G2TP Function

From a computational point of view, handling the TP model transformation even for a 3-way tensor is a challenging task, not to mention higher dimensions. By the way, it is desirable to have a generic algorithm that makes use of all advantages of the transformation, so that gain scheduled controllers could be easily synthesized in a reasonable amount of time.

Grid-based and polytopic LPV models complement each other when it comes to control design approaches. In MATLAB, each model representation can be accessed through corresponding toolboxes - *LPVTools* and *TP Tool*, respectively. However, those have different input-output interfaces which makes it inconsistent for smooth workflow.

The function described in this chapter aims to build a bridge between the two model representations allowing the user to use benefits of both freely within continuous control design.

1 Structure

The core function has 1 required input, 6 optional inputs, 3 outputs, and is comprised by 12 subfunctions. Some have their own subfunctions.

The function can be called from the Command Window entering the following command

```
varargout = G2TP(LPV_0,varargin)
```

where `LPV_0` - input grid-based LPV model;
`varargin` - optional input arguments put in a fixed order;
`varargout` - optional output arguments put in a fixed order.

Cell array `varargin` contains optional inputs such as

1. `wtype` - type of the weighting functions;
2. `ParNum_keep` - number of kept scheduling parameters which are assumed to be completely measurable;
   Remark: it is assumed that parameters to be kept are in the beginning and there are no parameters with possible measurement uncertainty between them.
3. `mode` - selection mode of singular values described below;
4. `TOL_NORM` - upper bound of the Vinnicombe Gap Metric δ_ν between the initial model and reduced one after executing RHOSVD [1];
5. `Pause_T` - pause between specific calculations;
6. `Basis` - Boolean variable enabling generation of basis functions.

If no optional input is specified, there are default settings predefined.
Cell array `varargout` contains optional outputs such as

1. `LPV_TProd` - reduced grid-based LPV model defined in **??**;
2. `BASIS_GRID` - basis functions of the input model represented in the form of their values and values of their first order partial derivatives evaluated at the grids point for each scheduling parameter;
3. `BASIS_TP` - basis functions of the reduced output model.

2 Subfunctions

In what follows, each subfunction is described. Also, all second-level subfunctions are mentioned as well.

2.1 Input data

The function executing input data formation

```
[wtype, ParNum_keep, mode,TOL_NORM,Pause_T,Basis] = Num_in(nargin,varargin)
```

where `nargin` - number of the function input arguments.

First, the default settings stored in a separate class `LPV_TP_opt` are returned. Then, if some optional arguments are specified, the input data is redefined.

2.2 Scheduling parameters

Before going further, it should be noted that LPV systems supported by *LPVTools* are `pss` objects and have fixed structure with the following properties:

1. `Domain` := (Ω, M). Moreover, rate bounds representing limits of the first order derivative for each scheduling parameter are stored here as well. A trivial notation could be $\boldsymbol{a_b} \leq \dot{\boldsymbol{\rho}}(t) \leq \boldsymbol{b_b}$, where $\boldsymbol{a_b}$ and $\boldsymbol{b_b}$ are the corresponding vectors containing the limits;
2. `Data` := $\{(\mathbf{A}(\boldsymbol{\rho}_k), \mathbf{B}(\boldsymbol{\rho}_k), \mathbf{C}(\boldsymbol{\rho}_k), \mathbf{D}(\boldsymbol{\rho}_k))\} \in (\Omega, M)$;
3. `Parameter` := $\boldsymbol{\rho}_k(t) \in (\Omega, M)$.

In fact, sets $\{(\mathbf{A}(\boldsymbol{\rho}_k), \mathbf{B}(\boldsymbol{\rho}_k), \mathbf{C}(\boldsymbol{\rho}_k), \mathbf{D}(\boldsymbol{\rho}_k))\}$ form an array of state-space models (`ss` objects). To enhance the flexibility of the function, it is allowed to use both the initially given grid-based LPV representation and array of SS models as an input `LPV_0`.

