LIMSI-CNRS, Rue John von Neumann,
Campus universitaire d'Orsay, F-91403 Orsay, France
Technische Universität Braunschweig, ISM
Hermann-Blenk-Straße 37, D-38108 Braunschweig

Reference manual

xROM — a Toolkit for Reduced Order Modelling of Fluid Flows (Version 1.0)

Bernd R. Noack, **Daniel Fernex** & **Richard Semaan**

April 2017

Abstract

This report describes a software package xROM for analysis, modeling and control of fluid flows with steady domains. We discuss standard operating procedures for a large class of control-oriented Galerkin models — employing snapshots or pre-existing modes. In addition, we outline the structure and content of a corresponding software package. This package is called xROM for eXecute Reduced-Order Modeling. It shall enable an interdisciplinary distributed group of researchers to collaborate on a single reduced-order model. A key goal is to achieve small transfer times of ROM-related data or programs between different groups — at most 10 minutes per partner and transfer. The report suggests a file system and data structure for design parameters, computational tools and the ROM. The report comes with a basic xROM package which allows to reproduce the Galerkin models of Noack et al. [2003].

Keywords: proper orthogonal decomposition, Galerkin method

Contents

1. Introduction

Dynamic reduced-order models (ROM) for understanding and controlling fluid flows are a rapidly evolving area [Brunton and Noack, 2015, Noack et al., 2011]. ROM can be considered as a part of theoretical fluid dynamics (TFD). Considerations of TFD determine the structure of models which incorporate data from experimental and computational fluid dynamics.

Around 1900, vortex models were the most popular choice of ROM for about 5 decades. Vortex models have been used to explain the geometry of the von Kármán vortex street, the evolution of shear layers, the motion of vortex ring and many other phenomena [Lugt, 1996]. Today, vortex models are employed as computationally inexpensive plants for the evolution of shear flows and mixing.

Around 1950, increasing successes of stability theory lead a framework for low-order linear and nonlinear Galerkin models. These models gained increasing popularity [Fletcher, 1984]. Successes include explanations for onset of oscillations and various forms of transition to turbulence.

Meanwhile reduced-order modeling is dominated by empirical Galerkin models [Holmes et al., 2012]. These models post-process the increasing amount of superb experimental and numerical data for understanding and control. Construction of control-oriented ROM often require an interdisciplinary group of people from CFD, experiments, theoretical fluid mechanics, control theory and other fields of mathematics and physics.

This report focuses on empirical ROM and addresses a small yet crucial aspect of numerical tasks in ROM: the book-keeping and interface problems associated with the collaboration of different often spatially distributed groups. We follow the UML philosophy and focus on about 80% of the typical tasks of the pipeline from data to dynamical model. The remaining 20% of the tasks may be performed by easy plug-ins, auxiliary programs and the like.

The execution of the ROM pipeline shall be doable by different possibly distant groups with their own tools. The interface time, i.e. passing data from one to another group should require less than 10 minutes for each side. This requires a well documented standard operating procedure (SOP) with an analytical pipeline. Our corresponding endeavor is termed `xROM` for 'eXecute Reduced Order Modelling'. `xROM` assumes a multi-author, multi-source, multi-data and multi-Linux platform.

The report has three main chapters. Chapter 2 details the theory to construct a POD-based Galerkin model, from the data to the dynamical system. Chapter 3 is the user guide. After reading this chapter, the user should be able to prepare and organize the data, choose the correct parameters for his case and run the program. The procedure is exemplified with a cylinder wake model (from [Noack et al., 2003]) in Chapter 4.

This software package comprises over two decades worth of ROM experience with many collaborators. We thank Chris Airiau, Laurent Cordier, Oliver Lehmann, Guillaume Lehnasch, Mark Luchtenburg, Marek Morzyński, Michael Schlegel and Gilead Tadmor for their contribution to `xAMC`, an earlier software package for a similar purpose Noack and Cordier [2012]. We are deeply indebted to Marian Albers, Maciej Balajewiecz, Jason Bourgeois, Steven Brunton, Camila Chovet, Nicolas Herouard, Angelo Iollo, Eurika Kaiser, Laurent Keirsbulck, Siniša Krajnović, Jean-Christophe Loiseau, François Lusseyran, Robert Martinuzzi, Lionel Mathelin, Pascal Meysonnat, Marek Morzyński, Robert Niven, Luc Pastur, Bartek Protasz, Peter Schmid,

Wolfgang Schröder, Witek Stankiewicz, Razvan Stefanescu, Jan Östh and Clancy Rowley for continual discussions about future applications and future directions of ROM.

2. Standard operating procedure for a POD Galerkin model

In this chapter, we outline the path from the flow configuration to a low-order dynamical system. This path comprises the initial boundary value problem (Sec. 2.1), the required data (Sec. 2.2), the Galerkin expansion as reduced-order representation (Sec. 2.3), the Galerkin projection as compression of the Navier-Stokes equation (Sec. 2.4), and the dynamical system as plant for further investigations (Sec. 2.5).

2.1 Configuration

We assume an incompressible viscous flow in a steady domain Ω. A location in this domain is denoted by $\boldsymbol{x} = (x, y, z) = (x_1, x_2, x_3)$ or its 2-D pendant. The time is represented by t. The velocity and pressure fields read $\boldsymbol{u} = (u, v, w) = (u_1, u_2, u_3)$ and p, respectively. The Newtonian fluid is described by the density ρ and dynamic viscosity μ.

Let D and U be characteristic size and velocity scales of the configuration. The flow properties are characterized by the Reynolds number $Re = UD/\rho\mu$, or, equivalently by its reciprocal $\nu = 1/Re$. In the sequel, we assume that all quantities are non-dimensionalized with D, U, ρ and μ.

The flows may be manipulated with volume forces or with boundary-imposed unsteadiness, e.g. local actuators. The volume force $\boldsymbol{g}(\boldsymbol{x}, t)$ shall be approximated by an expansion of N_V fields $\boldsymbol{g}_l$ and their amplitudes b_l for $\boldsymbol{x} \in \Omega$:

$$\boldsymbol{g}(\boldsymbol{x}, t) = \sum_{l=1}^{N_V} b_l(t)\, \boldsymbol{g}_l(\boldsymbol{x}). \tag{2.1}$$

The continuity and Navier-Stokes equation read

$$\nabla \cdot \boldsymbol{u} = 0, \tag{2.2}$$

$$\partial_t \boldsymbol{u} + \nabla \cdot (\boldsymbol{u} \otimes \boldsymbol{u}) = -\nabla p + \nu \triangle \boldsymbol{u} + \sum_{l=1}^{N_V} b_l\, \boldsymbol{g}_l. \tag{2.3}$$

The boundary-imposed unsteadiness shall be comprised by N_A actuators such that the Dirichlet boundary conditions reads at walls $\boldsymbol{x} \in \partial\Omega$. The resulting time-varying base-flow is approximated by

$$\boldsymbol{u}^B(\boldsymbol{x}, t) = \boldsymbol{u}_0(\mathbf{x}) + \sum_{i=-N_A}^{-1} a_i(t)\, \boldsymbol{u}_i(\boldsymbol{x}). \tag{2.4}$$

The actuators may be spatially disjoint, e.g. in case of two separated acoustic actuators, or may overlapping, e.g. in case of the superposition of different dynamics [Semaan et al., 2016].

