Structural analysis in Julia for dynamic systems in OpenModelica

Liubomyr Vytvytskyi Bernt Lie

Department of Electrical engineering, Information Technology and Cybernetics, University of South-Eastern Norway, Porsgrunn, Norway, {Liubomyr.Vytvytskyi,Bernt.Lie}@usn.no

Abstract

In control theory for dynamic systems, the information about observability and controllability of states plays a key role to evaluate the possibility to observe states from outputs, and use inputs to move states to a desired position, respectively. Th automatic determination of observability and controllability is possible, in particular for linear models where typically observability and controllability grami-ans are considered. In the case of large scale systems, e.g., complex models of regional energy systems, standard analysis becomes challenging. For large scale systems, structural analysis based on directed graphs is an interesting alternative: structural observability (or: controlla-bility) is a necessary requirement for actual observability (or: controllability). Directed graphs can be set up directly for linear models, but can also be extracted from nonlinear models.

Modelica is a suitable language for describing large scale models, but does not support graph algorithms. One possibility is to integrate the Modelica model into a language supporting graph algorithms, e.g., Julia: this integration can be done using package OMJulia which works with the free tool OpenModelica. OMJulia does not give direct access to the nonlinear model in Modelica, but a linear model approximation can be extracted and used for setting up the system graph. In this study, an experimental implementation of automated structural analysis is done in Julia using the *LightGraphs.jl* package. As an example, this structural analysis is tested on hydropower models of different complexity that are modelled in OpenModelica using our in-house hydropower Modelica library — *OpenHPL*, where different models for hydropower systems are assembled.

Keywords: observability, controllability, structural analysis, graph theory

1 Introduction

1.1 Background

Modelling and simulation of dynamic systems (e.g., a hydropower system in this paper) plays an important role as efficient analysis tools for control analysis and design. As an example, tools for designing a new or testing an existing controller for stability and performance in different operating regimes might be of interest.

Model based analysis of state observability and controllability is important for control design, and it is of interest to consider tools for aiding such analysis. Classically, observability and controllability properties might be checked using the well known tests based on rank conditions, (Simon, 2006; Šiljak, 2011). However, numerical problems can arise for the rank computations in complex, large scale systems. Still, structural observability and controllability based on the system structure can be used in such cases due to the simplicity of these methods. In addition, a relative degree of the system indicates how directly control inputs affect outputs, and can also be defined based on the system structure. Assuming linear models, analysis tools based on graph theory can be implemented in Julia[1], e.g., using the *LightGraphs.jl* package.

Models of various dynamic systems might be directly modeled in Julia using the *DifferentialEquations.jl*[2] (Rackauckas and Nie, 2017) and *ControlSystems.jl*[3] packages. However, an object-oriented language such as Modelica[4] has richer support for describing complex, large scale systems with inputs and outputs. One such Modelica based tool is OpenModelica[5] which offers an open-source modeling and simulation environment. OpenModelica also comes with the OMJulia.jl package which offers integration of Modelica models in Julia.

1.2 Previous Work

Basic graph theory for different engineering applications is provided in (Deo, 2017). Structural modeling and analysis of complex systems are described by (Šiljak, 2007, 2011; Lunze, 1992; Boyd and Vandenberghe, 2018). Based on this graph theory, large scale systems can be further tested and analyzed for control and parameter estimation purposes; see, e.g., (Perera, 2016) who used structural analysis of Modelica models in JModelica[6] and Python[7] to analyze an industrial copper electrowinning process.

The OMJulia package[8] (Julia API) for OpenModelica provides possibilities to run simulations and carry out linearization of OpenModelica. Julia in turn gives rich possibilities for plotting, analysis, and optimization (e.g., using

[1] https://julialang.org/
[2] https://goo.gl/5wxZfR
[3] https://goo.gl/d2xyf2
[4] https://www.modelica.org
[5] https://openmodelica.org
[6] https://jmodelica.org/
[7] https://www.python.org
[8] https://goo.gl/WpAMds

Julia packages *Plots.jl*, *LightGraph.jl*, *JuMP.jl*, etc.).

Some work on modeling a waterway for high head hydropower system together with a generator, a Francis turbine, and a governor, has already been carried out using OpenModelica (Vytvytskyi and Lie, 2017, 2018a). Unit models have been assembled in our in-house Modelica library *OpenHPL*[9]. Similarly to Julia, a Python API[10] for OpenModelica already exists and a use of this API for the *OpenHPL* is presented in (Vytvytskyi and Lie, 2018b).

1.3 Overview of Paper

In this paper, the main contribution is the prototyping and testing of automated structural analysis for dynamic systems in Julia using directed graphs from the *LightGraphs.jl* package.

The paper is structured as follows: Section 2 gives an overview of the graph and structural analysis theory. The Julia implementation of these analysis methods is discussed in Section 3. Applying the structural analysis methods on hydropower models is presented in Section 4. Finally, discussion and conclusions are given in Section 5.

