so the function call would be

```
r=adamInitialize();
```

The strings for the creation of all intended layers are created by using the new render files described in the following subsections. After generating as much layers as needed for the referring example, calling

```
retstr=adamComposeResultString();
```

takes the information from the cell array to create an Excel-file containing strings which represent the html-code with the information for the construction of the required layers.

3.1. Rendering Textual Output

In former times, the creation of textual output did require html knowledge, as can be seen in Fig. 6.

```
retstr = 'TEXT';
retstr = [retstr, sprintf('The vector product a x b
  results in: ( %s )', num2str(ab))];
retstr = [retstr, sprintf('<br><br>The vector product
  b x a results in: ( %s )',num2str(ba))];
retstr = [retstr, sprintf('<br>The vector product
 -a x b results in: ( %s )', num2str(minab))];
retstr = [retstr, '<br><br>You see that
 -a x b = b x a']
```

Figure 6: Rendering Text without the New Render Functions

Using the new render function *adamRenderText*, common string variables can be used as textual return value. As you see in Fig. 7, *adamRenderText* reqires several input parameters.

```
function [r]=adamRenderText(r,instruct,text,varargin)
%Function to render Textual output


%handling with optional parameters
if length(varargin)>0
 s.title=varargin{1};
end;


%defining class and adding it to the cell array
s.type='TEXT';
s.text=text;
r{end+1}=s;
end
```

Figure 7: Source Code for the New Text Rendering Function

r is the already allocated cell array being either empty but created with *adamInitialize* or containing the information of previous layers. As seen here the input structure *instruct* of the MMT example has to be used as an input parameter of the render function too as it not only contains the manual chosen parameters of the MMT experiment but also a unique filename extension (*instruct.mlimgfilename*) for saving files to a temporary folder. *text* contains the common matlab string which the programmer would like to have displaced on the layer. *varargin* is an optional input which can contain a string with the desired title for the layer. The function appends a structure variable containing strings with the information about the title (line 6 in Fig.7), type (line 10) and content (line 11) of the output, which would now be *text*, to the cell array. It's important to call this function similar to

```
r=adamRenderText(r,instruct,'sample text',...
'optional title');
```

to make sure the existing cell array is appended.

3.2. Rendering Static Images

As mentioned before, until this year's March the only way to return more than one picture was using subplots in MATLAB. How unclear the resulting picture can get this way is shown in Fig. 8.

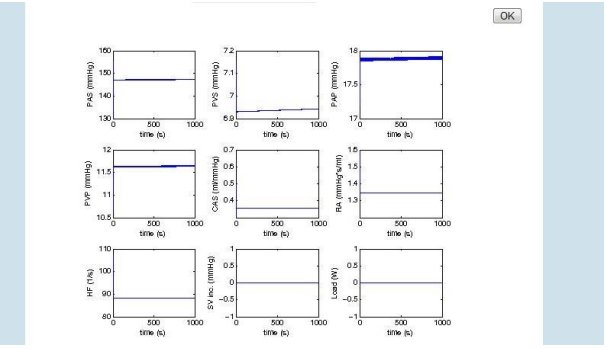

Figure 8: Output of Nine Figures by Using Subplots

In addition, returning this picture to the MMT server required rather unintuitive MATLAB code, as can be seen in Fig. 9.

```
% produces a *.jpeg image for graphical output.
drawnow;
wsprintjpeg(Pic, instruct.mlimgfilename);
retstr = 'IMAGE';
```

Figure 9: Former Code for the Submission of a Picture to the MMT Server

Now it's possible to create one layer for every picture the user wants to be returned with much easier code since all additional work is done with the new function *adamRenderImage*. Before calling this function, a MATLAB figure has to be created. Its visibility is usually set to 'off' as graphical output on the server which executes MATLAB in console mode and does not provide a graphical interface, is suppressed anyway. A source code example for this is given below.

```
Pic = figure('visible','off');
plot(sin(0:0.001:2*pi));
```

The figure further has to be transferred to the render function by calling

```
r=adamInitialize();
r=adamRenderImage(r,instruct,Pic,'Sine');
retstr=adamComposeResultString(r);
```

The source code for *adamRenderImage* is shown in Fig.10.

```
function [r]=adamRenderImage(r,instruct,Pic,varargin)
%Routine for image saving to temp folder as .png

%handling with optional parameters
if length(varargin)>0
    s.title=varargin{1};
end;

%Defining class entry for cell array
s.type='IMAGE';
s.imagetype='png';

%Defining filename of .png file
picId=length(r);
filename=sprintf(instruct.mlimgmultipath,picId);

%Setting figure position fitting to 576*432 pixels
%72pixels/inch is the default resolution for
nonscreen
%figureoutput
set(Pic,'PaperPositionMode','manual','PaperUnits',...
  'inches','PaperPosition',[0 0 576/72 432/72]);

%Saving process, using filename and
%resolution '-r72' => 72pixels/inch
print(Pic, '-dpng', '-r72',[filename,'.png']);

%saving class to cell array
r{end+1}=s;
end
```

Figure 10: Source Code for the New Rendering Function for Images

The input parameters are basically the same as in *adamRenderText* whereupon the text string is replaced by the MATLAB figure. Besides appending again the information about type and title (lines 6, 11 and 12), this function prints the MATLAB plot to a pre-allocated picture on the MMT server (line 26).

