DSAnalysis User Manual

Philippe De Rua

December 1, 2023

Funding
This document and the DSAnalysis Toolbox it describes were realised as part of the PhD studies of Philippe De Rua, funded by the Research Foundation – Flanders (FWO) under Grant No. 11E8720N.

Citing this work
If DSAnalysis is used in a scientific work, reference should be made to the following publication: P. De Rua, *Model Transformations and Periodic Trajectory Calculations for Stability Assessments of Modular Multilevel Converter-Based Systems.* PhD thesis, Department of Electrical Engineering, KU Leuven, 2023

Contact details
Prof. Jef Beerten
Electa Research Group
Department of Electrical Engineering (ESAT), KU Leuven
Kasteelpark Arenberg 10, 3001 Leuven, Belgium

Contents

Chapter 1

User Manual

1.1 Introduction

DSAnalysis[1] is a Matlab toolbox supporting the *analysis of dynamic systems*. The current version supports:

- Definition of dynamic systems by means of analytical differential equations in state-space form, their numerical parameters and inputs (control references and external sources);
- Calculation of periodic trajectories or equilibria;
- Symbolic linearisation of the analytical differential equations in state-space form;
- Transformation of linear time-periodic systems into linear time-invariant systems by means of frequency-lifting;
- Calculation of eigenvalues/characteristic roots, enabling small-signal modal analysis (including participation factors);
- Calculation of frequency responses, enabling frequency-domain stability assessments;
- Numerical integration of the differential equations, i.e. time-domain simulation.

Additionally,

- The toolbox offers separation between numerical data and analytical equations;
- The toolbox relies on a set of files supporting the different types of calculations in a generic manner;
- The toolbox offers an easy access to all parameters, options and results within a unique data structure.
- The toolbox supports the definition of continuous-time systems with or without delays. Switched systems and discrete-time systems are not covered in this version.

[1] Or DSA in short, for *Dynamic System Analysis*

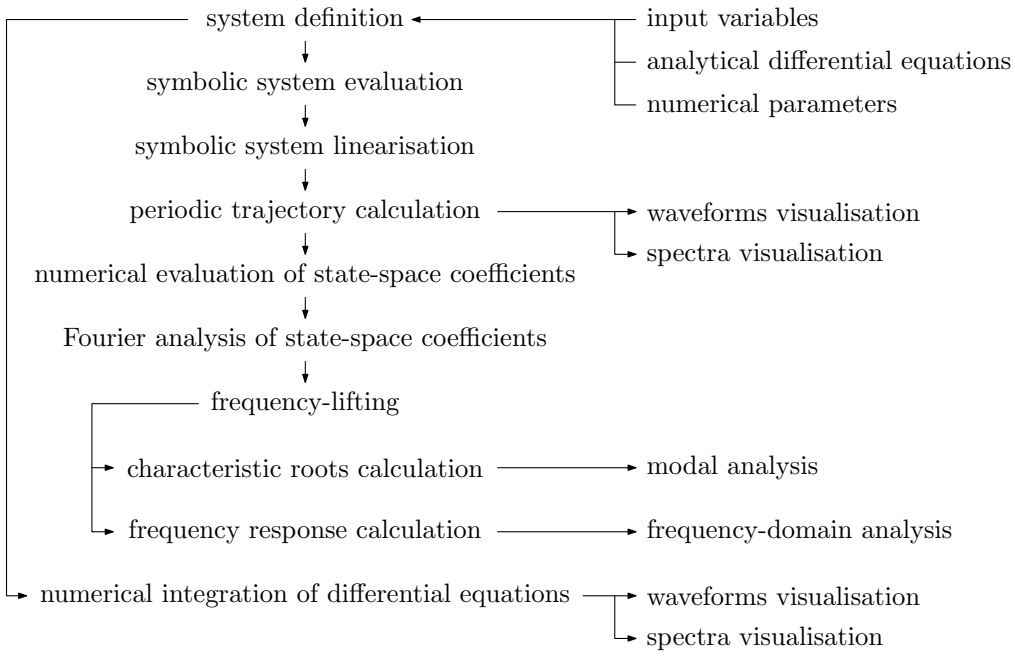

Figure 1.1: Structure of the DSA toolbox

1.2 Getting started

1. You need Matlab[2] with the Optimization Toolbox and the Symbolic Math Toolbox.
2. Extract the DSAnalysis folder from its .zip.
3. Set the DSAnalysis folder as the current Matlab path.
4. Open the `DSA_main` file and you are ready to run an example.

In the remainder of this document,

- Section 1.3 describes the different files and folders necessary to represent a dynamic system in DSAnalysis.
- Section 1.4 describes the structure of the code, the main data structure in which all parameters, options and results are stored.
- Section 1.5 describes the available Matlab commands used to analyse the dynamic systems, and provides some theoretical background and references wherever applicable.

[2] The current version of the DSAnalysis toolbox has been tested on Matlab release 2021b.

1.3 Files and folders

1.3.1 User-defined functions: how to define your own systems and parameters

The dynamic systems considered in this code are characterised by analytical differential equations, numerical parameters and a list of variables. More precisely:

1. **Analytical differential equations** in state-space form:

   - without pure delays:

$$\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{1.1a}$$

$$\boldsymbol{y}(t) = \boldsymbol{h}(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{1.1b}$$

   - with pure delays:

$$\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{w}(t)) \tag{1.2a}$$

$$\boldsymbol{z}(t) = \boldsymbol{g}(t, \boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{w}(t)) \tag{1.2b}$$

$$\boldsymbol{y}(t) = \boldsymbol{h}(t, \boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{w}(t)) \tag{1.2c}$$

$$\boldsymbol{w}(t) = \boldsymbol{z}(t - T_d) \tag{1.2d}$$

   In this document, $\boldsymbol{x}$ is the vector of states, $\boldsymbol{u}$ is the vector of inputs, $\boldsymbol{y}$ is the vector of outputs, $\boldsymbol{z}$ is the vector of to-delay variables.

2. **Parameters**, i.e. numerical values used in the differential equations, for instance

   - circuit parameters;
   - control parameters;
   - input parameters;
   - time delays (if any);
   - ...

3. **A list** of input variables, state variables, output variables, and delay variables (if any).