2 Structural analysis

Structural analysis of models is the evaluation of model behavior base on a model structure. In this study, the model structure is represented by graphs. That is why, the graph theory is described first.

2.1 Graph theory

A graph G connects nodes (vertices, points) $N = \{n_1, n_2, \ldots, n_N\}$ via edges (lines) $E = \{e_1, e_2, \ldots, e_E\}$. Here, we will consider a *directed graph* (digraph); a digraph may be defined by a relation R consisting of a set of *ordered pairs* (n_i, n_j) with unidirectional information flow between these nodes.

As examples, Fig. 1 shows the undirected graph G_1 (left) and the directed graph G_2 (right). Observe that each pair corresponds to an edge; G_1 has 5 edges because its relation R_1 holds 5 (unordered) pairs, while G_2 has 6 edges because R_2 holds 6 (ordered) pairs.

Instead of describing the graph via a relation, it can be described via either an *adjacency* matrix A or an *incidence* matrix I. The incidence matrix description I with $\dim I = n_E \times n_N$ relates edges and nodes. The incidence matrix is not suitable for describing self edges, and is not discussed further here.

The adjacency matrix relates unidirectional flow between two nodes, and is defined by $A_{i,j} = 1$ for $(i,j) \in R$, or $A_{i,j} = 0$ for $(i,j) \neq R$ The adjacency matrix is square, $\dim A = n_N \times n_N$, with nodes represented by both rows and columns. Adjacency matrix A_2 for G_2 in Fig. 1 is $A_2 = \begin{smallmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{smallmatrix}$. Observe that nonzero diagonal elements

[9]Open Hydro Power Library is developed by the first author within his PhD study.

[10]https://goo.gl/Qyjqq2

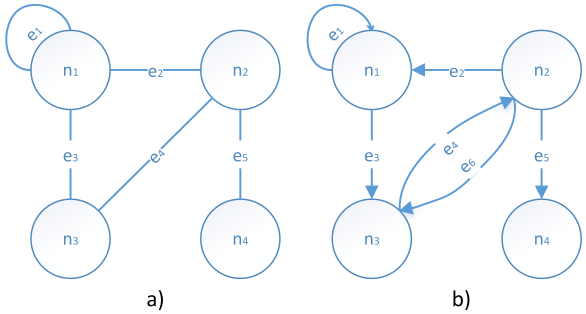

Figure 1. Examples of (a) undirected graph, and (b) directed graph.

in the adjacency matrix implies self edges, i.e., edges that emanates from the node and returns to the node.

An important concept in graph theory is the length ℓ between two nodes: the length is the number of nodes traversed to go from node n_i to node n_j. In G_2 of Fig. 1, the length in going from n_1 to n_1 is $\ell = 1$ because of the self edge. The length in going from n_1 to n_3 is $\ell = 1$ because there is an edge from n_1 to n_3. The length in going from n_1 to n_2 is $\ell = 2$: it is necessary to first go from n_1 to n_3, and then from n_3 to n_2.

The same principal can be used to represent a model structure of linear models with inputs and outputs, (Šiljak, 2011). These models can be represented by Eq. 1:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \tag{1}$$

Here, $x \in \mathbb{R}^{n_x}$ is the state, $u \in \mathbb{R}^{n_u}$ is the input/control signal, and $y \in \mathbb{R}^{n_y}$ is the output. $A \in \mathbb{R}^{n_x \times n_x}$, $B \in \mathbb{R}^{n_x \times n_u}$, $C \in \mathbb{R}^{n_y \times n_x}$ and $D \in \mathbb{R}^{n_y \times n_u}$ are constant matrices and consist of elements $a_{i,j}$, $b_{i,j}$, $c_{i,j}$ and $d_{i,j}$, respectively.

In order to represent a structure of this system, the interconnection square matrix M should be created, (Šiljak, 2007). This interconnection matrix M combines information from all the constant matrices, A, B, C, D, and represents the relationships between states, inputs and outputs. The matrix M is found as follows:

$$M = \begin{bmatrix} A & B & 0 \\ 0 & 0 & 0 \\ C & D & 0 \end{bmatrix} \tag{2}$$

In M, the second block row is zero because u_k is an input and not a response variable, while the third block column is zero because y_k is a response variable and not an input.

2.2 Structural controllability

In control theory, the mathematical duals observability and controllability are important properties of control systems. Using controllability, it is possible to evaluate the capability of the external input capability to influence the internal state. Observability, on the other side, gives an understanding of the possibility of a system state to be inferred from an external output.