2.2 Data from the high-fidelity plant

We assume that we have M snapshots at times t_m, $m = 1, \ldots, M$ as well as the corresponding forcing:

$$\begin{aligned} \boldsymbol{u}^m(\boldsymbol{x}) &:= \boldsymbol{u}(\boldsymbol{x}, t_m), \quad m = 1, \ldots, M; && (2.5)\\ a_l^m &:= a_l(t_m), \quad l = -N_A, \ldots, -1, m = 1, \ldots, M; && (2.6)\\ b_l^m &:= b_l(t_m), \quad l = 1, \ldots, N_V, m = 1, \ldots, M. && (2.7) \end{aligned}$$

2.3 Galerkin expansion

Following Noack et al. [2004], the Galerkin expansion reads

$$\boldsymbol{u}(\boldsymbol{x}, t) = \sum_{i=-N_A}^{N} a_i(t)\, \boldsymbol{u}_i(\boldsymbol{x}). \tag{2.8}$$

The mode amplitudes for the snapshots are $a_l^m := a_l(t_m)$, $m = 1, \ldots, M$, $l = -N_A, \ldots N$. Negative indices $i < 0$ correspond to pre-determined *actuation modes*, a vanishing index $i = 0$ to the stationary *basic mode* and positive indices $i > 0$ to *expansion modes*.

The computation of the POD modes is performed in following steps:

Snapshots: Starting point are M flow snapshots at times t_m, $m = 1, \ldots, M$.

$$\boldsymbol{u}^m(\boldsymbol{x}) := \boldsymbol{u}(\boldsymbol{x}, t_m), \quad m = 1, \ldots, M. \tag{2.9}$$

Subtraction of boundary-induced unsteadiness: In the next step, the actuation effect is subtracted so that the resulting snapshots satisfy steady boundary conditions:

$$\boldsymbol{v}^m(\boldsymbol{x}) := \boldsymbol{u}^m(\boldsymbol{x}) - \sum_{i=-N_A}^{-1} a_i^m\, \boldsymbol{u}_i(\boldsymbol{x}), \quad m = 1, \ldots, M. \tag{2.10}$$

Here, the actuation amplitudes are denoted by $a_i^m := a_i(t_m)$, $m = 1, \ldots, M$, i.e. share the same superscripts as the corresponding snapshots.

Computation of the basic mode: The basic mode absorbs the inhomogeneity of the steady boundary condition:

$$\boldsymbol{u}_0(\boldsymbol{x}) := \frac{1}{M} \sum_{m=1}^{M} \boldsymbol{v}^m(\boldsymbol{x}) \tag{2.11}$$

This basic mode coincides with the mean flow, $\overline{\boldsymbol{u}} = \boldsymbol{u}_0$ only if the actuation amplitudes have vanishing mean values,

$$\overline{a_i} = \frac{1}{M} \sum_{m=1}^{M} a_i^m = 0, \quad i = -N_A, \ldots, -1.$$

The resulting fluctuations

$$\boldsymbol{w}^m(\boldsymbol{x}) := \boldsymbol{v}^m(\boldsymbol{x}) - \boldsymbol{u}_0, \quad m = 1, \ldots, M \tag{2.12}$$

fulfill homogenized boundary conditions.

Computation of the correlation matrix: The $M \times M$ correlation matrix $\boldsymbol{R}$ has the elements

$$\boldsymbol{R}_{mn} := \frac{1}{M} \left(\boldsymbol{w}^m, \boldsymbol{w}^n\right)_\Omega, \quad m, n = 1, \ldots, M \tag{2.13}$$

Spectral analysis of the correlation matrix: This gramian matrix $\boldsymbol{R}$ is symmetric and positive semi-definite. This implies real and non-negative eigenvalues as well as orthogonal eigenvectors. Let

$$\boldsymbol{\alpha}_i := \begin{pmatrix} \alpha_i^1 \\ \vdots \\ \alpha_i^M \end{pmatrix}$$

be the ith eigenvector of the correlation matrix, i.e.

$$\boldsymbol{R}\, \boldsymbol{\alpha}_i = \lambda_i\, \boldsymbol{\alpha}_i, \quad i = 1, \ldots, M. \tag{2.14}$$

Without loss of generality, we assume the real eigenvalues sorted in decreasing order

$$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_M = 0$$

and that the eigenvectors are orthonormal

$$\boldsymbol{\alpha}_i \cdot \boldsymbol{\alpha}_j = \sum_{m=1}^{M} \alpha_i^m \alpha_j^m = \delta_{ij}, \quad i, j = 1, \ldots, M.$$

Note the M-th eigenvalue must vanish, because M snapshots span at most a $M-1$-dimensional hyperplane.

Computation of the POD modes: The POD modes are given by

$$\boldsymbol{u}_i(\boldsymbol{x}) := \frac{1}{\sqrt{M\, \lambda_i}} \sum_{m=1}^{M} \alpha_i^m\, \boldsymbol{w}^m(\boldsymbol{x}), \quad i = 1, \ldots, N. \tag{2.15}$$

Here, $N \leq M - 1$ is the number of expansion modes. The POD modes are orthonormal by construction:

$$\left(\boldsymbol{u}_i, \boldsymbol{u}_j\right)_\Omega = \delta_{ij}, \quad i, j = 1, \ldots, N.$$

The mode amplitudes scale — not accidentally — with the corresponding eigenmodes of the correlation matrix

$$a_i^m = \sqrt{\frac{\lambda_i}{M}} \alpha_i^m$$

and satisfy

$$\overline{a_i} = \frac{1}{M} \sum_{m=1}^{M} a_i^m = 0, \quad i = 1, \ldots, N; \tag{2.16a}$$

$$\overline{a_i\, a_j} = \frac{1}{M} \sum_{m=1}^{M} a_i^m a_j^m = \lambda_i\, \delta_{ij}, \quad i, j = 1, \ldots, N. \tag{2.16b}$$

2.4 Galerkin projection

Projection of the expansion (2.8) on the Navier-Stokes equations (2.3) along the test functions $\boldsymbol{v}_i$, $i = 1, \ldots, N$ yields

$$\sum_{j=-N_A}^{N} m_{ij}\,\dot{a}_j = \nu \sum_{j=-N_A}^{N} l_{ij}^{\nu}\,a_j + \sum_{j=-N_A}^{N} l_{ij}^{+} a_j + \sum_{j,k=-N_A}^{N} \left(q_{ijk}^{c} + q_{ijk}^{p}\right) a_j\,a_k + \sum_{l=1}^{Nv} g_{il} b_l \quad i = 1, \ldots, N, \tag{2.17}$$

where

$$m_{ij} = (\boldsymbol{v}_i, \boldsymbol{u}_j)_\Omega\,, \tag{2.18a}$$
$$l_{ij}^{\nu} = (\boldsymbol{v}_i, \triangle \boldsymbol{u}_j)_\Omega\,, \tag{2.18b}$$
$$q_{ijk}^{c} = (\boldsymbol{v}_i, \nabla \cdot (\boldsymbol{u}_j \boldsymbol{u}_k))_\Omega\,, \tag{2.18c}$$
$$q_{ijk}^{p} = (\boldsymbol{v}_i, -\nabla p_{jk})_\Omega\,, \tag{2.18d}$$
$$g_{il} = (\boldsymbol{v}_i, \boldsymbol{g}_l)_\Omega\,. \tag{2.18e}$$

In case of a POD model [Holmes et al., 2012], the test functions are generally identical to the POD modes , i.e.

$$\boldsymbol{v}_i = \boldsymbol{u}_i, \quad i = 1, \ldots, N.$$

In case of an expansion with stability eigenmodes, the test functions are generally the adjoint eigenmodes with

$$(\boldsymbol{u}_i, \boldsymbol{v}_j) = \delta_{ij}, \quad i, j = 1, \ldots, N.$$

Again, the reader is asked to refer to the original literature for the details.