3.3. Rendering Animated Images

```
function
[r]=adamRenderAnimatedCreate(r,instruct,Pic,varargin)
%Routine for image saving to temp folder as .png

%handling with optional parameters
if length(varargin)>0
    delay=varargin{1};
    if length(varargin)>1
        s.title=varargin{2};
    end;
else
    delay=0;
end;

%Defining class entry for cell array
s.type='IMAGE';
s.imagetype='gif';
%Defining filename of .gif file
picId=length(r);
filename=sprintf(instruct.mlimgmultipath,picId);

%Setting figure position fitting to 576*432 pixels
%72pixels/inch is the default resolution for
nonscreen
%figureoutput
set(Pic,'PaperPositionMode','manual','PaperUnits','in
ches','PaperPosition',[0 0 576/72 432/72],'Color',[1
1 1]);

%Saving process, using filename and
%resolution '-r72' => 72pixels/inch
PP = hardcopy(Pic,'-dOpenGL','-r72');
[P,cm]=rgb2ind(PP,256);
imwrite(P,cm,filename,'gif','Loopcount',inf,'DelayTim
e',delay);

%saving class to cell array
r{end+1}=s;
end
```

Figure 11: Source Code for the New Rendering Function Creating the First Frame of an Animated gif

For animated gifs, plots have to be created for every frame of the picture. To create the first frame, the picture is initialized with *adamRenderAnimatedCreate* (see Fig.11).

The only significant difference to *adamRenderImage* is the option of a delay-time defining how fast the frames of the gif are intended to change. Setting the delay to zero causes the fastest change of frames. All frames besides the first one are rendered by calling the function *adamRenderAnimatedAppend*, which is shown in Fig. 12.

```
function
[r]=adamRenderAnimatedAppend(r,instruct,Pic,varargin)
%Routine for image appending to an existing animated-
gif


%handling with optional parameters
if length(varargin)>0
    delay=varargin{1};
else
    delay=0;
end;


%Detecting filename of .gif file to append to
picId=length(r);
filename=sprintf(instruct.mlimgmultipath,picId-1);


%Setting figure position fitting to 576*432 pixels
%72pixels/inch is the default resolution for
nonscreen
%figureoutput


set(Pic,'PaperPositionMode','manual','PaperUnits',...
  'inches','PaperPosition',...
  [0 0 576/72 432/72],'Color',[1 1 1]);


%Saving process, using filename and
%resolution '-r72' => 72pixels/inch


PP = hardcopy(Pic,'-dOpenGL','-r72');
[P,cm]=rgb2ind(PP,256);
imwrite(P,cm,filename,'gif','WriteMode','append',...
  'DelayTime',delay)
end
```

Figure 12: Source Code for the New Rendering Function Appending Frames to an Animated gif

An example source code for the creation of an animated gif is given below.

```
Pic = figure('visible','off');

plot(sin((0:0.001:2*pi)*t));

r=adamInitialize();
r=adamRenderAnimatedCreate(r,instruct,Pic,0,'sine');

for t=1:time;
   plot(sin((0:0.001:2*pi)*t/20));
r=adamRenderAnimatedAppend(r,instruct,Pic,0);
end;

retstr=adamComposeResultString(r);
```

3.4. Example with Multiple Output

The following example shows how multiple output layers can be returned to the MMT server. To enable students downloading the source code from the MMT server to easily convert the file into an executable one

for their own computer, all parts only necessary for the data exchange with the MMT server are framed by

```
%=========== BEGIN OF SPECIFIC CODE - CUT HERE ======
```