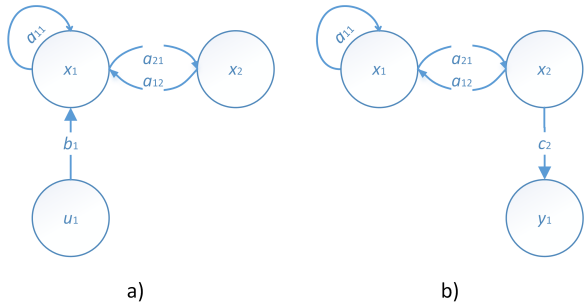

Figure 2. Examples of (a) structurally controllable, and (b) structurally observable systems.

As mentioned above, numerical problems can arise in the classical methods for observability and controllability computations in complex, large scale systems. As an alternative, structural observability and controllability are considered in this study due to the simplicity of these methods. In addition, structural observability and controllability provide necessary conditions for observability and controllability. This means that if the complex system is not structurally observable or not controllable, then it is not observable or not controllable. On the other hand, the system may be structurally observable/controllable, while in reality the system is not observable/controllable, (Šiljak, 2011).

Consider a linear system with an external input, see Eq. 1, e.g., $A = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right]$ and $B = \left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]$. First, the system structure based on the digraph G should be created with the interconnection matrix, $M = \left[\begin{smallmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{smallmatrix}\right]$, see Fig. 2 (a). Then, structural controllability of the system can be demonstrated if there is a directed path in the digraph G from (at least one) input-node to every single state-node. As seen in Fig. 2 (a), the system is structurally controllable, because there are paths from the input-node u_1 to the state-node x_1 with the edge b_1 and to the state x_2 with the edges b_1 and $a_{2,1}$.

2.3 Structural observability

Similarly to structural controllability, structural observability requires that there is a directed path from every single state-node to (at least one) output-node in the digraph G. Let us suppose a linear system with an external output, see Eq. 1, e.g., $A = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right]$ and $C = \left[\begin{smallmatrix} 0 & 1 \end{smallmatrix}\right]$. The interconnection matrix is then as follows, $M = \left[\begin{smallmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{smallmatrix}\right]$. The system structure based on the digraph G is created using the matrix M and is show in Fig. 2 (b). Here, the structural observability of the system is proven due to state-nodes x_1 and x_2 with directed paths ($a_{2,1}$,c_2) and (c_2) to the output node y_1, respectively.

2.4 Relative degree of system

Another property that can be found from structural analysis is the relative degree of the system, which shows how the input affects the system output. More precisely, the relative degree r represents the number of differentiations of the output y needed for the input u to appear, (Slotine and Li, 1991). From the digraph structure of the system, this relative degree can be found as the smallest number of state-nodes through which a directed path from an input-node to output-node goes.

Combining the two previous examples with one input and one output, e.g., $A = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right]$, $B = \left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]$ and $C = \left[\begin{smallmatrix} 0 & 1 \end{smallmatrix}\right]$, it can be shown that the relative degree of the system equals two, see Fig. 2 (a) and (b) together. This is because the path from the input-node u_1 to the output-node y_1 is (b_1,$a_{2,1}$,c_2) and this path goes through two state-nodes x_1 and x_2. This statement can be also shown by the condition for relative degree r from (Slotine and Li, 1991):

$$r = \min_{\rho} \left\{ L_g L_f^{\rho-1} h(x) \neq 0 \right\} \tag{3}$$

Here, we consider the system $\dot{x} = f(x) + g(x)u$ and $y = h(x)$. Symbols L_g and L_f are the Lie derivatives of $h(x)$ along $g(x)$ and $f(x)$, respectively, i.e., $L_g h(x) = \frac{\partial h(x)}{\partial x} g(x)$ and $L_f h(x) = \frac{\partial h(x)}{\partial x} f(x)$. Hence, in our example it is proven by Eq. 4 that the relative degree r equals two:

$$\begin{array}{lll} \text{for} & \rho = 1: & L_g h(x) = \left[\begin{smallmatrix} 0 & 1 \end{smallmatrix}\right] \left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right] = 0 \\ \text{for} & \rho = 2: & L_g L_f h(x) = \left[\begin{smallmatrix} 0 & 1 \end{smallmatrix}\right] \left[\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right] \left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right] = 1 \end{array} \tag{4}$$

In the case of multiple inputs or outputs, a set of relative degrees appears for each output-node. In such cases, in addition to this set of relative degrees, a total relative degree of the system is defined. The total relative degree is nothing but sum of the set of relative degrees, (Slotine and Li, 1991).

3 Julia implementation

For the prototype tools in this paper, a linear model in state space form as in Eq. 1 is assumed. Such a representation might be found in two ways. In one case, a dynamic system is modeled directly in Julia with *DifferentialEquations.jl* package, (Rackauckas and Nie, 2017), and then can be linearized using the *ForwardDiff.jl* package[11], (Revels et al., 2016). Alternatively, the model can be represented in Modelica, and OpenModelica with OMJulia can be used for the model linearization. The Julia API of OpenModelica with OMJulia is similar to the Python API of OpenModelica with OMPython which has been discussed in previous work, (Vytvytskyi and Lie, 2018b).