The function which main task is to calculate the dimension of $\boldsymbol{\rho}(t)$

```
[ParNum, LPV_0,Tick_ss] = SchParams(LPV_0,ParNum_keep,wtype,mode,Basis)
```

where `ParNum` := $dim(\boldsymbol{\rho}(t))$;

`Tick_ss` - Boolean variable of the subfunction `Typo_ss` checking for typos in optional input arguments in case if the input system is a `ss` object.

What is more, if `LPV_0` is either a `ss` or `pss` object, all the required information including $dim(\boldsymbol{\rho}(t))$ and its data for each scheduling parameter can be obtained from the initially system automatically. Otherwise, the data needed is constructed once again automatically utilizing integers as trivial names and range $[-1, 1]$. This is done with the help of a class `SS_Array` defined specially to reproduce the structure of a grid-based LPV model. This is why `LPV_0` is both the input and output of this subfunction.

2.3 Typos

The function checking for typos in each optional input argument in case if the input system is a `pss` object

```
Tick_pss = Typo_pss(LPV_0,wtype,ParNum_keep,ParNum,mode,Basis)
```

The goal of such a function is to improve the interaction between the user and the core function.

2.4 Uncertainty

It may happen that some scheduling parameters cannot be precisely known and given as variations of nominal values so we consider the associated scheduling parameters as irrelevant and want to combine it in terms of the stored data to reduce the dimension complexity of $\mathcal{S}^D$. The subfunction executing the mentioned algorithm

```
[LPV_0,Par_range_pr,ParNum] = sys_reshape(LPV_0,ParNum_keep,ParNum)
```

where `Par_range_pr` - product of all grid points of the corresponding scheduling parameters in each dimension of (Ω, M).

Reshape is done when the condition `ParNum_keep` $\neq$ `ParNum` is fulfilled.

2.5 Discretized state-space tensor

The function executing formation of $\mathcal{S}^D$

```
S_G = Tensor_S(LPV_0,Par_range_pr,ParNum)
```

$\mathcal{S}^D$ becomes a multidimensional `double` object at this stage so its reformulation is required afterwards.

2.6 TP-type HOSVD-based transformation

Finite element polytopic form is obtained from the subfunction

```
[U_weigh_fun,S_weigh_fun,Par_vec,ParNum] = sv_selection(ParNum,ParNum_keep,mode,
TOL_NORM,S_G,Par_range_pr,LPV_0,wtype,Pause_T)
```

where `U_weigh_fun` := $\mathbf{W}^{D(\Omega,M)}(\boldsymbol{\rho}(t))$;
`S_weigh_fun` := $\mathcal{S}^D_{red}$;
`Par_vec` - vector for HOSVD of the set of LTI models.

$\mathbf{W}^{D(\Omega,M)}(\boldsymbol{\rho}(t))$ is a cell array each cell of which is actually $\mathbf{W}^{D(\Omega,M)}(\boldsymbol{\rho}_n(t))$. Also, `Par_vec` tells us in which dimension of $\mathcal{S}^D$ HOSVD is to be executed. Speaking of that, there is a special case when some of the dimensions of $\boldsymbol{\rho}(t)$ are scalars. HOSVD cannot be executed for these dimensions so `ParNum` is reduced in this case and `Par_vec` is reconstructed.

The idea of obtaining the first two outputs is based on the RHOSVD which is implemented through discarding of singular values for each scheduling parameter and generation of convex hulls of the specified type. In this thesis, the CNO type of weighting functions is used.

First, SVD for each matrix $\mathbf{W}^{D(\Omega,M)}(\boldsymbol{\rho}_n(t))$ is executed which allows to discard reasonably small singular values. If the data cannot be handled by the personal computer because of its too large size, so-called tall arrays are first constructed and then evaluated to resolve a lack of memory. Next, generation of convex hulls is done for each truncated $\mathbf{W}^{D(\Omega,M)}(\boldsymbol{\rho}_n(t))$. Finally, $\mathcal{S}^D_{red}$ is obtained following **??**.