All of the above is defined in a set of user-defined functions and scripts:

function	type	description
`sstm_DATA(DSA)`	function	defines the numerical data, i.e. all numerical values of the parameters.
`sstm_inst_VARS(DSA)`	function	defines the lists of inputs, states, outputs and delay variables (if any).
`sstm_inst_EQNS`	script	defines the dynamic equations.
`sstm_inst_HD(DSA, HDi)`	function	defines the harmonic spectrum of inputs. Additionally, it defines an initial guess for the harmonic spectrum of the states and delayed variables, a priori unknown.
`sstm_inst_uOft(t, DSA)`	function	defines of time-domain waveforms of the inputs. It is equivalent to `sstm_inst_HD` but is dedicated to time-domain simulations.

Table 1.1: user-defined functions and scripts

`VARS` stands for *variables*, `EQNS` for *equations*, `HD` for *harmonic data* or *harmonic domain*, `uOft` for *u of t*, i.e. the inputs provided as a function of time.

note

The code revolves around a Matlab structure named `DSA`, which contains all numerical data, code options, calculation results, etc. Its fields are presented in more details in the next sections.

note

When implementing dynamic systems, users can define *systems* and *instances* of those systems. This is a way of partially avoiding code duplication. In the code, a *system* is referred to as `sstm` and an *instance* is referred to as `inst`, where `sstm` and `inst` are replaced with their actual names.

example

Consider that we want to study the dynamics of a power-electronic converter, for instance the modular multilevel converter. We refer to this system as "MMC". Let us further consider that we want to compare two controllers, e.g. a circulating current controller, referred to as "CCC", and an arm energy controller, referred to as "AEC". These alternatives can be seen as two instances of the same MMC system. Although quite similar, they have different sets of analytical differential equations and different sets of variables, so they are described with their own files: the MMC with CCC is described with `MMC_CCC_VARS` for the variables and `MMC_CCC_EQNS` for the equations. The MMC with AEC is described with `MMC_AEC_VARS` for the variables and `MMC_AEC_EQNS` for the equations. However, both instances have practically the same set of parameters, which should ideally not be duplicated to simplify the user's workflow. Consequently, all instances of the MMC may share the same set of numerical parameters, which defined in function `MMC_DATA`.

tip

Using several instances under one system is convenient but only optional.

warning

The naming convention is important:

- Both `sstm` and `inst` variables must be defined in the main script (`DSA_main`);
- The file names rely on `sstm` and `inst`, and must be coherent with the variables defined in the main script.

1.3.2 Folders: where to save user-defined files

The code will find the user-defined files by itself as long as they are stored in the correct folders with the correct names. User-defined files are stored in the `.\sstms` folder.

- File `sstm_DATA` related to system `sstm` is stored in the folder `.\sstms\sstm`, where `sstm` is replaced with its actual name. For example, `MMC_DATA` is stored in `.\sstms\MMC`.
- All other files corresponding to system `sstm` and instance `inst` are stored in the folder `.\sstms\sstm\inst`, where `sstm` and `inst` are replaced with their actual names. For example, `MMC_CCC_VARS` and `MMC_CCC_EQNS` are stored in `.\sstms\MMC\CCC`, see also Fig. 1.2.

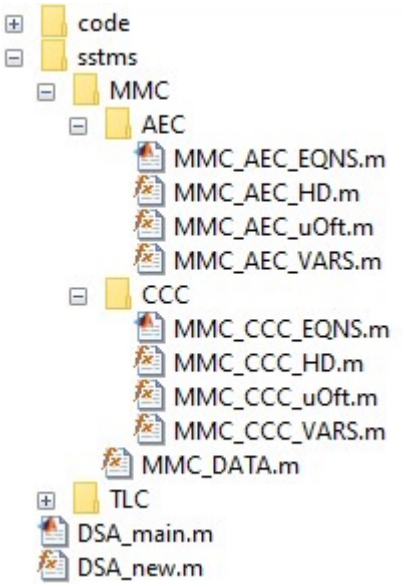

Figure 1.2: Example of folder tree: There are two different systems: MMC and TLC. The MMC has two instances: CCC and AEC. These two instances share the same DATA file, named `MMC_DATA`.

1.4 Data management: the DSA structure

As mentioned, all information related to data, equations, options, calculations, results, etc. is stored in a Matlab structure called "DSA". Its content looks like this:

			link
DSA	`sstm`	system name	1.4.1
	`inst`	instance name	1.4.1
	`info`	system information	1.4.2
	`data`	numerical data	1.4.3
	`vars`	variables	1.4.4
	`NLSS`	nonlinear system	1.4.5
	`LNSS`	linearised system	1.4.6
	`fcts`	function handles	1.4.7
	`calc`	supporting calculation data	1.4.8
	`COLL`	collocation method	1.4.9
	`TDSIM`	time-domain simulations	1.4.10
	`FLIFT`	frequency-lifting	1.4.13
	`HD`	harmonic-domain results	1.4.11
	`TD`	time-domain results	1.4.11
	`LTP`	linearised time-periodic system	1.4.12
	`LTI`	linearised time-invariant system	1.4.14
	`FRESP`	frequency responses	1.4.15

Table 1.2: DSA structure

The fields are added by calling specific functions, depending on the type of calculations being performed. The next paragraphs are dedicated to describing the fields of the DSA structure.

1.4.1 System identifiers

System identifiers are stored directly in the `DSA` structure.

field	description	type	example
`sstm`	name of the system being studied.	string	`'MMC'`
`inst`	name of the instance of the system being studied	string	`'CCC'`

Table 1.3: sstm and inst

1.4.2 System information

The `DSA.info` field contains information about the original (i.e. non-lifted) system dimensions. Every field of `DSA.info` is automatically initialised or updated when calling functions `DSA_new` or `DSA_set`.

tip

Lower-case letters refer to the dimensions of the small-signal system representation (after linearisation). Upper-case letters refer to the dimensions of the original large-signal system representation (before linearisation). Normally, only the number of inputs and outputs changes between large- and small-signal representations, when input-output models are determined between a subset of input variables to a subset of output variables. The number of states and delayed variables does not normally change.

field	field	description	type	example
`info`	fields related to large-signal systems			
	`N`	number of states	int	
	`M`	number of inputs	int	
	`P`	number of outputs	int	
	`D`	number of pure-delayed variables.	int	
	fields related to small-signal systems			
	`n`	number of states	int	
	`m`	number of inputs	int	
	`p`	number of outputs	int	
	`d`	number of pure-delayed variables	int	
	other fields			
	`path_sstm`	system path	string	`'.\sstms\MMC'`
	`path_inst`	instance path	string	`'.\sstms\MMC\CCC'`