In order to work with graphs in Julia the *LightGraphs.jl* package[12] and *GraphPlot.jl* package[13] can be used for graph creation and plotting, respectively. In addition to these packages, other Julia packages for this study are also required, i.e., *Plots.jl*[14] and *DataFrames.jl*[15] packages.

[11] https://goo.gl/en5JMu
[12] https://goo.gl/tveMx1
[13] https://goo.gl/ifVw1p
[14] https://github.com/JuliaPlots/Plots.jl
[15] https://github.com/JuliaData/DataFrames.jl

It should be noted that examples with results of using all functions presented in this section, are given in Section 4.

3.1 Graphical structure of system

The *LightGraphs.jl* package[16] can be used to create a digraph of the linear system structure in Julia. Using interconnection matrix M (described in eq. 2) as an input to the *DiGraph()* command, the digraph G can be created and then plotted with the *gplot()* command from the *GraphPlot.jl* package[17]. An example, Julia code for creating and plotting the digraph of a simple three by three interconnection matrix M previously presented for the controllability example (see Fig. 2 (a)) looks as follows:

```
M = [ 1.0 1.0 1.0;
      1.0 0.0 0.0;
      0.0 0.0 0.0] // intercon. matrix
G = DiGraph(M) // create the digraph
gplot(G, layout=spring_layout,
    NODESIZE = 0.1,
    nodefillc=colorant"turquoise",
    NODELABELSIZE = 5,
    nodelabel=["x1","x2","u1"],
    nodelabelc = colorant"black",
    EDGELINEWIDTH=0.5,
    edgestrokec=colorant"grey",
    arrowlengthfrac=0.08,
    arrowangleoffset = pi/10)) // plotting
```

Here, plotting of the graph with *gplot()* command has various options:

- various possibilities for graph layout (random, circular, spring, shell, stressmajorize, and spectral layouts);
- setting size and color for nodes (*NODESIZE*, *nodefillc*), nodes' labels (*NODELABELSIZE*, *nodelabelc*), or edges (*EDGELINEWIDTH*, *edgestrokec*);
- specifications of nodes' labels names (*nodelabel*);
- setting edges' arrows shape (*arrowlengthfrac*, *arrowangleoffset*).

It should be noted that all color settings in the *gplot()* command might be specified with a vector of colors, one for each nodes/edges, similarly to the presented name vector for the nodes' labels. All the discussed options can be specified by the user according to their choice.

The results of running the presented code is shown in Fig. 3, where the simple digraph of three nodes is presented. Hence, using the presented commands for graphs creation and plotting, our own functions for system structure construction can be developed. The first function is named *obtain_graph_structure()* and provides digraph G together with interconnection matrix M and a data table, where the nodes' labels are structured with respect

[16] https://goo.gl/tveMx1
[17] https://goo.gl/ifVw1p

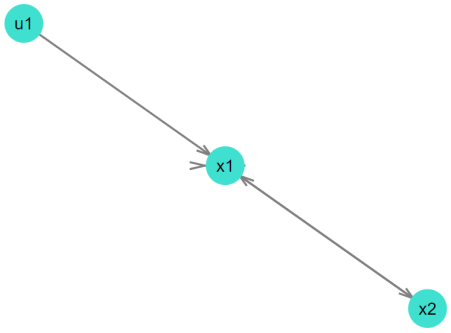

Figure 3. Digraph from the simple example of Julia code for the system in Fig. 2 (a).

to state/input/output names. This function also gives specific color arrays for nodes, nodes labels and edges. All nodes are colored according to their type, i.e., individual colors for states (turquoise), inputs (light blue), and outputs (light green). Edges in turn are colored based on their connection, i.e., individual color for self loops (red), state interactions (grey), connections from input (blue), and to output (brown). The inputs to this function are the constant system matrices A, B, C, and D. Three string arrays with the variables' names of state, input and output are inputs as well.

An example of the *obtain_graph_structure()* function calling is provided below. Here, commands for displaying of the graph G and the data table df are also used. As it is seen, the *gplot()* command for the graph plotting is specified with different options found with the *obtain_graph_structure()* for the names of nodes and color vectors for nodes, labels and edges.

```
G,M,df,Node_c,Edg_c,Nodelable_c,Nodelble =
    obtain_graph_structure(A,B,C,D,
    StateName,InputName,OutputName);
println(df) // display the data table
gplot(G, layout=circular_layout,
    NODESIZE = 0.05,
    NODELABELSIZE = 5,
    nodefillc=Node_c,
    EDGELINEWIDTH=0.3,
    edgestrokec=Edg_c,
    nodelabel=Nodelble,
    nodelabelc = Nodelable_c,
    arrowlengthfrac=0.08,
    arrowangleoffset = pi/10)
```

In cases when the user does not want to display (print/-plot) the results, another function *system_structure()* can be used. The function gives the possibility to show the structure of the system directly after execution. The use of this function is provided below:

```
system_structure(A,B,C,D,
    StateName,InputName,OutputName)
```

3.2 Structural observability and controllability

As presented above, structural digraph paths should be checked in order to show the structural observability or controllability of the system. To be structurally observable, there should be a directed path from every state-node to at least one output-node. Similarly, there should be a directed path from at least one input-node to every state-node in order to be structurally controllable.