Since model reduction is based on the selection of singular values, it was decided to let the user choose between the auto mode of the selection and manual mode.

Auto mode

Auto mode is built on two assumptions:

1. Maximum and minimum possible numbers of kept singular values are 6 and 2, respectively;

2. Reduced grid-based LPV model is compared to the original one after each iteration based on δ_ν metric which calculates the distance between each pair of LTI systems.

The reduction starts from keeping 6 singular values for each scheduling parameter and continues with the iteration step of 1 until either the smallest possible number of singular values (same for each scheduling parameter) is reached or the maximum δ_ν metric for some pair is greater than the predefined upper bound `TOL_NORM`. In latter case, a step back is produced automatically to fulfill the tolerance condition. By the way, the IRNO type of weighting functions is prescribed during ordinary iteration in sake for short running time. The HOSVD-based canonical form as well as the nu-gap metrics are obtained from the second-level subfunction `SSM_red`.

It may happen that the resulting number of sungular values fulfilling the tolerance condition and defining successfully reduced model makes it infeasible to actually construct such a model. By default, *LPVTools* increases the number of kept singular values in this case automatically, however after conducting comparison tests an extra condition was added for executing a step back within the iterative process since δ_ν seems to be smaller this way.

Manual mode

Manual mode allows the user to specify how many singular values are to be kept for each scheduling parameter individually. This is done through interaction with the Command window.

Once again, execution of the step back in case of construction infeasibility still holds.

2.7 Sorting of vertices

The goal here is to sort properly the vertex systems since after obtaining a polytopic form, order of the data connecting $\mathbf{W}^{D(\Omega,M)}(\boldsymbol{\rho}(t))$ with $\mathcal{S}^D_{red}$ is lost so there is a subfunction executing that kind of sorting

```
[idx_u_asc, I_perm] = Vertex_sys_sort(ParNum,Par_vec,U_weigh_fun)
```

where `idx_u_asc` - cell array containing sequences of vertices put in an ascending order in each cell;

`I_perm` - cell array containing an arrangement of the elements of `idx_u_asc` in each cell.

2.8 Sorting of indices

Next step is to put each element of $\mathcal{S}^D_{red}$ in the correct order according to its dimensions.

The corresponding subfunction executing the aforementioned task

```
[I_perm_ordered,Par_range_pr_hull] = Indices_sort(ParNum,U_weigh_fun,I_perm)
```

where `I_perm_ordered` - row vector containing all the indices put in the correct order;

`Par_range_pr_hull` - product of dimensions of $\mathcal{S}^D_{red}$.

In other words, `Par_range_pr_hull` gives the total number of grid points defined over (Ω, M).

Sorting algorithm

It is noted that the developed sorting algorithm seems to be just an acceptable option which is not claimed to be an optimal solution.

Consider the following steps that are implemented within the subfunction.

Construction of initial row vector and its concatenation

First, construction of the initial vector P is produced as follows

Algorithm 1 Initial row vector

$P \leftarrow NaN$
for $ii = 1 : (ParNum - 1)$ **do**
  for $j = 1 : \text{numel}(I\{1, ii+1\})$ **do**
    $P(1, jj) \leftarrow \prod_{ii=1}^{ii} (N\{1, ii\}) \times (I\{1, ii+1\}(j, 1) - 1)$
  end for
end for
$P \leftarrow [I\{1, 1\}^T \, P]$ ▷ *Concatenation of vectors*

where $P \equiv$ `Product_array`
$N\{1, ii\} \equiv$ `numel(I_perm{1,ii})`
$I \equiv$ `I_perm`

Example

Consider the following case:

$$\begin{cases} I\{1,1\}^T = [4\,3\,1\,2\,5] \\ I\{1,2\}^T = [2\,1\,3] \\ I\{1,3\}^T = [3\,2\,1] \end{cases}$$

Algorithm 1 results in

$$P = [4\,3\,1\,2\,5\,5\,0\,10\,30\,15\,0]$$

Calculation of the correct order of indices

Variable `I_perm_numbers_cell` $\stackrel{\triangle}{=} I^*$ is used to partition the values of P into separate cells based on the number of elements stored in the corresponding cells of I.