2.5 Dynamical system

The Galerkin system (2.17) preserves the physics, i.e. each Galerkin system term approximates a Navier-Stokes term. The system contains several constant, linear and quadratic terms in time-varying amplitudes $t \mapsto a_i(t)$, $i \neq 0$. From a dynamical systems or control theory point of view, the Galerkin system contains a large redundancy of terms, e.g. constant and linear terms of four different sources. In the following transcriptions, we remove this redundancy.

2.5.1 Dynamical system for the expansion modes

First, a dynamical system is derived from the Galerkin system. Equation (2.17) can be compressed in the form:

$$\dot{a}_i = \sum_{j,k=-N_A}^{N} q_{ijk}^{+}\,a_j\,a_k + \sum_{l=-N_A}^{N_V+N_A} g_{il}^{+} b_l, \quad i = 1, \ldots, N \tag{2.19}$$

where $a_0 \equiv 1$ and $b_l = \dot{a}_l$ for $l < 0$. Note that we have N equations for $N + N_V + 2N_A$ amplitudes. The control law determines $N_V + 2N_A$ amplitudes,

$$b_l = \begin{cases} a_l & l \in \{-N_A, \dots, -1\} \\ b_l & l \in \{1, \dots, N_V\} \\ \dot{a}_{l-N_V} & l \in \{N_V + 1, \dots, N_V + N_A\} \end{cases}$$

Full-state closed-loop control implies

$$b_l = g_l(a_1, \dots, a_N), \quad \text{for} \quad l = -N_A, \dots, -1 \quad \text{or} \quad l = 1, \dots, N_V + N_A. \tag{2.20}$$

We assume that the control law respects that the actuation mode amplitude a_l and its derivative $\dot{a}_l$, $l \in \{-N_A, \dots, -1\}$, are dependent quantities.

2.5.2 Descriptor system for the whole phase space

We comprise dynamics and control laws in a larger dimensional phase space $N^+ = N+N_V+2N_A$:

$$a_i = \begin{cases} a_i & i \in \{1, \dots, N\} \\ a_{N-i} & i \in \{N+1, \dots, N+N_A\} \\ \dot{a}_{N+N_A-i} & i \in \{N+N_A+1, \dots, N+2N_A\} \\ b_{i-N-2N_A} & i \in \{N+2N_A+1, \dots, N+2N_A+N_V\} \end{cases} \tag{2.21}$$

Now, (2.19) can be formulated as a *descriptor system*, or, equivalently, as *differential-algebraic equations* (DAE):

$$\dot{a}_i = f_i^+(a_1, \dots, a_{N^+}), \quad i = 1, \dots, N \tag{2.22a}$$

$$a_i = f_i^+(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_N) \quad i = N+1, \dots, N^+ \tag{2.22b}$$

where f_i^+, $i = 1, \dots, N$ is a polynomial of a_i of second order. In many examples, the control laws $a_i = f_i^+$, $i = N+1, \dots, N^+$ have a similar polynomial structure.

In the following, we drop the '+' superscript. Moreover, we introduce the indicator function $z_i = \pm 1$, depending on if i specifies a differential ($z_i = 1$) or an algebraic equation ($z_i = -1$). Now, (2.22) can be rewritten in the general form:

$$f_i = \sum_{j,k=0}^{N} q_{ijk} a_j a_k = \begin{cases} \dot{a}_i & \text{if} \quad z_i > 0 \\ a_i & \text{if} \quad z_i < 0 \end{cases}. \tag{2.23}$$

Note that (2.23) allows to code control laws and slaving on inertial manifolds in the same manner. Numerically, there is no difference between both manifolds.

3. User guide

This chapter gives all required information to prepare the data and run the program. First, an overview of the program capabilities and options is given in section 3.1. A more detailed description on how to properly use the program is then presented: how to download and execute the package (section 3.2), a description of the file structure (section 3.3) and the available parameters and their usage (section 3.4).

3.1 xROM capabilities

xROM is mainly designed to generate POD Galerkin models for a wide range of flow problems. To accomplish this, it has several additional pre-processing tools and options that extend its capabilities and configurability. The main features that make xROM attractive are presented below.

Pre-processing: two tools are available for the pre-processing phase. First, a PIV converter that directly converts PIV snapshots from their original `.txt` or `.dat` format to snapshots readable by xROM. Second, a tool to reduce the domain according to user-defined specifications, in order to compute the POD modes only in a specific region of interest.

Formats: xROM currently reads/writes two formats: CFD General Notation System (CGNS), efficient binary open-source format, and the ASCII format from Tecplot, `.dat`, which is widely used in the research community.

Case types: xROM support a wide variety of mesh types: cartesian equidistant, cartesian, structured and unstructured grids, in $2D$ and $3D$. The unstructured cases include the following elements:

- $2D$: triangles, quadrilaterals, and polygons with any number of nodes.
- $3D$: tetrahedra, hexahedra, pyramids, prisms and polyhedra with any number of nodes.
- Mixed meshes (with different element types in the same zone).

Parallelization (premium version): the software is fully parallelized using MPI, which offers two main advantages:

1. In case of a big dataset, the RAM of a single computer can quickly become insufficient. Parallelization allows to split the RAM load on multiple cluster nodes.
2. The computation time is reduced.

Please note that xROM currently *does not* offer a domain decomposition tool. The user has to decompose the domain himself and store each subdomain in the correct folder.

3.2 Download and run the program

xROM is distributed as a single binary executable file, which makes its usage very easy. It should work on any Linux distribution with a C library version 2.19 or newer. To check the C library version, run in a terminal:

```
$ ldd --version
```

Comment: if you are using an older Linux distribution, i.e. the command above returns a version older than 2.19, we can provide an executable compatible with older distributions. Note, however, that it will have limited features.

The package can be downloaded from this page[1]. It is password protected so be sure you have contacted the xROM team before downloading the package. The archive contains three items: the xROM software, the documentation and a sample case. Once the archive is extracted, open a terminal and go to where the binary is stored. Make sure that the binary is executable. If not, run:

```
$ chmod +x xROM
```

xROM can be run in serial or in parallel. In both cases, it needs the path to the input folder as argument to run correctly. There are two possibilities to define it: by specifying it directly as command line argument or by using the pop-up window. The two ways to define the input folder, as well as the parallel execution are shown below:

Input folder from the command line: in that case, simply specify the correct path to the case folder when running the program, such as:

```
$ ./xROM /absolute/path/to/folder
```

Please note that it has to be the absolute path, which can be found using:

```
$ pwd
```