Table 1.4: DSA.info

1.4.3 Numerical data

The `DSA.data` field is filled in via the user-defined `sstm_DATA` function and contains the system's numerical values for parameters and delays (if any).

field	field	description	type	example
data	f_1	the fundamental frequency of the system under study, in Hz.	int	50
	param	contains all the numerical values of the parameters involved in the differential equations. For example, the line param.K_p = 4.2 assigns the value 4.2 to parameter K_p. Then, K_p can be used symbolically in the analytical algebraic and differential equations defined in the sstm_inst_EQNS file, without having to input it as a numerical value directly in the differential equations. Some parameters defined in param may be unused by some instances, in particular when several instances use the same dataset.	struct	
	delay	contains all information related to delays of the system. For systems **without** delays, this structure does not have to be defined by the user. For system **with** delays, this structure must contain the delay values individually assigned to the to-delay variables. For instance, if variable n_ref is delayed by 0.2 seconds, then delay.n_ref=0.2. If one variable is subjected to more than one delay value, then it must be duplicated with a different name.	struct	
	base	(optional) may contain all data related to the scaling system, if a scaling system is used. This field is optional and the code works without it. In general, it is recommended to normalise/scale the dynamic system, which gives it better numerical properties. It may happen that the solvers do not find a periodic trajectory simply because the system is not properly scaled. Time-domain simulations are also slower for non-normalised systems of equations.	struct	

Table 1.5: DSA.data

1.4.4 Variables

The DSA.vars field contains all information related to the system variables. The fields are set up via the user-defined sstm_inst_VARS function. There are two main categories of variables:

- the large-signal (= *lasi*) variables, i.e. the variables related to the original nonlinear dynamic system.
- the small-signal (= *smsi*) variables, i.e. the variables related to the linearised versions of the system.

The main reason for using these two sets of variables is that, when linearising the system, we are sometimes only interested in the dynamic relationship from a subset of inputs to a subset of outputs. Mostly when frequency-lifting is used, it is interesting to limit the number of inputs and outputs to what is strictly necessary. Extra inputs and outputs are thus disregarded in the linearised models.

field	field	description	type
vars	fields related to large-signal variables		
	`lasi_states`	all state variables (`x`).	cell
	`lasi_inputs`	all input variables (`u`). As a general rule, input variables encompass control inputs and references, as well as external sources. Input variables are susceptible to vary with time, unlike parameters, which are often kept constant during a time-domain simulation. However, some constant input variables can technically be defined as parameters. Likewise, parameters that are susceptible to change value during the course of a simulation can be defined as inputs. Note that the code is only able to calculate transfer functions and frequency responses between a given input and a given output if the input is defined as a variable, not if it is defined as a numerical parameter.	cell
	`lasi_outputs`	all output variables (`y`). Is basically defined as output any intermediate or output variable that the user would like to display as waveforms or harmonic spectra after a periodic trajectory calculation or a time-domain simulation. The expression of variables listed as outputs must be given in the `sstm_inst_EQNS` file.	cell
	`lasi_todelay`	when delays are included, the code must establish the relationship between delayed variables (`w`) and their *not-yet-delayed* versions (`z`). Those *not-yet-delayed* variables are referred to as *to-delay* variables and are listed in this cell. For systems without delays, this cell is left empty. However, it must be defined for the code to work properly and automatically determine when there are no delays.	cell
	`lasi_delayed`	when delays are included, this cell contains the names of the delayed versions (`w`) of the variables. For systems without delays, this cell is left empty. However, it must be defined for the code to work properly and automatically determine when there are no delays.	cell
	fields related to small-signal variables		
	`smsi_inputs`	normally a subset of `lasi_inputs`.	cell
	`smsi_outputs`	normally a subset of `lasi_outputs`.	cell
	`smsi_states`	automatically set equal to `lasi_states`: the states does not normally change during linearisation.	cell
	`smsi_todelay`	automatically set equal to `lasi_todelay`: the delay variables do not normally change during linearisation.	cell
	`smsi_delayed`	automatically set equal to `lasi_delayed`: the delay variables do not normally change during linearisation.	cell
	other fields		
	`lasi_all`	internal variable combining all lasi variables	cell
	`lidx`	internal variable providing the list index of the variables	struct

Table 1.6: DSA.vars

note

lasi means large-signal; *smsi* means small-signal. Not to be confused with *sdst*, which means steady-state, nor *ss*, which means state-space.

1.4.5 Nonlinear system: symbolic representation

The `DSA.NLSS`[3] field contains a symbolic representation of the original (usually nonlinear) system. The symbolic representation is built from the analytical algebraic and differential equations in `sstm_inst_EQNS` by calling function `DSA = DSA_get_NLSS(DSA)`, which then stores the symbolic expressions in `DSA.NLSS`.

field	field	field	description	type
NLSS	dxdt		contains the symbolic expression of the time derivatives of the states, i.e. `f(x,u)` or `f(x,u,w)`.	symbolic vector
	lasi	z	symbolic expression of the to-delay variables, i.e. `g(x,u,w)`.	symbolic vector
	smsi	z	same as `lasi.z`. Note that these expressions are still nonlinear!	symbolic vector
		y	symbolic expression of the smsi outputs, i.e. `h(x,u)` or `h(x,u,w)`. Note that these expressions are still nonlinear!	symbolic vector

Table 1.7: DSA.NLSS

The expressions of the to-delay variables and of the outputs are specified under the `smsi` field, as they are intended to be linearised, however they cover only the subset of requested small-signal outputs.

1.4.6 Linearised state-space system symbolic representation

The `DSA.LNSS`[4] field contains a symbolic representation of the linearised system. The symbolic representation is built from the symbolic equations in `DSA.NLSS` by calling function `DSA = DSA_linearise(DSA)`, which then stores the matrix coefficients of the linearised equations in `DSA.LNSS.mat`.

[3] `NLSS` stands for *nonlinear state-space*.