Therefore, following the example case

$$\begin{cases} I^*\{1,1\} = [4\,3\,1\,2\,5] \\ I^*\{1,2\} = [5\,0\,10] \\ I^*\{1,3\} = [30\,15\,0] \end{cases}$$

Denoting a row vector containing all the ordered indices by ID, general algorithm for indices can be written as follows

Algorithm 2 Correct order of indices

$ID, ID_0 \leftarrow NaN$
if $ParNum == 1$ **then**
  $ID = I^*$
else if $ParNum == 2$ **then**
  for $jjj = 1 : \text{numel}(I^*\{1,2\})$ **do**

$ID_0 \leftarrow I^*\{1,1\} + I^*\{1,2\}(1, jjj)$ ▷ *Matrix-slacar addition*
$ID \leftarrow [ID\, ID_0]$ ▷ *Update of the index list*
end for
else
for $jjj = 1 : \text{numel}(I^*\{1,2\})$ **do**
$ID_0^* \leftarrow I^*\{1,1\} + I^*\{1,2\}(1, jjj)$
$ID_0 \leftarrow [ID_0\, ID_0^*]$
end for
for $iii = 3 : \text{numel}(I^*)$ **do**
for $jjj = 1 : \text{numel}(I^*\{1, iii\})$ **do**
$ID_{00} \leftarrow ID_0 + I^*\{1, iii\}(1, jjj)$
$ID \leftarrow [ID\, ID_{00}]$
end for
$ID_0 \leftarrow ID$
if $iii \neq \text{numel}(I^*)$ **then**
$ID \leftarrow [\ ]$ ▷ *Deletion of the temporal data if the loop is not complete*
end if
end for
end if

Thus, algorithm 2 considers 3 different cases depending on the dimension of $\boldsymbol{\rho}(t)$. Based on the data above:

1. Single scheduling parameter:

$$ID = I^*\{1,1\} = [4\,3\,1\,2\,5]$$

2. Two scheduling parameters:

$$ID = [9\,8\,6\,7\,10\,4\,3\,1\,2\,5\,14\,13\,11\,12\,15]$$

3. Three and more scheduling parameters:

$$ID = [39\,38\,36\,37\,40\,34\,33\,31\,32\,35\,44\,43\,41\,42\,45\,24\,23\,21\,22\,25\,19\,18\,16\,17\,20\,29\,28\,26\,27\,30\,9\,8\,6\,7\,10\,4\,3\,1\,2\,5\,14\,13\,11\,12\,15]$$

Afterwards, ID is used to restore the correct order of the elements in the TP-based model through turning it into a tensor.

2.9 Ordered reduced polytopic representation

Ordered polytopic form is obtained from the subfunction

```
[S_perm,Par_range_vec_red,U]  = S_G_red(S_weigh_fun,I_perm_ordered,I_perm,
ParNum,Par_vec,U_weigh_fun,LPV_0)
```

where `S_perm` := $\mathcal{S}_{red}^D$ with permuted elements;
`Par_range_vec_red` - row vector containing dimensions of $\mathcal{S}_{red}^D$;
`U` := $\mathbf{W}^{D(\Omega,M)}(\boldsymbol{\rho}(t))$ with permuted elements.