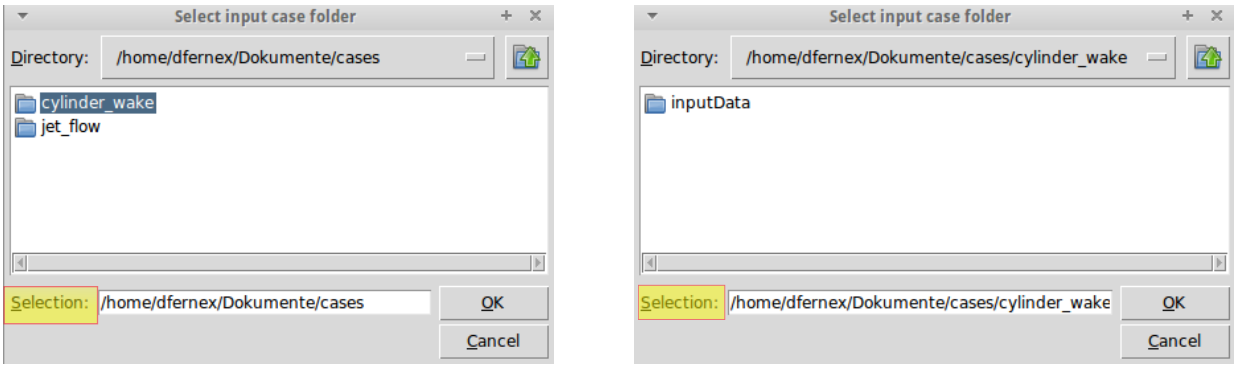
Input folder from the pop-up: if no path is defined in the command line, a pop-up window opens in which the user must select the right folder. In this case, the correct command is:

```
$ ./xROM
```

Be careful to correctly select the folder: *it is valid only when the field 'Selection' of the window contains the full path, including the input folder.* In practice, the user has to click twice on the case folder (i.e. enter the folder) to get the path, unlike other common GUIs where the user only have to click once. This is exemplified in fig. 3.1, which shows how the `cylinder_wake` folder is wrongly (3.1a) and correctly (3.1b) selected.

Execution in parallel: it requires `mpiexec` and works with the two above presented methods to select the input folder. Either with the full path:

[1] http://reducedordermodelling.com/xROM



(a) `cylinder_wake` folder not selected. (b) `cylinder_wake` folder correctly selected.

Figure 3.1: Incorrect and correct selection of the `cylinder_wake` folder.

```
$ mpiexec -n <number of nodes> xROM /absolute/path/to/folder
```

or with the GUI:

```
$ mpiexec -n <number of nodes> xROM
```

The provided `cylinder_case` is a numerical test case of a flow over a cylinder as presented in [Noack et al., 2003]. It can be used as a guide and as a test.

3.3 File structure of the case folder

Before running `xROM`, the user must prepare the case folder, which has to follow specific structure and names. Some files and folders are automatically created by `xROM` during the execution but some are required as input. The root folder (or case folder, which can have any name, e.g. 'cylinder wake',...) contains three elements: the configuration file (`ConfigFile`), the folder containing all input data provided by the user (`InputData/`) and the folder containing the output data created by `xROM` (`OutputData/`). The file structure is illustrated in fig. 3.2.

```
Case folder/ ..... user defined name
├── ConfigFile
├── InputData/
└── OutputData/
```

Figure 3.2: File structure of the case folder

The content of the `InputData` and `OutputData` folders are described respectively in section 3.3.1 and section 3.3.2. The configuration file is detailed separately in (section 3.4).

Comment: in this section, all the field data files are labeled with the `.cgns` extension as an example. Tecplot `.dat` format is also supported.

3.3.1 InputData/

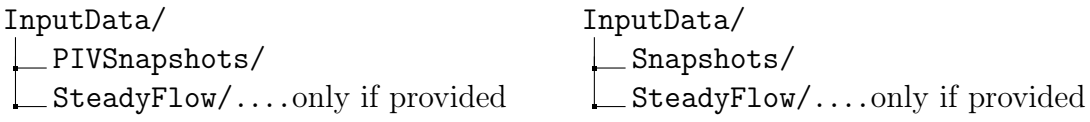
This folder is the only one in which the user has to store data. The data is organized in subfolders, which must be filled differently depending on the case configuration. It is however important to emphasize that only part of them need to be created by the user. The others are automatically created by xROM, depending on the case configuration. Therefore, the first part of this section details the folders that must be created. All the possible folders of InputData/ and their content are detailed thereafter.

User-created folders

The folders that are created by the user differ in the two following cases:

- xROM executed with PIV conversion: in that case, the user needs to create the PIVSnapshots/ folder.
- xROM executed without PIV conversion: this is the standard case and the only required folder is Snapshots/.

In both cases, if the steady flow shall be used, then the SteadyFlow/ folder must also be created. The required folders depending on the case are illustrated in fig. 3.3. Depending on the



(a) With PIV conversion (b) Without PIV conversion

Figure 3.3: User-defined minimum required folders with (3.3a) and without PIV (3.3b) data conversion

case configuration, xROM may create other subfolders in InputData. The complete structure is shown in fig. 3.4, and detailed thereafter. Notice that the PIVSnapshots/ and ReducedDomain

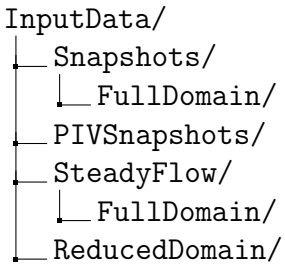


Figure 3.4: File structure of the InputData folder

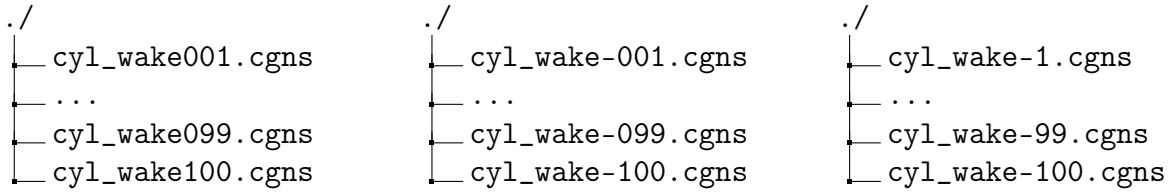
folders do not contain a FullDomain subfolder. This is because these features do not support parallel execution yet.

`Snapshots/`

This is the standard folder containing the snapshots (in `CGNS` or `.dat` format). Their name must be of the form:

`<any name><digit specifying the snapshot number>.<format>`

Where `<format>` is the the file extension (`.cgns` or `.dat`). For instance, `cylinderWake-0001.cgns` or `cylinderWake027.cgns` are valid names, but `0001-cylinderWake.cgns` is not. Also, the number of digits has to be constant. Examples of valid and invalid snapshots lists is shown in fig. 3.5.



(a) Valid list (b) Valid list (c) Invalid list

Figure 3.5: Validity of different snapshots naming.

If this folder contains files in other formats, they will simply be ignored.

The snapshots are stored differently if the execution is serial or in parallel:

Serial case In serial, all the snapshots must be stored in `FullDomain/` under `Snapshots/`. The resulting structure is presented in fig. 3.6.

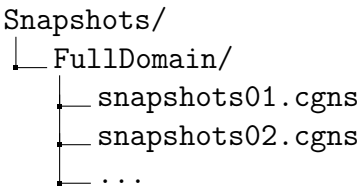


Figure 3.6: `Snapshots/` folder structure for a serial execution case.