and

```
%=========== END OF SPECIFIC CODE - CUT HERE ========
```

The source code of an example creating layers for textual, animated and static images is shown in Fig. 13.

```
1   %% GRAPHICAL OUTPUT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2   % Graphical output is generated here.
3   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4
5   %======= BEGIN OF SPECIFIC CODE - CUT HERE ======
6   % disables graphical output on the remote system.
7   Pic1 = figure('visible','off');
8   %======= END OF SPECIFIC CODE - CUT HERE ========
9
10  plot(sin((0:0.001:2*pi)*t));
11
12  %======= BEGIN OF SPECIFIC CODE - CUT HERE ======
13  r=adamInitialize();
14  r=adamRenderAnimatedCreate(r,instruct,Pic1,0,...
15  'Animation');
16  %======= END OF SPECIFIC CODE - CUT HERE ========
17
18  for t=1:100;
19  plot(sin((0:0.001:2*pi)*t/20));
20
21  %======= BEGIN OF SPECIFIC CODE - CUT HERE ======
22  r=adamRenderAnimatedAppend(r,instruct,Pic1,0);
23  %======= END OF SPECIFIC CODE - CUT HERE ========
24
25  end;
26
27  %======= BEGIN OF SPECIFIC CODE - CUT HERE ======
28  % disables graphical output on the remote system.
29  Pic2 = figure('visible','off');
30  %======= END OF SPECIFIC CODE - CUT HERE ========
31
32  plot(cos(1:0.001:2*pi));
33
34  %======= BEGIN OF SPECIFIC CODE - CUT HERE ======
35  r=adamRenderImage(r,instruct,Pic2,'Image');
36  r=adamRenderText(r,instruct,...
37  'To be or not to be...','Text');
38  retstr=adamComposeResultString(r);
39  %======= END OF SPECIFIC CODE - CUT HERE ========
40
41  %% END OF EXAMPLE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
42
43  end
```

Figure 13: Creation of Output Layers for Three Output Types in One Example

In lines 7-10 the first frame for an animated gif is created. Before returning this first output to the MMT server by *adamRenderAnimatedCreate* in line 14, *adamInitialize* has to be called (line 10). Lines 18-25 show the code for creating and appending further frames to this animated image with delay 0. The creation of a static picture with the catching title 'image' is shown in lines 29-35. In line 36, a short text is returned to the third frame. The cell array created by the call of all render functions is further transferred to *adamComposeResultString* (see line 38) which returns the string *retrstr*. This string finally contains all information needed from the MMT system to establish enough output layers with the correct title and contents while the pictures have already been transferred to the server during the execution of the render functions.

The outputs of this MATLAB function on the MMT server can be seen in Fig. 14. Of course, due to the disability of print media to show animation, the movement for the first output has to be imagined.

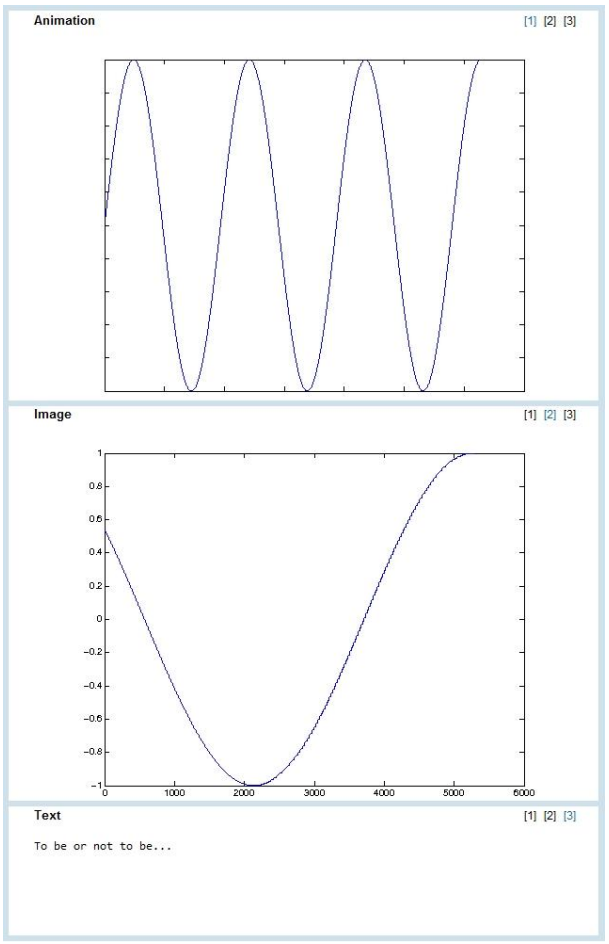

Figure 14: Different Output Layers of One MMT Example

4. USAGE OF THE MMT SERVER IN EXAMS

The MMT system has also become a very helpful tool for exams in lectures about modelling and simulation. The questions for these exams are posed via TUWEL, a moodle based e-learning tool of the Vienna University of Technology. To answer these questions, students have to log in to the MMT server and vary input parameters to achieve a certain result. An example for such a question could be finding the maximum step size for a certain solver algorithm to stay within a given error tolerance. Another question demands the evaluation of the turning point of a PT2 controller for certain input parameters which have to be set on the MMT server. The TUWEL question for this example can be seen in Fig.15.