2.10 LTI grid made up of the vertices

Multidimensional array of SS models is created by the following subfunction

```
LPV_TP = LTI_grid(Par_range_pr_hull,S_perm,LPV_0,Par_range_vec_red)
```

2.11 Reduced grib-based LPV model

Reduced grid-based representation of the initial grid-based LPV model is obtained executing

```
[LPV_TProd,BASIS_GRID,BASIS_TP,Tick_bad_precision] = LPV_TP_red(ParNum,LPV_0,
idx_u_asc,LPV_TP,Basis,U)
```

where `LPV_TProd` - reduced grib-based LPV model,

`Tick_bad_precision` - logical variable checking for presence of identical grid points in the reduced model for each scheduling parameter.

In fact, `BASIS_GRID` and `BASIS_TP` are obtained utilizing another second-level subfunction for basis functions generation that is introduced below.

By the way, state names, input names, and output names of the initial grid-based LPV model are assigned to the corresponding character variables of the reduced grid-based LPV model as all the data is lost after obtaining the disretized tensor $\mathcal{S}^D$.

2.12 Output data

The subfunction generating outputs

```
Output(S_G,Par_vec,ParNum,LPV_0,U,Pause_T)
```

Basically, all singular values for each scheduling parameter are displayed in the Command Window and the weighting functions of the reduced model are plotted employing piecewise linear approximation between each pair of adjacent grid points.

3 Basis functions

One of the crucial steps in controller design is to set proper behavior of a controller in response to the changes in the system. In case of the LMI-based design, this can be done utilizing the so-called basis functions. Basically, these functions are constructed from the weighting functions directly.

Formally, a basis function following the structure in **??** could be defined as

$$f \in \mathcal{C}^1 : \mathbb{R} \to \mathbb{R}, \forall n, sv_j : f(\rho_{n,w_i}) = \omega_{w_i,sv_j} \tag{1}$$

where $w_i = \{1, 2, \cdots, w_n\}$;

$sv_j = \{1, 2, \cdots, sv_n\}$.

In MATLAB, one needs both values of the basis functions evaluated at the corresponding grid points and their first partial derivatives to create an object containing discrete data for controller design.

Thus, the only question in this case is how one can approximate the derivatives, since the functions values are already known after executing the TP model transformation. A common approach is the numerical differentiation.

By the way, approximation of a basis function by picking combinations of primitive continuously differentiable functions for understanding system's behavior sounds promising since well-behaved functions allow to set a smooth and quite accurate behavior of controllers. However, this is not a trivial task since there is no such a recipe. In most cases, the best one can do is to guess a suitable combination approximating the functions.

To solve this problem, the second-level subfunction `Basis_gen` was implemented to generate basis functions with acceptable accuracy regardless of their potential shape.

The subfunction utilizes piecewise cubic hermite interpolating polynomials that are essentially piecewise cubic polynomials interpolating the given data with specified derivatives at the interpolating points [2].

If one considers two adjacent grid points x_l and x_r, - left and right points, respectively - then the corresponding PCHIP has the following form

$$f(x) = a_3(x - x_r)^2(x - x_l) + a_2(x - x_r)(x - x_l)^2 + a_1(x - x_l) + a_0 \tag{2}$$

where $a_i, i = \{0, 1, 3\}$ are polynomial coefficients.

This family of cubic polynomials has an advantage of preserving the curve's shape by passing through the specified grid points. Nevertheless, first order derivatives evaluated at joint points using different piecewise polynomials are not the same.That is why it was decided to break down the approximation of basis functions into 3 parts:

1. Applying PCHIPs for an initial approximation;
2. Grid refinement based on the data obtained from the initial approximation and performed two times following an iterative process to make further calculations more accurate;
3. Numerical differentiation on the refined grid. For the last grid point, the derivative is evaluated using analytical form of the corresponding PCHIP.

References

[1] Zhou, K., Doyle, J.C., Essentials of Robust Control. London, UK: Pearson, 1997.

[2] C. Moler, Numerical Computing with MATLAB, Chapter 3. Interpolation, 2004.