[4] `LNSS` stands for *linear state-space*, not to be confused with `DSA.NLSS`.

field	field	field	description	type
LNSS	mat	fields related to systems with no delays		
		A	dimensions $n \times n$	symbolic matrix
		B	dimensions $n \times m$	symbolic matrix
		C	dimensions $p \times n$	symbolic matrix
		D	dimensions $p \times m$	symbolic matrix
		fields related to systems with delays		
		A	dimensions $n \times n$	symbolic matrix
		B1	dimensions $n \times m$	symbolic matrix
		B2	dimensions $n \times d$	symbolic matrix
		C1	dimensions $p \times n$	symbolic matrix
		C2	dimensions $d \times n$	symbolic matrix
		D11	dimensions $p \times m$	symbolic matrix
		D12	dimensions $p \times d$	symbolic matrix
		D21	dimensions $d \times m$	symbolic matrix
		D22	dimensions $d \times d$	symbolic matrix

Table 1.8: DSA.LNSS

1.4.7 Function handles

Field `DSA.fcts` contains several function and script handles. All fields are automatically set up via the `DSA_set` function. Some handles call user-defined functions and scripts and thus make a link between the generic code and the user-defined systems.

field	field	description	type
fcts	DATA	calls `sstm_DATA`	function handle
	VARS	calls `sstm_inst_VARS`	function handle
	HD	calls `sstm_inst_HD`	function handle
	uOft	calls `sstm_inst_uOft`	function handle
	EQNS	runs `sstm_inst_EQNS`	script handle
	COLL_JAC	runs `sstm_inst_COLL_JAC`	script handle
	h2i_h_m	maps harmonic index h in $(-h_m, ..., h_m)$ to vector index i in $(1, ..., n_{h_m})$ i.e. `@(h)h+h_m+1`	function handle
	open	opens the user-defined files related to the current system and instance	function handle
	save	saves the current DSA structure, with all its fields as such, at the location of `path_inst`	function handle

Table 1.9: DSA.fcts

1.4.8 Useful automatic pre-calculations

Field `DSA.calc` contains the results of some useful precalculations related to the solvers. All fields are automatically set up via the `DSA_set` function and should technically not be changed manually (i.e. directly) by the user.

field	field	description	type	example
calc	f_b	base frequency in Hz, as defined in DSA.data, Section 1.4.3.	int	50
	omega_b	base angular frequency in rad/s, i.e. $2\pi f_b$		
	T_b	fundamental period in s, i.e. $1/f_b$		
	h_m	maximum harmonic index for the periodic trajectory calculations with the COLL method. When studying an LTI system, set h_m to 0 and the COLL solver will find an constant equilibrium point. Note that the periodic trajectory calculation results can be inaccurate if h_m is lower than the actual maximum harmonic index necessary to precisely define the periodic trajectory. h_m is also used as maximum harmonic index for the display of harmonic spectra, regardless of the method used to obtain these spectra.	int	6
	nh_m	total amount of harmonics (including negative and positive frequencies) in the periodic trajectory, i.e. nh_m = 2*h_m + 1.	int	
	h	a vector of length nh_m, containing the harmonic indices between $-h_m$ and $+h_m$	vector	
	f	a vector of length nh_m, containing the harmonic frequencies between $-h_m f_b$ and $+h_m f_b$, in Hz	vector	
	TD2HD	an inverse FFT matrix		
	HD2TD	a direct FFT matrix		
	TTM_diff	a time-transfer matrix to perform time derivation of sampled-time vectors.		
	TTM_delay	a time-transfer matrix to perform time delay of sampled-time vectors.		
	DISP.T_s	sampling time (in s) to display the collocation waveforms.		
	DISP.f_s	sampling frequency (in Hz) to display the collocation waveforms.		
	DISP.t_1P	Time vector covering one fundamental period at sampling frequency DISP.f_s.	vector	

Table 1.10: DSA.calc

1.4.9 Options and by-products of the COLL calculations

The COLL solver finds periodic trajectories of dynamic systems. It relies on a Fourier-based collocation method and can handle any continuously-differentiable type of nonlinearity. Intuitively, COLL finds the same steady-state as time-domain simulations would, but "jumps" directly to steady-state. Additionally, COLL is able to find both stable and unstable periodic orbits. See [1, Ch.5] for the theory behind the Fourier-based collocation method.

field	field	description	type	example
COLL	jacobian_type	tells whether the COLL solver must use an analytical pre-calculated representation of the collocation jacobian or the numerical jacobian calculation embedded in fsolve. For small systems, the numerical approach is often sufficient. For larger systems, it is sometimes necessary to use an analytical jacobian for the COLL solver to converge within a reasonable time. It is recommended to start with 'numerical'. Use 'analytical' if the COLL solver converges too slowly. Note however that lack of convergence is not always a question of jacobian calculation, but may be caused by mistakes in the differential equations, improper numerical values, or a badly scaled system.	string	'numerical', 'analytical'
	jacobian_reset	to re-generate the analytical jacobian. Reset is ignored if 'numerical' is chosen. Must be set to 1 when an analytical jacobian is requested for the first time. Must be reset to 1 whenever the analytical differential equations of the instance under study are modified. Keep to 0 the rest of the time to reduce calculation times.	boolean	1 or 0
	def_inputs_from	definition of input variables from the user-defined sstm_inst_HD function, or from the latest solution HD_COLL_1P, or from another HD source available in DSA.HD	string	'user', 'solution', 'source'
	init_guess_from	tells whether the solver should start from an initial guess provided in the user-defined sstm_inst_HD function, or from the latest solution HD_COLL_1P, or from another HD source available in DSA.HD	string	'user', 'solution', 'source'
	HD_source	not the HD source itself, but the *name* of the HD source, if def_inputs_from or init_from are set to 'source'	string	
	algo_tol	algorithm tolerance	double	10^{-12}
	max_iter	maximum number of iterations	int	10^3
	L_DISP	number of sampling points per fundamental period for the display of the resulting time-domain waveforms. This corresponds to the *resampling* of the collocation results.	int	1625
	J_COLL	jacobian matrix of the last iteration of the COLL calculation. For systems without delays, the eigenvalues of this matrix are the frequency-lifted Floquet exponents, i.e. the eigenvalues of the frequency-lifted system representation		