Functions for checking structural observability and controllability of the system have been developed: *check_sys_observ()* and *check_sys_control()*, respectively. The calling of these functions are the same as for the functions presented in the previous subsection, and an example is given below:

```
check_sys_observ(A,B,C,D,
    StateName,InputName,OutputName)

check_sys_control(A,B,C,D,
    StateName,InputName,OutputName)
```

Both functions operate in similar way, and in the case that all states of the system are structurally observable/-controllable, they return a message with the following text: "All states are structurally observable/controllable". Otherwise, these functions provide information of which states are structurally unobservable/uncontrollable. In both cases, the functions also display the digraph with a structure of the system. In addition, in the case with some unobservable/uncontrollable states, some transparency colors are used to display these state-nodes and the edges connected to these nodes.

In some cases, it might be of interest to specifically check some of the system states for observability or controllability. Because of this, another two functions that check structural observability/controllability of specified states are developed. The use of these functions are similar to the previous two functions, but here the user should also specify the state that will be checked. The state number from the node's label (*state_num*) is used for this specification. An example looks as follows:

```
check_state_observ(A,B,C,D,StateName,
    InputName,OutputName,state_num)
check_state_control(A,B,C,D,StateName,
    InputName,OutputName,state_num)
```

Both functions return a message that shows if the specified state is structurally observable/controllable or not. They also display the digraph with a structure of the system where the specified state-node and a path (edges and nodes) which shows its structural observability/controllability are highlighted. Colors of all other nodes and edges are somewhat transparent.

3.3 Relative degree of system

In order to determine the relative degree of the system presented by digraph G, a smallest number of state-nodes should be found through which a directed path from an input-node to output-node goes. For this task, the *sys_relative_degree()* function is developed. This function defines the relative degree of the system and then returns a message that shows the value of the defined relative degree. For a system with multiple inputs and outputs, information about the total relative degree is provided together with a set of the relative degrees of all outputs. Moreover, a digraph is displayed with the structure of the system. In this digraph, colors of all nodes and edges are a bit muted, except for the path/paths (edges and nodes) that is/are the basis for the relative degree.

The use of the function for checking the relative degree looks as follows:

```
sys_relative_degree(A,B,C,D,
    StateName,InputName,OutputName)
```

4 Results

The various hydropower models that are implemented in OpenModelica using our in-house hydropower library, *OpenHPL*, are used here for testing of the developed functions for system structural analysis. Description and information about these hydropower models already have been presented previously, (Vytvytskyi and Lie, 2017, 2018b). For use with the structural analysis code, these models are first linearized in Julia using package OMJulia for OpenModelica. The constant A, B, C, and D matrices for the linearized hydropower state space models together with ordered lists (vectors) of state, input and output names are then used for structural analysis.

4.1 Simple waterway model

First, a simple model of the hydropower system with basic models for the waterway (incompressible water and inelastic pipes, (Vytvytskyi and Lie, 2017)) is used. This model consists of 5 states and has one input and one output. The *system_structure()* function provides the model structure, see Fig. 4. Here, the states ($x_1 - x_5$) are colored turquoise and consist of the volumetric flow rates in the penstock and surge tank, and the water masses in the surge tank, reservoir, and tail water. The input (u_1) is the control signal for the turbine and is colored light blue. The output (y_1) is colored light green and represents the flow rate in the turbine which is the same as the penstock flow rate in this model. Figure 4 shows the digraph with the model structure using the circular layout for the graph plotting. This can be changed to another style in options to the *gplot()* command.

Next, the hydropower model can be checked for structural observability and controllability using *check_sys_observ()* and *check_sys_control()* commands. The results for these studies are shown in Fig. 5 for observability and in Fig. 6 for controllability. It is seen from Fig. 5 that the system is structurally observable because all system states transmit information through digraphs to the output. In the same way, there are two uncontrollable states which make system structurally

7×4 DataFrames.DataFrame

Row	NodeLabel	State	Input	Output
1	x1	penstock.V_dot		
2	x2	reservoir.m		
3	x3	surgeTank.V_dot		
4	x4	surgeTank.m		
5	x5	tail.m		
6	u1		u	
7	y1			dotV

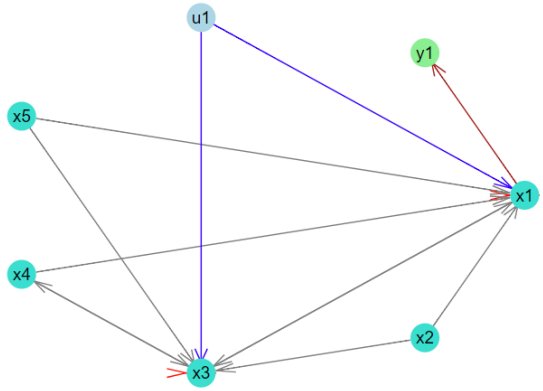

Figure 4. The digraph with the simple model structure determined by the *system_structure()* function.