Parallel case In parallel, the snapshots are divided in subdomains, and each subdomain must be stored in a distinct folder, named `SubDomain<i>/` where i is the subdomain number, that follows the same numbering convention as the snapshots. For instance, if there are twelve subdomains, the first subdomain will be named `SubDomain01`. Note that the numbering of the `SubDomains/` starts at one. The structure for a two parallel processors case is shown in fig. 3.7.

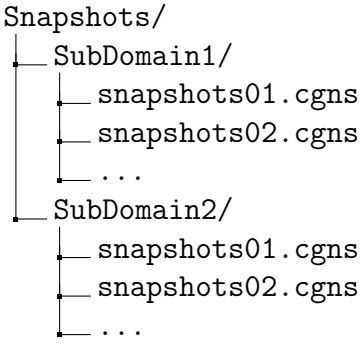


Figure 3.7: `Snapshots/` folder structure for a parallel execution case.

`PIVSnapshots/`

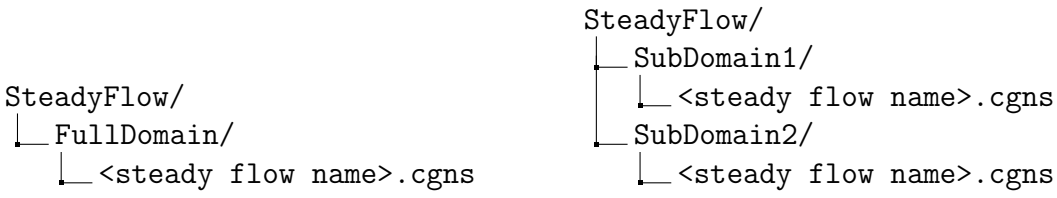
This folder is where the PIV snapshots in `.dat` or `.txt` format (depending on the PIV software) have to be stored. Lavision and Dantec-Dynamic Studio formats are both acceptable. The names follow the same convention as that of the standard snapshots, i.e. they must be of the form:

`<any name><digit specifying the snapshot number>.<format>`

The format is `.txt` for LaVision data and `.dat` for Dynamic Studio data. This folder *mustn't* contain other files than the PIV snapshots, otherwise an error will be raised.

`SteadyFlow/`

If the user wants to use the steady flow as base mode or to compute the shift mode, this is where the file of the steady flow is stored. It can have any name, as long as it has the correct extension. Again, there's a difference between a serial and a parallel execution, as shown in fig. 3.8.



(a) Serial case (b) Parallel case

Figure 3.8: Storage of the steady flow for a serial (3.8a) and a parallel execution (3.8b) (with 2 processors).

`ReducedDomain/`

In case of domain reduction, the snapshots of the reduced domain are stored in this folder. This folder doesn't require any user interaction, it is automatically created. The files will carry the same names as those from the original snapshots before reduction.

3.3.2 OutputData/

This is where all the data created during the POD-ROM process is stored. The output folder is organized according to the execution steps: Galerkin expansion, Galerkin projection and dynamical system.

There are three types of output files: `.CGNS` (or `.dat`) files for all field data (snapshots, POD modes), text files for matrices and tensors, and png files for the eigenvalues, the POD mode amplitudes, and the ROM mode amplitudes. The output folder structure and its content are self-explanatory and are presented in fig. 3.9.

```
OutputData/
├── GalerkinExpansion/
│   ├── BaseFlow.cgns ..... cgns
│   ├── CorrelationMatrix ..... matrix
│   ├── EigenValues ..... matrix
│   ├── EigenValues.png ..... Figure
│   ├── PODModes/
│   │   ├── Mode0001.cgns ..... cgns
│   │   ├── Mode0002.cgns ..... cgns
│   │   ├── ...
│   │   └── Mode1000.cgns ..... cgns
│   ├── PODAmplitudes ..... matrix
│   └── PODAmplitudesPlot/
│       ├── PODAmplitude0001.png ..... Figure
│       └── ...
├── GalerkinProjection/
│   ├── MassMatrix ..... matrix
│   ├── ViscosityMatrix ..... matrix
│   └── ConvectionTensor ..... tensor
└── DynamicalSystem/
    ├── DSAmplitudes ..... matrix
    └── DSAmplitudesPlot
        ├── DSAmplitude0001.png ..... Figure
        └── ...
```

Figure 3.9: File structure of the `OutputData` folder

The matrices and tensors are stored in simple text files so that they can be easily read, for instance with a spreadsheet.

- The matrices are simply stored in text format, as illustrated in fig. 3.10.
- The tensors are stored 'by slices'. That is, the tensor Q of dimensions (l, m, n) will be stored in l matrices representing $Q(l, :, :)$. Each of these matrices are preceded by the slice number, i.e. <slice number>jk. Figure 3.11 shows for instance how a tensor of dimensions (l, m, n) is saved (only two first 'slices' are shown).

$$\begin{matrix} l_{1,1} & l_{1,2} & \cdots & l_{1,n} \\ l_{2,1} & l_{2,2} & \cdots & l_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ l_{m,1} & l_{m,2} & \cdots & l_{m,n} \end{matrix}$$

Figure 3.10: Structure of a martix l of dimensions (m, n) as saved by `xROM`.

$1jk$

$$\begin{matrix} Q_{1,1,1} & Q_{1,1,2} & \cdots & Q_{1,1,n} \\ Q_{1,2,1} & Q_{1,2,2} & \cdots & Q_{1,2,n} \\ \vdots & \vdots & \ddots & \vdots \\ Q_{1,m,1} & Q_{1,m,2} & \cdots & Q_{1,m,n} \end{matrix}$$

$2jk$

$$\begin{matrix} Q_{2,1,1} & Q_{2,1,2} & \cdots & Q_{2,1,n} \\ Q_{2,2,1} & Q_{2,2,2} & \cdots & Q_{2,2,n} \\ \vdots & \vdots & \ddots & \vdots \\ Q_{2,m,1} & Q_{2,m,2} & \cdots & Q_{2,m,n} \end{matrix}$$

Figure 3.11: Structure of a tensor Q of dimensions (l, m, n) as saved by `xROM`. Only the first two 'slices' of the tensor are shown.

The png figures are created to give a quick visualization of the results.

3.4 Configuration file

The configuration file contains all the processes and their options to be executed: the tools that will be used (PIV conversion, domain reduction, ...), the steps that will be executed, and the parameters for each of these steps. It is the *only* required control input from the user. All capabilities of `xROM` can be set up in this configuration file.

3.4.1 General description

The general form of an input is:

```
<parameter> :  <value>
```

The detailed specifications are the following:

- There's only one parameter and its value per line
- The parameter and its value are separated by a column ':'
- The spaces in the line are irrelevant
- The position of a line in the file does not matter: it is for instance the same if a parameter is defined in the first line or in the last line
- To deactivate an option, the user must add the comment symbol '#' at the beginning of the line.

Table 3.1 shows some input examples and their validity.

`parameter : value`	valid
`parameter : value # a comment`	valid
`# parameter : value`	not read

Table 3.1: Input example and their validity

All available input parameters are presented in the next sections (3.4.2 – 3.4.6). Each section starts with a table summarizing the parameters available for this step, followed by a detailed description for each parameters.

Some parameters require a subparameter. For instance, it is possible to reduce the domain in a circle or in a box. If the user chooses the circle, it is necessary to specify the circle center and radius, which are then both subparameters. The required subparameters, are shown in the upcoming tables under the column 'Subparam'. An example of an configuration file is presented in appendix A.