Table 1.11: DSA.COLL

1.4.10 Options for the TDSIM calculations

field	field	description	type	example
TDSIM	tspan	Start and stop time (in s) for the numerical integration of the differential equations. It is recommended to start with a small time span to test the simulation first, then increase for longer simulations. For delayed systems initialised from a previous solution, the start time of the new calculation must be set to the stop time of the previous calculation. This is not a requirement for non-delayed systems.	vector	$[0, 0.2]$
	T_s	sampling period (in s) for the numerical integration. Matlab solvers may use a smaller sampling period internally, but they do return results which have this sampling period. This value controls the resolution of the obtained waveforms		10^{-6}
	f_s	sampling frequency related to T_s		10^{-6}
	NP	total duration of the time-domain simulation, expressed as a number of fundamental periods. NP means *N periods* and is automatically calculated based on the provided time span and fundamental frequencies.	int	10
	XP	in order to determine the harmonic content of the variables at the end of a time-domain simulation, the code automatically performs a Fourier analysis (FFT) on the last fundamental period of the simulation (=1P analysis). In addition, the code automatically performs a Fourier analysis on the last XP fundamental periods of the simulation (=XP analysis). This way, it is possible to identify the harmonic content at frequencies smaller than the fundamental frequency. For instance, if f_b=50, then XP=2 gives frequencies down to 25 Hz, and XP=10 gives frequencies down to 5 Hz. Naturally, XP must be lower than or equal to NP. Ideally, the system should have reached steady-state before the start of the last XP fundamental periods. XP is automatically decreased if the time span is not long enough.	int	2
	init_from	tells whether the solver should start from an initial point on a periodic trajectory described by the harmonic spectra provided in either the user-defined sstm_inst_HD function ('user') or in another HD source available in DSA.HD ('source'), or from the last solution ('solution'), or from zero ('zero'). When restarting from the last solution, the numerical integration starts again from the last point of the previously calculated waveforms.	string	'user', 'solution', 'source', 'zero'
	HD_source	not the HD source itself, but the *name* of the HD source, if init_from is set to 'source'. The corresponding HD structure must be available in DSA.HD	string	
	t_1P	time vector: one fundamental period (sampl. freq. f_s).	vector	
	t_XP	time vector: XP fundamental periods (sampl. freq. f_s).	vector	
	t_NP	the complete time vector with start and stop points given in tspan (sampl. freq. f_s).	vector	
	solver	one of the Matlab ode solvers. For delayed systems, this option is disregarded and dde23 is used automatically.	string	'ode45', 'ode15s', ...
	sim_tol	integration tolerance	double	10^{-6}

Table 1.12: DSA.TDSIM

1.4.11 Results of calculations and simulations

Fields `DSA.HD` and `DSA.TD` contains the results of the periodic trajectory calculations (COLL) and of the time-domain simulations (TDSIM). They are generated by functions `DSA_run_COLL` and `DSA_run_TDSIM`. In both cases, there are essentially two types of results: HD results, i.e. harmonic spectra of the variables, and TD results, i.e. time-domain waveforms of the variables.

field	field	description	type
HD	HD_COLL_1P	harmonic data resulting from the COLL calculation, corresponding to one fundamental period and with Fourier coefficients up to $h_m f_b$	struct
	HD_TDSIM_1P	harmonic data related to TD_TDSIM_1P and with Fourier coefficients up to $h_m f_b$	struct
	FHD_TDSIM_1P	harmonic data related to TD_TDSIM_1P and with Fourier coefficients up to half the sampling frequency	struct
	HD_TDSIM_XP	harmonic data related to TD_TDSIM_XP and with Fourier coefficients up to $h_m f_b$	struct
	FHD_TDSIM_XP	harmonic data related to TD_TDSIM_XP and with Fourier coefficients up to half the sampling frequency	struct
TD	TD_COLL_1P	waveform data related to HD_COLL_1P	struct
	TD_TDSIM_1P	waveform data resulting from the TDSIM calculation, spanning the very last fundamental period	struct
	TD_TDSIM_XP	waveform data resulting from the TDSIM calculation, spanning the last XP fundamental periods	struct
	TD_TDSIM_NP	waveform data resulting from the TDSIM calculation, covering the complete time span	struct

Table 1.13: DSA.HD and DSA.TD

1.4.12 Data related to LTP system representation

Field `DSA.LTP` contains the A(t), B(t), C(t), and D(t) matrices of the state-space representation (or counterpart matrices in case of generalised LTP representation of delayed systems) that results from the linearisation of the original system around a periodic trajectory. Yet, instead of representing the matrices as functions of time, they are represented as the set of their Fourier coefficients, spanning all harmonic indices between $-h_m$ and $+h_m$. Each Fourier coefficient is a 2D matrix, and these matrices are stacked, resulting in a 3D matrix, the third dimension being that of the harmonic indices. Note that if the system has a constant operating point instead of a steady-state operating trajectory, then $h_m = 0$ and the 3D matrices are flat (=2D) because they only have a single 0 Hz Fourier coefficient.
Field `DSA.LTP` is created when calling function `DSA_get_3D_LTP`.

field	field	description	type
LTP	HD_source	not the HD source itself, but the *name* of the HD source that defines the periodic trajectory of the variables used for the calculation of the state-space coefficients of the linearised system.	string
LTP.mat	fields related to systems with no delays		
	A_3D	dimensions $n \times n \times n_{h_m}$	matrix
	B_3D	dimensions $n \times m \times n_{h_m}$	matrix
	C_3D	dimensions $p \times n \times n_{h_m}$	matrix
	D_3D	dimensions $p \times m \times n_{h_m}$	matrix
	fields related to systems with delays		
	A_3D	dimensions $n \times n \times n_{h_m}$	matrix
	B1_3D	dimensions $n \times m \times n_{h_m}$	matrix
	B2_3D	dimensions $n \times d \times n_{h_m}$	matrix
	C1_3D	dimensions $p \times n \times n_{h_m}$	matrix
	C2_3D	dimensions $d \times n \times n_{h_m}$	matrix
	D11_3D	dimensions $p \times m \times n_{h_m}$	matrix
	D12_3D	dimensions $p \times d \times n_{h_m}$	matrix
	D21_3D	dimensions $d \times m \times n_{h_m}$	matrix
	D22_3D	dimensions $d \times d \times n_{h_m}$	matrix
	T	length d, all delay values	vector

Table 1.14: DSA.LTP

1.4.13 Options for the FLIFT calculations

Frequency-lifting essentially boils down to establishing the block-Toeplitz forms of the state-space matrices. For this reason, frequency-lifting relies on the 3D matrices stored in LTP.mat.