All states are structurally observable

7×4 DataFrames.DataFrame

Row	NodeLabel	State	Input	Output
1	x1	penstock.V_dot		
2	x2	reservoir.m		
3	x3	surgeTank.V_dot		
4	x4	surgeTank.m		
5	x5	tail.m		
6	u1		u	
7	y1			dotV

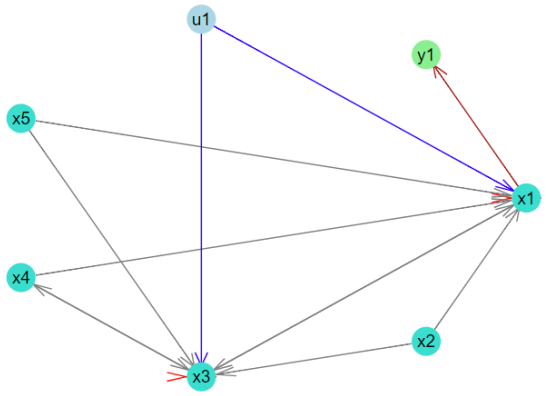

Figure 5. The results of checking the structural observability for simple model by the *check_sys_observ()* function.

uncontrollable, see Fig. 6. This uncontrollability for the water masses in the reservoir and tail water is caused by model simplification: these masses are kept constant in the model, (Vytvytskyi and Lie, 2018b); the uncontrollability is thus fictitious in this case.

The relative degree of this simple hydropower model can be found using the developed *sys_relative_degree()* function. The result of running this command for the simple model is shown in Fig. 7. Here, it is seen that the relative degree, r, equals one, which means that the control signal directly affects a state that influences the output.

4.2 Detailed waterway model

A more detailed model of the hydropower system is used next. This model is similar to the previous simple model, but here the penstock unit is described by a more detailed pipe model instead of the basic pipe model (here, compressible water and elastic pipes are considered in the penstock, (Vytvytskyi and Lie, 2017)). This model consists of 24 states and also has one input and one output. The result of the *system_structure()* function provides the model structure, see Fig. 8. Here, the states $(x_1 - x_{24})$ consist of the pressures $(U[1,..,10])$ and mass flow rates $(U[11,..,20])$ in the penstock segments, volumetric flow rate in the surge tank, and the water masses in the surge tank, reservoir, and tail water. The input (u_1) is the control signal for the turbine and the output (y_1) is the flow rate through the turbine. The state, input, and output nodes are colored in the same way as previously. It is seen from Fig. 8 that for more complex systems (more nodes), it becomes harder to observe visually how the nodes are connected. One way to study the system structure is to decompose the system in smaller subsystems. This can easily be

The uncontrollable states are: reservoir.m, tail.m

7×4 DataFrames.DataFrame

Row	NodeLabel	State	Input	Output
1	x1	penstock.V_dot		
2	x2	reservoir.m		
3	x3	surgeTank.V_dot		
4	x4	surgeTank.m		
5	x5	tail.m		
6	u1		u	
7	y1			dotV

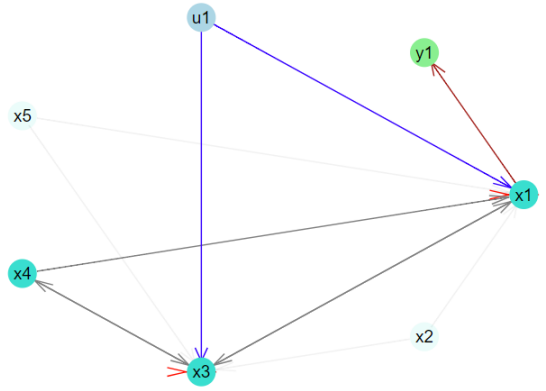

Figure 6. The results of checking the structural controllability for simple model by the *check_sys_control()* function.

```
Relative degree of the system is: r = 1
```

7×4 DataFrames.DataFrame

Row	NodeLabel	State	Input	Output
1	x1	penstock.V_dot		
2	x2	reservoir.m		
3	x3	surgeTank.V_dot		
4	x4	surgeTank.m		
5	x5	tail.m		
6	u1		u	
7	y1			dotV

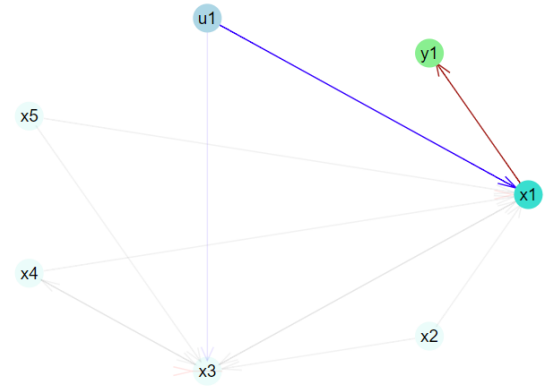

Figure 7. The results of checking the relative degree for simple model by the *sys_relative_degree()* function.