3.4.2 Steps to execute

The standard process of xROM is divided into three main steps:

1. Galerkin expansion
2. Galerkin projection
3. Dynamical system

Each of these steps contains its owns options and tools which are controlled through the configuration file. The user has the possibility to selectively execute individual steps. It is possible, for instance, to compute only the Galerkin expansion, or the Galerkin expansion *and* the projection without the dynamical system. It is even possible to use already computed steps without having to run them again. The following scenario illustrate the advantage:

- Day one: the user runs the POD decomposition to analyse the POD modes.
- Day two: the user wants to run the Galerkin projection with the previous POD modes. This is possible *without* having to perform the POD decomposition again, and thus saves a lot of time.

The parameters controlling the steps to execute are listed in table 3.2

Parameter	**Required**	**Value**	**Subparam.**	**Value**
GalerkinExpansion	yes	yes no		
GalerkinProjection	yes	yes no		
DynamicalSystem	yes	yes no		
useExistingRun	no	continue	**useRun**	name of the run
		copy	**useRun**	name of the run

Table 3.2: Available parameters to control the steps to execute

GalerkinExpansion, GalerkinProjection, DynamicalSystem

These three keywords determine whether a step should be executed.

Input options:

```
GalerkinExpansion       :    yes # or no
GalerkinProjection      :    yes # or no
DynamicalSystem         :    yes # or no
```

It is therefore possible to run steps independently from each others. For example, if one only wants to compute the POD, the input will be:

The operations computed for each step are the detailed below:

```
    GalerkinExpansion       :    yes
    GalerkinProjection      :    no
    DynamicalSystem         :    no
```

Galerkin expansion: In addition to the POD modes, xROM offers a wide range of options for the expansion modes. It is possible, for instance, to choose between the mean flow or the steady flow for the basic mode, as well as to include a shift mode (e.g. [Noack et al., 2003]). This step computes:

- Correlation matrix
- POD modes

Galerkin projection: The only option for the Galerkin projection is the number of POD modes used for the projection, which can differ from the number of computed modes (as long as it is lower than or equal to the number of the computed POD modes). This step computes the following:

- Mass matrix
- Viscosity matrix
- Convection tensor

Dynamical system: The dynamical system is controlled by the integration parameters: starting time, ending time and time step. The solution of the dynamical system is computed with an efficient ODE solver (lsoda from the ODEPACK library) with adaptive time step. This step computes:

- Integration of the dynamical system

useExistingRun (optional)

This option allows to use the results from a previous run. There are two possibilities: either continue a run (`continue`) or copy a run (`copy`). If this parameter is used, it requires the subparameter `useRun` which will specify which run (that is, which output folder) will be used.

Input options:

```
    useExistingRun          :    continue
    useRun                  :    <name of the run folder>
```

or:

```
    useExistingRun          :    copy
    useRun                  :    <name of the run folder>
```

`continue`: The `continue` option will select the folder specified by `useRun` as output folder. This allows to use results from an older run and continue with further execution steps. For instance, if the POD modes are already computed in `run1/`, it is possible to use this folder as output folder by specifying the `continue` option with `run1` as input for `useRun`. That way, the computed POD modes will be used for the next steps (Galerkin projection and dynamical system for instance). The two steps described in the example above are illustrated below:

1. First run to compute the POD modes: Writes output in `run1/` for example.

```
GalerkinExpansion       :    yes
GalerkinProjection      :    no
DynamicalSystem         :    no
```

2. Second run to compute the dynamical system, using the POD modes computed in `run1/`:

```
GalerkinExpansion       :    no
GalerkinProjection      :    yes
DynamicalSystem         :    yes
useExistingRun          :    continue
useRun                  :       run1 # To use the POD
                           modes from run1/
```

`copy`: The `copy` keywords is similar to `continue` except that instead of using the folder specified by `useRun` as output folder, it will copy it to a new folder and use that new folder as output folder. That way, all results from the `useRun` folder will be kept and can be used for comparison.

For instance, if the user executed the full Galerkin process in `run1/` and wants to try a different dynamical system with the same Galerkin projection, he will process as follows:

1. First run to compute the whole Galerkin process Writes output in `run1/` for example.

```
GalerkinExpansion       :    yes
GalerkinProjection      :    yes
DynamicalSystem         :    yes
```

2. Second run to compute the dynamical system with different parameters, using the Galerkin projection computed in `run1/`.

3.4.3 Pre-processing

This section details the possible pre-processing tools performed before the Galerkin process. The input options for this step are listed in table 3.3.

```
GalerkinExpansion       :    no
GalerkinProjection      :    no
DynamicalSystem         :    yes
useExistingRun          :    copy
useRun                  :       run1 # To use the POD
                           modes from run1/
```

Parameter	Required	Value	Subparam.	Value
RunName	no	<name>		
DataFormat	yes	dat cgns		
tStartSnap **tEndSnap** **dtSnap**	yes yes yes	<starting time> <ending time> <time step>		
plotFigures	no	yes no		
GridSpec	no	cartesian equidistant		
PIV	no	LaVision DynamicStudio	**MaskedStatus**	<status of maksed nodes>
reduceDomain	no	Sphere	**Center** **Radius**	(xC,yC,zC) r
		Box	**MinPoint** **MaxPoint**	(x,y,z) (x,y,z)

Table 3.3: Parameters to control the pre-processing

RunName

Input options:

```
RunName                 :    <name>
```

This option allows to specify a name for the output folder. By default, the output folders are named `run<n+1>` where n is the number of output folders already present in `OutputData/`.

But it is possible to give any name using `RunName`. If this keyword in specified then the output folder will be `<name>`. This can be very useful to avoid confusion between output folders.

DataFormat

Input options:

```
DataFormat              :    dat
```
or:
```
DataFormat              :    cgns
```

So far, `xROM` can read from two formats: `.cgns` and the ASCII format from Tecplot `.dat`. Both

are described below:

CGNS: is a format that is designed to facilitate the exchange and storage of CFD data. It is an international standard, open-source and cross-platform. It is a compact binary format, readable with most CFD readers, such as Paraview and Tecplot. It has the advantage of being very fast to read/write.

dat: is the ASCII format from tecplot. Its definition is detailed in the 'Data Format Guide' from tecplot. Its main advantage is that it is very widely used and popular among scientists. However, since it's a plain text format, it has the drawback of requiring more disk space than CGNS and is also much slower to read/write.

Please note that `xROM` can also read PIV data in their native format, as presented in section 3.4.3.

tStartSnap, tEndSnap, dtSnap

Input options:

```
    tStartSnap          :    <starting time>
    tEndSnap            :    <ending time>
    dt                  :    <time step>
```

The keywords `tStartSnap`, `tEndSnap` and `dtSnap` specify, respectively, the start time, end time and the time step of the input case. These keywords are not optional and are used mainly to be able to compare the mode amplitudes with the solution of the dynamical system on the same time scale.

plotFigures (optional)

Input options:

```
    plotFigures         :    yes
```
or:
```
    plotFigures         :    no
```

If this keyword is set to `yes`, all figures as shown in fig. 3.9 will be created. *This can slightly increase the computation time.* It `plotFigures` is set to `no`, then no picture will be written to the output folder.