field	field	description	type	example
FLIFT	h_t	truncation rank for the frequency-lifting. The infinite-dimensional frequency-lifted system is truncated to this maximum harmonic rank. Note that h_t and h_m may have different values in general. In general, the frequency-lifted representation is only accurate if h_t is larger than or equal to h_m, unless all harmonic indices h such that $h_t < h \leq h_m$ are negligible. If h_t is larger than h_m, all harmonic coefficients between h_m and h_t are set to zero. If h_t is smaller than h_m, all harmonic coefficients between h_m and h_t are ignored. It is generally advantageous to set h_t larger than h_m in order to display longer vertical lines of eigenvalues in the eigenvalue plots.	int	6
	h_f	forced periodic rank. This is normally kept equal to COLL.h_m. Can be set to a lower value to purposely discard potentially non-negligible Fourier coefficients.	int	h_m

Table 1.15: DSA.FLIFT

1.4.14 Data related to LTI system representation

Field `DSA.LTI` contains the frequency-lifted version of the LTP system, thereby being an LTI system. It is generated when calling function `DSA_get_HSS`. Lifted matrix $\boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{N}}$ is stored in `mat.A`, $\boldsymbol{\mathcal{B}}$ is stored in `mat.B`, and so on. Despite the name, `DSA_get_HSS` creates either an HSS or a DHSS model, depending on whether the considered systems has pure delays or not.

field	field	description	type
LTI	fields related to systems with no delays		
	`mat.A`	dimensions $(n \cdot n_{h_m}) \times (n \cdot n_{h_m})$	matrix
	`mat.B`	dimensions $(n \cdot n_{h_m}) \times (m \cdot n_{h_m})$	matrix
	`mat.C`	dimensions $(p \cdot n_{h_m}) \times (n \cdot n_{h_m})$	matrix
	`mat.D`	dimensions $(p \cdot n_{h_m}) \times (m \cdot n_{h_m})$	matrix
	fields related to systems with delays		
	`mat.A`	dimensions $(n \cdot n_{h_m}) \times (n \cdot n_{h_m})$	matrix
	`mat.B1`	dimensions $(n \cdot n_{h_m}) \times (m \cdot n_{h_m})$	matrix
	`mat.B2`	dimensions $(n \cdot n_{h_m}) \times (d \cdot n_{h_m})$	matrix
	`mat.C1`	dimensions $(p \cdot n_{h_m}) \times (n \cdot n_{h_m})$	matrix
	`mat.C2`	dimensions $(d \cdot n_{h_m}) \times (n \cdot n_{h_m})$	matrix
	`mat.D11`	dimensions $(p \cdot n_{h_m}) \times (m \cdot n_{h_m})$	matrix
	`mat.D12`	dimensions $(p \cdot n_{h_m}) \times (d \cdot n_{h_m})$	matrix
	`mat.D21`	dimensions $(d \cdot n_{h_m}) \times (m \cdot n_{h_m})$	matrix
	`mat.D22`	dimensions $(d \cdot n_{h_m}) \times (d \cdot n_{h_m})$	matrix
	`mat.T_d`	length $d \cdot n_{h_m}$ all delay values of the lifted system	vector
	other fields		
	`HSS_SYS`	the system as a matlab ss (=state-space) object. Is actually a DHSS in case of delayed systems.	ss
	`EIGS`	eigenvalues of the state matrix or characteristic roots of the delayed system	vector
	`RVEC`	right eigenvectors corresponding to the eigenvalues or characteristic roots	matrix
	`LVEC`	left eigenvectors corresponding to the eigenvalues or characteristic roots	matrix

Table 1.16: DSA.LTI

1.4.15 Frequency responses

Field `DSA.FRESP` contains options and results related to frequency-response calculations. The use of a transfer function identifier `TFid` allows storing several frequency responses within the `DSA.FRESP` structure. The frequency response is calculated between an input variable and an output variable of the linearised (small-signal frequency-lifted) system representation.

field	field	description	type	example
FRESP.(TFid)	ivarname	name of input smsi variable	string	'v_dc'
	ovarname	name of output smsi variable	string	'i_dc'
	f_values	vector of frequency values	vector	
	sign_factor	1 or -1 to change the sign convention	int	1 or -1
	h_in	input harmonic channel	int	0
	h_shift	harmonic shift. Set to 0 to obtain the traditional no-shift frequency response	int	0
	HTF	contains the complete harmonic transfer function (in the form of a frequency response) from all smsi inputs to all smsi outputs	struct	
	HLIN	contains the individual transfer function	struct	

Table 1.17: DSA.FRESP

1.5 Matlab commands

1.5.1 Initialisation

`DSA = DSA_new(sstm, inst)` is the first command to run in order to try an example, or once the user-defined functions and scripts are ready. It takes the system and instance identifiers as arguments and returns a new version of the DSA structure, with most DSA fields initialised and, if relevant, pre-filled.

`DSA = DSA_set(DSA)` is the second command that finishes to pre-fill the DSA fields before being able to run a COLL or a TDSIM calculation. Although `DSA = DSA_set(DSA)` may be called by the user, it does not have to be called by the user: it is automatically run at the start of every COLL and TDSIM calculations (among others). This is so that calculations are run with the latest numerical values and options, accounting for modifications in numerical parameters and options by the user.

1.5.2 Collocation (COLL) calculations

Function `DSA = DSA_run_COLL(DSA)` performs the periodic trajectory calculation for the given nonlinear system, the given set of numerical parameters and the given input variables (provided by the user in the form of harmonic spectra).

The function is structured as follows:

1. If the user chooses to work with an analytical jacobian, then the jacobian is precalculated symbolically;
2. Input variables, defined as harmonic spectra, are retrieved in the form of sampled-time vectors.
3. Initial guesses for the unknowns, i.e. the states and delayed variables, are retrieved in the form of sampled-time vectors.
4. Levenberg-Marquardt algorithm provided by Matlab function `fsolve()` of the Optimization Toolbox, is used to solve the COLL formulation. Each iteration of fsolve calls `sstm_inst_EQNS`;
5. After a solution is found (or whenever the solver stops), the remaining error is plotted in a figure;
6. Function `sstm_inst_EQNS` is called one last time to extract the solution for all output variables, as well as the last-iteration jacobian matrix;
7. At this point, all variables are still in sampled-time forms. All sampled-time vectors are thus converted to harmonic vectors by means of a DFT and are saved as harmonic spectra in `DSA.HD.HD_COLL_1P`;
8. The inverse DFT is applied to all harmonic vectors at the `f_s` sampling frequency to obtain the time-domain waveforms over one fundamental period. All waveforms are stored as vectors of sample points in `DSA.TD.TD_COLL_1P`;
9. For systems without delays, the eigenvalues of the jacobian matrix are also the (duplicated-shifted) Floquet exponents of the periodic trajectory, which is equivalent to the eigenvalues of the LTP system after applying frequency-lifting with `h_t=h_m`. Consequently, the eigenvalues of the jacobian matrix contain information about system stability. They are plotted in a figure. This is not done for systems with pure delays, as the generalisation is not direct. See [1, Ch.5] for more information.