26×4 DataFrames.DataFrame

Row	NodeLabel	State	Input	Output
1	x1	penstock.U[1]		
2	x2	penstock.U[2]		
3	x3	penstock.U[3]		
4	x4	penstock.U[4]		
5	x5	penstock.U[5]		
6	x6	penstock.U[6]		
7	x7	penstock.U[7]		
8	x8	penstock.U[8]		
⋮				
18	x18	penstock.U[18]		
19	x19	penstock.U[19]		
20	x20	penstock.U[20]		
21	x21	reservoir.m		
22	x22	surgeTank.V_dot		
23	x23	surgeTank.m		
24	x24	tail.m		
25	u1		u	
26	y1			dotV

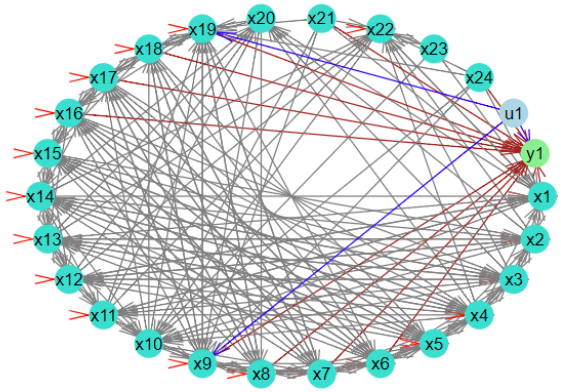

Figure 8. The digraph with the detailed model structure determined by the *system_structure()* function.

done by picking up the appropriate rows and columns in the A, B, and C matrices with respect to interested states.

On the other hand, the structural analysis for observability, controllability, and relative degree of the system may be still performed for the complete model. This can be done by running the same functions for the detailed model: *check_sys_observ()* — for observability, *check_sys_control()* — for controllability, and *sys_relative_degree()* — for relative degree. The results of these functions are not presented here to save space. However, the resulting information is as follows:

- "All states are structurally observable".
- "The uncontrollable states are: reservoir.m, tail.m", similarly to the case with simple waterway model.
- "Relative degree of the system is: r = 0", the input signal directly affects the output, i.e., the constant matrix D is not zero.

4.3 Simple waterway model with generator

Here, the simple model presented above is studied with a model of a synchronous generator that is connected to the grid. The models of this electrical part (generator, grid, etc.) are taken from the *OpenIPSL*[18] library, and is used in OpenModelica. OpenIPSL is the Open-Instance Power System Library, where a wide variety of power system components are available. The model of the simple hydropower waterway and generator consists of 7 states and has one input and 3 outputs. Here, the states ($x_1 - x_7$) consist of the generator shift angle and angular velocity, the volumetric flow rates in the penstock and surge tank, and the water masses in the surge tank, reservoir and tail water. The input (u_1) is the control signal for the turbine and the outputs ($y_1 - y_3$) are the generator power production and angular velocity, and flow rate through the turbine. The state, input and output nodes are colored in the same way as previously.

The result of the *system_structure()* function provides the model structure, Fig. 9. This structure is presented with the digraph of another layout type (spring layout), in order to demonstrate another structural view. The structural analysis for observability, controllability and relative degree of the system is also performed for this model case. This can be done by running the same functions as for the detailed model: *check_sys_observ()* — for observability, *check_sys_control()* — for controllability, and *sys_relative_degree()* — for relative degree. The results of executing these functions are not shown here, but the results are summarized as follows:

- "All states are structurally observable".
- "The uncontrollable states are: reservoir.m, tail.m", similarly to the two previous cases.

[18] https://openipsl.readthedocs.io/en/latest/

11×4 DataFrames.DataFrame

Row	NodeLabel	State	Input	Output
1	x1	order2_1.delta		
2	x2	order2_1.w		
3	x3	penstock.V_dot		
4	x4	reservoir.m		
5	x5	surgeTank.V_dot		
6	x6	surgeTank.m		
7	x7	tail.m		
8	u1		u	
9	y1			P
10	y2			dotV
11	y3			w

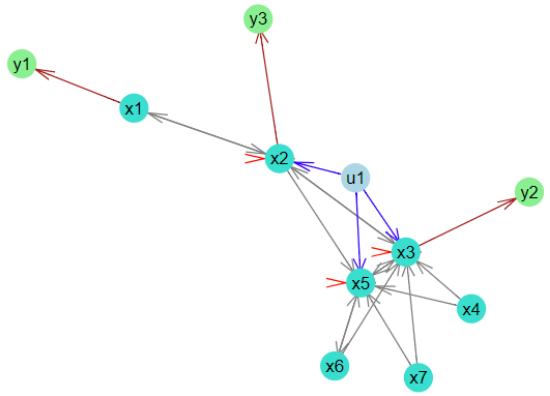

Figure 9. The digraph of the model structure determined by the *system_structure()* function. The model of the simple hydropower waterway and generator is used.