GridSpec (optional)

Input options:

```
    GridSpec            :    cartesian
```
or:
```
    GridSpec            :    equidistant
```

In case of cartesian or equidistant meshes, `xROM` uses the best routines that bring a big computational improvement in comparison to the structured mesh. However, the CGNS format does not

make the difference between structured, cartesian and equidistant meshes (they are all considered as structured). It is therefore necessary, *in case of a cartesian or cartesian equidistant mesh*, to specify it in the configuration file so that xROM will use the best routine.

Note that this option only works with single zone CGNS files. It is not possible (yet) to specify different grid characteristics on different zones.

PIV (optional)

Input options:

```
    PIV                 :   LaVision
```

or:

```
    PIV                 :   DynamicStudio
    MaskedStatus        :   <status>
```

This tool converts PIV snapshots from their native text format (from LaVision and Dynamic Studio) into xROM format (`.cgns` or `.dat`, depending on the user choice). It handles native data from the `LaVision` (`.txt`) and `DynamicStudio` (`.dat`) PIV softwares. The user has to save the PIV snapshots in the `InputData/PIVSnaphots/` folder and specify the `PIV` option with the correct software (`LaVision` or `DynamicStudio` in the configuration file). Only $2D$ PIV data are handled for now.

In case of PIV with masking, xROM needs to recognize the masked cells. `LaVision` simply imposes a velocity of zero at the masked nodes, so they can be identified by xROM. But `DynamicStudio` gives them a specific status. So in case of data from `DynamicStudio`, it is necessary to specify the status of the nodes that are masked. It is also possible to specify multiple status of nodes to neglect, in that case the different status have to be separated with a comma. For instance, if the nodes with status 1 and 17 are masked, the input will be:

```
    PIV                 :   DynamicStudio
    MaskedStatus        :   1, 17
```

Domain reduction

Input options:

```
    reduceDomain        :   Sphere
    Center              :   (xC,yC,zC)
    Radius              :   <radius value>
```

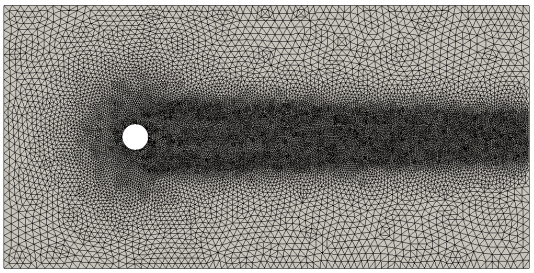
or:

```
    reduceDomain        :   Box
    MinPoint            :   (xMin,yMin,zMin)
    MaxPoint            :   (xMax,yMax,zMax)
```

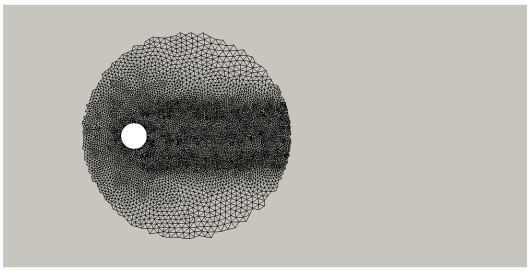
In case of numerical simulation data with a big domain, it is not uncommon for the solution to converge before the information has propagated into the far field. Including the far field into the computation would therefore result in faulty results. This tool allows the user to define a specific

area to be kept. All cells outside this area will be removed. There are two types of domain reduction: a sphere-shaped and a box-shaped reductions. The sphere is defined by a center and a radius and the box is defined by a minimum and a maximum point. In $2D$, the sphere becomes a disk and the box a rectangle (here the z component must be set to zero).

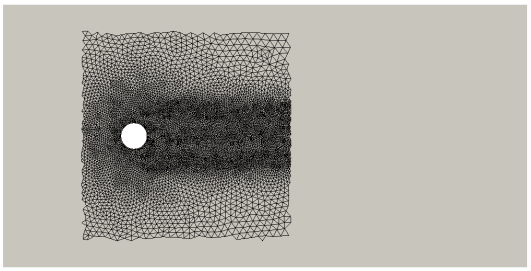
To exemplify this feature, the domain reduction is applied to this cylinder case. Both sphere and box reduction are shown on fig. 3.12.



(a) Original full mesh



(b) Sphere reduction with center $C = (2, 0, 0)$ and radius $r = 4$



(c) Box reduction with $min = (-2, -4, 0)$ and $max = (6, 4, 0)$

Figure 3.12: Domain reduction applied on the cylinder case.

Since it's a $2D$ case, all z coordinates of the reduction domain have to be set to zero. The reduction presented here was randomly chosen.

Warning: independently of the original mesh type, the resulting mesh will become *unstructured*. So if the mesh is structured, it will loose its computational advantage through the reduction process.

3.4.4 Galerkin expansion

This section details the input options of the Galerkin expansion, such as the number of POD modes and the type of basic mode. The parameters are listed in table 3.4.

Base flow

Input options:

Parameter	Required	Value	Subparam.	Value
BaseFlow	yes	MeanFlow		
		SteadyFlow	**SteadyFlowName**	<name>
FirstSnapshot	no	value		
LastSnapshot	no	value		
NPOD	yes	value		
ShiftMode	no	yes no		

Table 3.4: Parameters to control the Galerkin expansion

```
    BaseFlow                :    MeanFlow
or:
    BaseFlow                :    SteadyFlow
    SteadyFlowName          :    <name>
```

This option allows to choose between the mean and the steady flow for the base flow (also called basic mode u_0 in chapter 2). The mean flow is computed from the snapshots, but the steady flow have to be provided in the `./InputData/SteadyFlow/` folder and its name have to be provided under the `SteadyFlowName` keyword.

First snapshot, last snapshots (optional)

Input options:

```
    FirstSnapshot           :    <value>
    LastSnapshot            :    <value>
```

These keywords specify the first and last snapshot of the range of snapshots used to compute the mean flow. They only need to be defined if the mean flow is chosen as base flow. If they are not defined, all snapshots will be used. It is also possible to define only one of them. The mean is computed as follows:

- With all snapshots:

$$\boldsymbol{u}_0(\boldsymbol{x}) := \frac{1}{M} \sum_{m=1}^{M} \boldsymbol{v}^m(\boldsymbol{x})$$

- Using `FirstSnapshot` and `LastSnapshot`:

$$\boldsymbol{u}_0(\boldsymbol{x}) := \frac{1}{\mathsf{LastSnapshot} - \mathsf{FirstSnapshot} + 1} \sum_{m=\mathsf{FirstSnapshot}}^{\mathsf{LastSnapshot}} \boldsymbol{v}^m(\boldsymbol{x})$$

NPOD

Input options:

```
NPOD                    :    <value>
```

This defines how many POD modes will be computed and saved. It is *not* the number of POD modes used for the Galerkin projection. This one is defined by the keyword `NPODProjection` and presented in section 3.4.5. `NPOD` cannot be larger than the total number of snapshots, otherwise an error will be raised.

Shift mode (optional)

Input options:

```
ShiftMode               :    yes # or no
SteadyFlowName          :    <name>
```

This keyword allows to add the shift mode (as defined by [Noack et al., 2003]) in the Galerkin expansion. Since this shift mode requires the steady flow, its name have to be defined with `SteadyFlowName`. However, if the steady flow is already defined as a base flow, it is not necessary to define `SteadyFlowName` a second time.

3.4.5 Galerkin projection

The input options of the Galerkin projection are summarized in table 3.5.