1.5.3 Analytical COLL jacobian: symbolic pre-calculation

If the user chooses to use an analytical jacobian for the COLL calculation, then it is pre-calculated symbolically. When option `COLL.jacobian_reset` is set to 1, this is done automatically from within function `DSA_run_COLL` by means of the command `DSA_make_COLL_JAC(DSA)` which takes the DSA structure as argument. It does not return anything but automatically writes (or overwrites) the jacobian script in the instance's folder.

note

`DSA_make_COLL_JAC` needs `DSA.NLSS`, defined by `DSA_get_NLSS`. Consequently, this function is also called within `DSA_run_COLL` when `COLL.jacobian_reset` is set to 1.

As explained in [1, Ch.5], the jacobian of the Fourier-based collocation formulation can be separated into two parts: the RHS which depends on variables, and the LHS which does not depend on variables (only on the maximum harmonic rank). The symbolic jacobian precalculations only concern the RHS. The LHS is precalculated numerically in `DSA_run_COLL` and does not rely on `DSA_make_COLL_JAC`.

`DSA_make_COLL_JAC` is structured as follows:

1. Symbolic expressions of dxdt=f() and z=g() are retrieved from `DSA.NLSS`;
2. Symbolic vectors of unknowns variables (states and delayed variables) are defined based on `DSA.vars`;
3. Matlab function `jacobian()` is called to obtain a symbolic expression of `J_2`. This may take some time for large systems;
4. The actual dimensions of the RHS depend on the number of harmonics `h_m`, so it must be defined *dynamically*. The code takes care of this automatically. However, the way in which this is automated depends on the way in which the Matlab Symbolic Math Toolbox works, which may depend on the Matlab version being used;
5. Matlab function `matlabFunction()` is used to create a new Matlab script. This script is called during the COLL calculation and returns the numerical value of the RHS at every iteration step;
6. The analytical jacobian script is saved in the instance's folder.

1.5.4 Time-domain simulations

The command `DSA = DSA_run_TDSIM(DSA)` runs a time-domain simulation of the system, i.e. a numerical integration of the differential equations with respect to time.

1. The initial values (for ode systems, without pure delays) and history values (for dde systems, with pure delays) are defined;
2. The differential equations are integrated (=solved) over the requested time span:
   - The Matlab solver indicated by the user is used for ode systems. The whole time span is simulated in a single run.
   - The Matlab solver `dde23` is used for dde systems. The whole time span is split into a number of subintervals of lenth equal to the smallest non-zero delay. The subintervals are simulated one by one, each providing the history values for the next one. This allows to speed up the overall simulation of the delayed system.
3. Function `sstm_inst_EQNS` is called one last time to extract the waveforms of all output variables. All waveforms spanning the complete time span are extracted into `DSA.TD.TD_TDSIM_NP`.

4. The last 1P fundamental period is extracted into `DSA.TD.TD_TDSIM_1P` and the DFT is applied to obtain the harmonic spectrum of all variables, stored into `DSA.HD.HD_TDSIM_1P`.

5. The last XP fundamental periods are extracted into `DSA.TD.TD_TDSIM_XP` and the DFT is applied to obtain the harmonic spectrum of all variables, stored into `DSA.HD.HD_TDSIM_XP`.

1.5.5 Plotting waveforms and harmonic spectra

The function `DSA_plot(DSA)` facilitates the display of the results. To use this function correctly, is it necessary to open the file.

1. First, open the function and run the code snippet that says `run this code snippet`. It will print the list of available variables in the console.

2. Copy-paste the printed list somewhere below, in order to replace the cell named `sets`. Uncomment the variables of interest.

3. Adapt the cells `TD_sources` and `HD_sources` to specify the type of result you want to plot.

   - For `TD_sources`, each line contains four arguments:
     (a) the name of the TD data source to be displayed;
     (b) the line type for the plot;
     (c) the text to be displayed in the legend;
     (d) the density reduction (integer): the plot density is reduced by this factor. For instance, 10 means only one out of 10 data points is displayed in the waveform.
   - For `HD_sources`, each line contains two arguments:
     (a) the name of the HD data source to be displayed;
     (b) the text to be displayed in the legend.

4. Run the file. Requested TD waveforms are displayed in one figure. Requested HD spectra are displayed in separate figures.

1.5.6 Symbolic system definition

`DSA = DSA_get_NLSS(DSA)` builds a symbolic representation of the differential equations, and stores it in `DSA.NLSS`. It takes the DSA structure as argument and returns a new version of the structure. It works as follows:

1. definition of all parameters, input and state variables as symbolic variables;

2. symbolic evaluation of the algebraic relationships and of the dxdt vector in `sstm_inst_EQNS`.

1.5.7 Symbolic system linearisation

`DSA = DSA_linearise(DSA)` linearises the nonlinear symbolic equations stored in `DSA.NLSS`. Consequently, it is necessary to run `DSA_get_NLSS` before calling `DSA_linearise`. The symbolic matrix coefficients of the linearised system are stored in `DSA.LNSS.mat`.

1.5.8 Frequency-lifting (1 of 2): retrieving the 3D-LTP system

Once a periodic trajectory is available thanks to COLL or TDSIM, the command `DSA = DSA_get_3D_LTP(DSA)` numerically evaluates the symbolic linearised system representation stored in `DSA.LNSS`. The steady-state periodic trajectory around which the LTP system revolves is chosen by the user and must be available in `DSA.HD`.