- "The system have relative degree (2, 1, 1). Total relative degree of the system is: r = 4": here is shown first the relative degrees for each output and then the total relative degree of the system.

5 Discussion and Conclusions

This paper has explored the possibilities of using graph theory methods for structural analysis of dynamic system. Although the chosen examples hardly qualify as complex/large scale, graph methods scale well to huge systems. The presented methods have been implemented in Julia using the *LightGraphs.jl* and *GraphPlot.jl* packages. Using the *OpenHPL* hydropower library in OpenModelica and OMJulia for OpenModelica, the structural analysis methods have been tested on hydropower models of different complexity.

The results of testing the developed structural analysis functions look reasonable and can be further used for analysis related to state estimation and control: observability is a requirement for state estimators to work properly, controllability is required for control design, and relative degree is important in the design of nonlinear feedback controllers. One experience with the developed tools is that sometimes it can be hard to make a good visualization of the graph structure of complex (large scale) system. It can be hard to see the whole picture of the system structure (small subsystems are not easily seen) using the circular layout for the graph plotting. However, the user can do some testing of different layout types for the graph plotting to find the most appropriate one. Moreover, the graph can be stored in a picture with higher resolution and bigger size that can help to see the system structure in a better way. In addition, developers of *LightGraphs.jl* and *GraphPlot.jl* packages are planning to improve the plotting possibilities of graphs in future, e.g., to improve the display self loop edges, etc.

In summary, this paper has explored some possibilities with structural analysis. Further work should be put into streamlining the functions into a package, with better use of Julia coding conventions, integration with other modeling tools, integration with control packages, etc.

References

S. Boyd and L. Vandenberghe. *Introduction to Applied Linear Algebra*. Cambridge University Pr., 2018. ISBN 1316518965.

N. Deo. *Graph Theory with Applications to Engineering and Computer Science*. Dover Publications, 2017. ISBN 9780486820811.

J. Lunze. *Feedback control of large scale systems*. Prentice-Hall international series in systems and control engineering. Prentice-Hall, 1992. ISBN 9780133183535.

Magamage Anushka Sampath Perera. *State estimation and optimal control of an industrial copper electrowinning*. PhD thesis, University College of Southeast Norway, Faculty of Technology, Kongsberg, Norway, 2016.

C. Rackauckas and Q. Nie. Differentialequations.jl – a performant and feature-rich ecosystem for solving differential equations in julia. *Journal of Open Research Software*, 5(1), 2017. doi:10.5334/jors.151.

J. Revels, M. Lubin, and T. Papamarkou. Forward-mode automatic differentiation in julia. *arXiv:1607.07892 [cs.MS]*, 2016. URL `https://arxiv.org/abs/1607.07892`.

Dan Simon. *Optimal State Estimation: Kalman, H∞, and Nonlinear Approaches*. John Wiley & Sons, 2006.

Jean-Jacques E Slotine and W. Li. *Applied nonlinear control*. Prentice-Hall International Editions. Prentice-Hall, Englewood Cliffs, N.J, 1991.

D. D. Šiljak. *Large-Scale Dynamic Systems: Stability and Structure*. Dover Civil and Mechanical Engineering Series. Dover Publications, 2007. ISBN 9780486462851.

D. D. Šiljak. *Decentralized Control of Complex Systems*. Dover Books on Electrical Engineering Series. Dover Publications, 2011. ISBN 9780486486147.

L. Vytvytskyi and B. Lie. Comparison of elastic vs. inelastic penstock model using OpenModelica. In *Proceedings of the 58th Conference on Simulation and Modelling (SIMS 58) Reykjavik, Iceland, September 25th–27th, 2017*, number 138, pages 20–28. Linköping University Electronic Press, Linköpings Universitet, 2017. doi:10.3384/ecp1713820.

L. Vytvytskyi and B. Lie. Mechanistic model for Francis turbines in OpenModelica. *IFAC-PapersOnLine*, 51(2):103 – 108, 2018a. doi:10.1016/j.ifacol.2018.03.018.

L. Vytvytskyi and B. Lie. Linearization for Analysis of a Hydropower Model using Python API for OpenModelica. In *Proceedings of the 59th Conference on Simulation and Modelling (SIMS 59) Oslo, Norway, September 26th–28th, 2018*, 2018b.