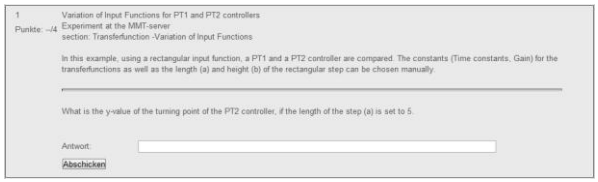

Figure 15: TUWEL Question for an Exam in Cooperation with the MMT System

By following the link in TUWEL, the students find the MMT example corresponding to this question, see Fig.16.

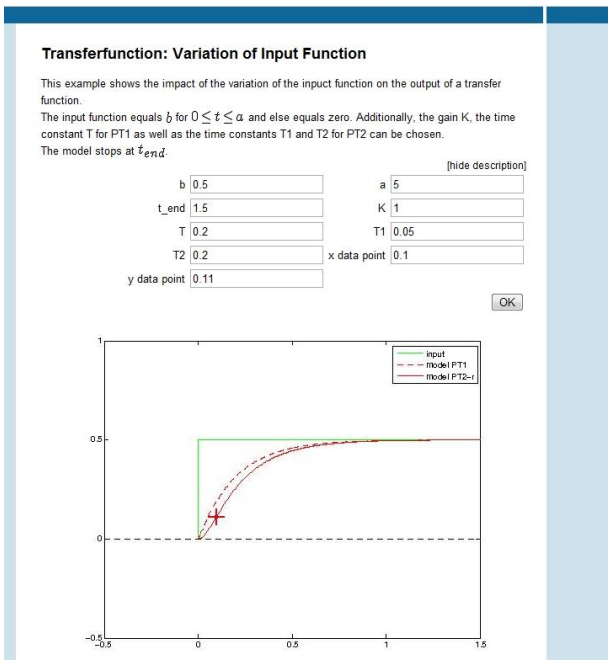

Figure 16: MMT Example Corresponding to a TUWEL Exam Question

For precise reading of the values to be found, additionally to the parameters for the PT controllers a data point can be set. At the chosen point a red cross is plotted which eases spotting certain data. The students further fill in the values they found out to the answer field in the TUWEL question. Grading is done independently by TUWEL.

5. CONCLUSION AND OUTLOOK

While most of the examples currently existing on the MMT server have been implemented by programmers of the Institute of Analysis and Scientific Computing of the Vienna University of Technology, recently many examples which have been developed during projects, diploma and bachelor theses are tested, validated and further included to the MMT system. That way other students are able to profit from those projects which else would probably have been forgotten.

All in all, the MMT server has become a very important tool for teaching basic mathematics as well as modelling and simulation at the Vienna University of Technology. Although all examples currently available for teaching on the MMT server are implemented in MATLAB, MMT has recently been developed to include also examples implemented in AnyLogic using Java applets which offer way more interesting possibilities for the output. Additionally, to loosen the dependency on the commercial software MATLAB, one of the next developments concerning the MMT system will be the inclusion of examples accessing Octave.

ACKNOWLEDGEMENTS

This work has been realized in the context of the ‘Applied Modelling, Simulation And Decision Making‘ project and funded by means of the City of Vienna by the ZIT GmbH - the Technology Agency of the City of Vienna, a subsidiary of the Vienna Business Agency.

The MMT server is administered in cooperation with the dwh Simulation Services.

REFERENCES

Winkler, S., Körner, A., Hafner, I., 2010. MMT – A web-based e-learning System for Mathematics, Modeling and Simulation using MATLAB. *7th EUROSIM Congress on Modelling and Simulation*, paper 231, Prague (Czech Republic)

Körner, A., Zauner, G., Schneckenreither, G., 2009. Ein e-learning System für MMT – Mathematik, Modellbildung und Tools, Systemerweiterung und Einbindung von graphischer Modellbildung. *20th Symposium Simulation Techniques*, pages 87–94, Cottbus (Germany)

Körner, A., Hafner, I., Bicher, M., Winkler, S., Breitenecker, F., 2011. MMT - A Web Environment for Education in Mathematical Modelling and Simulation. *ASIM 21. Symposium Simulationstechnik*, Winterthur (Switzerland), ISBN: 978-3-905745-44-3

Hafner, I., Bicher, M., Winkler, S., Fitsch, U., 2012. MMT - An E-Learning System based on Computer Numeric System for teaching Mathematics and Modelling. *MATHMOD 2012 - 7th Vienna Conference on Mathematical Modelling*, Vienna (Austria)