The function is structured as follows:

- All matrix coefficients of the symbolic linearised system representation stored in `DSA.LNSS` are retrieved;
- All variables are assigned with their periodic trajectory in the form of sampled-time vectors;
- All parameters are assigned with their numerical value from `DSA.data.param`;
- All matrix coefficients are numerically evaluated and then transformed back to the harmonic domain, in the form of their Fourier coefficients.
- The Fourier matrix coefficients are stacked as 3D matrices, the third dimension corresponding to harmonic indices.
- The 3D-matrices are stored in `DSA.LTP.mat`

note

Since the DFT is applied element-wise to the entries of the state-space matrices, this function is computationally intensive for large systems: it is not (yet) optimised for speed.

1.5.9 Frequency-lifting (2 of 2): LTP-to-LTI transformation

The command `DSA = DSA_get_HSS(DSA)` transforms the LTP system into an equivalent LTI system within a higher-dimensional space: this is done via frequency-lifting. See [1], specifically chapters 3 and 4, to learn about frequency-lifting in the form of HSS and DHSS.

This function essentially relies on a subfunction called `DSA_mat2toeplitz`, which transforms the 3D matrices of the LTP representation into large block-Toeplitz matrices made of the Fourier coefficients of the time-periodic matrices. The matrices of the resulting frequency-lifted LTI state-space representation are stored in `DSA.LTI.mat`. Additionally, the system is saved as an `ss` (=state-space) Matlab object in `DSA.LTI.HSS_SYS` where, despite the name, the system may be either HSS or DHSS, depending on the presence of pure delays.

note

The basic principle of frequency-lifting consists in realising that the state-space coefficients of linear time-periodic systems can be represented as Fourier series, and that the Fourier coefficients are constants. The LTP system can therefore be rearranged so that the new state-space coefficients are constant.

references

- The original PhD Thesis of Wereley on frequency-lifting in the form of harmonic state-space [2];
- The original PhD Thesis of Möllerstedt on frequency-lifting in the form of the harmonic transfer function [3];
- A book by Bittanti on periodic systems, where the harmonic state-space framework is presented as well [4];
- A presentation of frequency-lifting through a comparison of harmonic state-space with dynamic phasors [5];
- A generalisation of the harmonic state-space framework to systems with pure delays [6]
- The PhD thesis [1] on which this Matlab Toolbox relies.

1.5.10 Frequency response calculations

While obtaining symbolic expressions of the harmonic transfer function (HTF) of frequency-lifted systems is often impossible, it is possible to calculate the frequency response of the frequency-lifted system. Both "no-shift" and "shift" transfer functions can be retrieved by means of command `DSA = DSA_run_FRESP(DSA, TFid)` with `TFid` a transfer function identifier.
The principle of the frequency-response calculation is to call Matlab `freqresp` function on the state-space object `DSA.LTI.HSS_SYS`, for a vector of frequency values. The frequency responses are stored under `DSA.FRESP.(TFid)`, and a bode plot of the frequency responses is displayed.

note

The use of transfer function identifiers makes that it is possible to calculate multiple frequency responses and to store them in the `DSA.FRESP` structure without overwriting previously calculated frequency responses.

note

A version of `DSA_run_FRESP(DSA, TFid)` is also available for three-phase systems: `DSA_run_FRESP_3PH(DSA, TFid)`, where both abc-to-abc and pnz-to-pnz frequency responses are calculated (pnz referring to positive-negative-zero sequences).

1.5.11 Getting the eigenvalues of the LTI system

For systems without pure delays, command `DSA = DSA_eigs(DSA)` calculates the eigenvalues and eigenvectors of matrix `DSA.LTI.mat.A` (which, in the case of the HSS representation, corresponds to round-A minus round-N: $\mathcal{A}-\mathcal{N}$) by means of Matlab function `eig()`. Eigenvalues, right and left eigenvectors are stored in `DSA.LTI`.

The calculation of characteristic roots of delayed systems relies on additional files (not written by the author of the present document, but provided with the rest of the DSAnalysis code under subfolder `DDEstability`), with which the DSAnalysis Toolbox is interfaced. The files in the `DDEstability` subfolder are obtained from KU Leuven Prof. Wim Michiels' webpage: https://twr.cs.kuleuven.be/research/software/delay-control/. This additional set of functions implements the iterative infinite Arnoldi algorithm, referred to as the delay Arnoldi algorithm in [1, Ch.4].

1.5.12 Participation factors analysis

Function `DSA_partfactors(pfa)` performs a participation factor analysis. Unlike other functions, it uses its own `pfa` (=participation factor analysis) structure instead of the DSA structure:

- `pfa.EIGS`: A vector containing the eigenvalues;
- `pfa.RVEC`: A matrix whose columns are the right eigenvectors;
- `pfa.LVEC`: A matrix whose columns are the left eigenvectors;
- `pfa.VARS`: A cell containing the names (=strings) of the states;
- `pfa.MODES`: An empty vector means that all modes are displayed. Otherwise, `MODES` may contains the index of a subset of all the modes.
- `pfa.XMIN_TXT`: Mode indices are displayed as text on the figure only for eigenvalues with real part larger than the value of `XMIN_TXT`. Setting this to -Inf means that all indices will be displayed, which can take quite some time. Set this to a small negative value to display only the indices of a few predominant modes.

- `pfa.YMIN_TXT`: likewise, but along the imaginary axis.

Bibliography

[1] P. De Rua, *Model Transformations and Periodic Trajectory Calculations for Stability Assessments of Modular Multilevel Converter-Based Systems.* PhD thesis, Department of Electrical Engineering, KU Leuven, 2023.

[2] N. M. Wereley, *Analysis and control of linear periodically time varying systems.* PhD thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology (MIT), 1991.

[3] E. Möllerstedt, *Dynamic Analysis of Harmonics in Electrical Systems.* PhD thesis, Department of Automatic Control, Lund Institute of Technology (LTH), 2000.

[4] S. Bittanti and P. Colaneri, *Periodic Systems: Filtering and Control.* Springer-Verlag London, 1 ed., 2009.

[5] P. De Rua, Özgür Can Sakinci, and J. Beerten, "Comparative study of dynamic phasor and harmonic state-space modeling for small-signal stability analysis," *Electric Power Systems Research*, vol. 189, p. 106626, 2020.

[6] P. De Rua and J. Beerten, "Generalization of harmonic state-space framework to delayed periodic systems for stability analysis of the modular multilevel converter," *IEEE Transactions on Power Delivery*, vol. 37, no. 4, pp. 2661–2672, 2022.