Parameter	**Required**	**Value**	**Subparam.**	**Value**
Re	yes	<value>		
NPODProjection	yes	<value>		

Table 3.5: Input options for the Galerkin projection

Reynolds number

Input options:

```
Re                      :    <value>
```

The mass matrix and the convection tensor do not require any input options, whereas the viscosity matrix needs the Reynolds number to be computed.

Number of modes for the projection basis

Input options:

```
NPODProjection          :    <value>
```

It is possible to choose a different number of modes from that defined by `NPOD`. Note that `NPODProjection` cannot be larger than `NPOD`.

3.4.6 Dynamical system

The options for the dynamical system are the time integration settings (starting, ending times and time step) and the calibration type. The default ODE solver is the `odeint` solver from python, which uses `lsoda` from the FORTRAN library `odepack`. It is an efficient solver with an adaptive time step. The input options are summarized in table 3.6

Parameter	Required	Value	Subparam.	Value
tStartDS	no	<value>		
tEndDS	no	<value>		
dtDS	no	<value>		
Calibration	yes	<model>		

Table 3.6: Input options for the dynamical system integration

tStartDS, tEndDS, dtDS

Input options:

```
tStartDS        :   <value>
tEndDS          :   <value>
dtDS            :   <value>
```

These keywords allow to control the integration of the dynamical system. `tStartDS`, `tEndDS` and `dtDS` define respectively the starting and ending times of the integration and the time step. Their names end with `DS` for dynamical system, as opposed to the time parameters from pre-processing, which end with `Snap` and are only used to describe the input snapshots. The solution of the dynamical system will be written at every `dtDS` step.

Calibration

Input options:

```
Calibration     :   ModalNonLinear
```

`Calibration` allows to specify a calibration method for the POD ROM. There is currently only one method, a modal non-linear calibration, which is specified by the keyword `ModalNonLinear`. If no calibration is required, this line can be commented.

The modal non-linear subscale turbulence term is that of [Östh et al., 2013]. Note that the implemented calibration technique (modal nonlinear) does not guarantee stability of the dynamical system.

4. Example: ROM of a cylinder wake

In this chapter, we present an empirical Galerkin model of the cylinder wake. The effects of most of the parameters from the input file will be demonstrated.

4.1 Configuration

We consider the 2-D cylinder wake at $Re = 100$. The direct numerical simulation is performed with a FEM of third-order accuracy in space and time. The computational domain Ω_{DNS} is bounded by the rectangle $-5 < x < 15$ and $-5 < y < 5$, excluding the cylinder $\Omega_{\mathrm{cyl}} = \{(x, y) : x^2 + y^2 \leq 1/2\}$. Details can be inferred from Noack et al. [2003]. Figures 4.1 and 4.2 provide an impression of the domain, the grid and the periodic flow. In the ROM analysis, the observation domain $\Omega = \Omega_{\mathrm{DNS}}$ is identical to the computational domain.

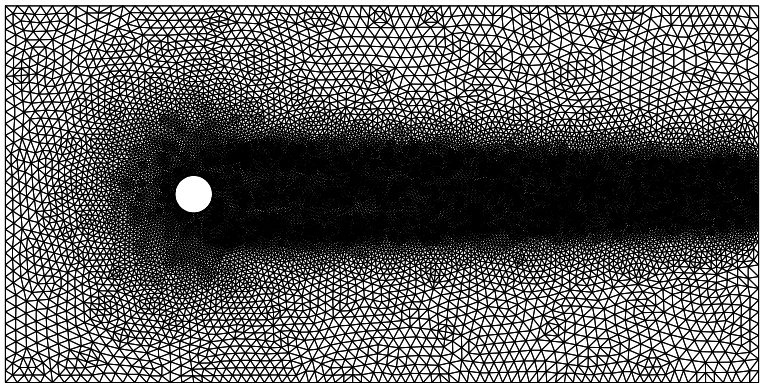


Figure 4.1: Unstructured grid for the FEM direct numerical simulation of cylinder wake.

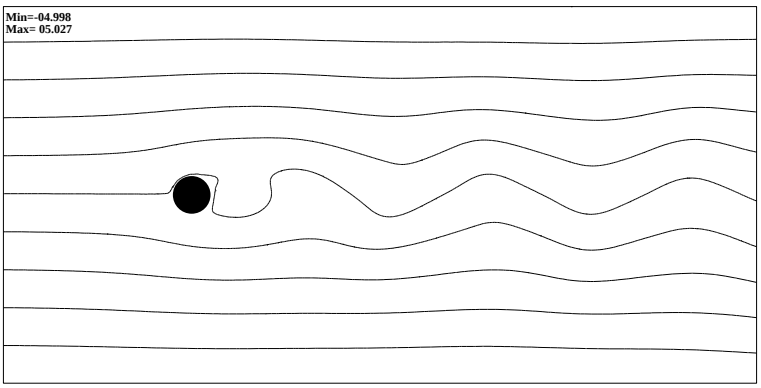


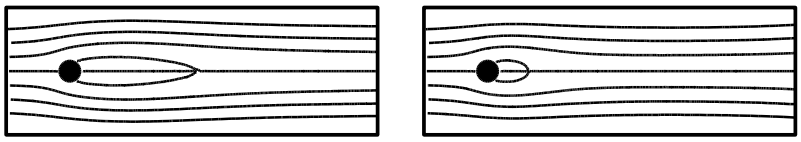
Figure 4.2: Streamlines of a snapshot.

4.2 Galerkin expansion

The POD decomposition with 8 modes resolves the periodic DNS solution,

$$\boldsymbol{u} = \boldsymbol{u}_0 + \sum_{i=1}^{8} a_i \boldsymbol{u}_i.$$

The basic mode $\boldsymbol{u}_0$ is visualized in Fig. 4.3b, the POD modes $\boldsymbol{u}_i$, $i = 1, \ldots, 8$ in Fig. 4.4, the amplitude evolutions over one period a_i, $i = 1, \ldots, 8$ in Fig. 4.5, and the POD spectrum λ_i, $i = 1, \ldots, 8$ in Fig. 4.6.



(a) $\boldsymbol{u}_1$ (b) $\boldsymbol{u}_2$

Figure 4.3: Base flows: steady solution (a) and mean flow (b). The flow is visualized with streamlines in a subdomain.

The steady solution (see Fig. 4.3a) is included in the Galerkin expansion via the shift mode [Noack et al., 2003] as the 9th mode (see Fig. 4.7).

4.3 Galerkin system

We display numerical values of the Galerkin system for selected indices of the most relevant expansion modes, namely the first harmonics $i = 1, 2$ and the shift mode $i = 9$. These expansions are needed for a least-order Galerkin model. Table 4.1 displays the mass matrix m_{ij} and the viscous term l^{ν}_{ij}. The mass matrix is up to numerical resolution the identity matrix. The viscous matrix is approximately a diagonal matrix with negative values. The dissipation increases with associated order of the harmonics.

Table 4.2 and 4.3 displays selected values of q^{c}_{ijk} associated with the convective term. The interactions of POD modes yield values up to order $O(0.01)$, the interaction of POD modes with the shift mode increases the order to $O(0.1)$ while interactions with the base flow $\boldsymbol{u}_0$ may increase the order further to $O(1)$. Interpretations and explanations of the coefficients are provided in mean-field Galerkin models [Noack et al., 2003, Tadmor et al., 2010]. The pressure term is neglected in agreement with literature starting with [Deane et al., 1991].

4.4 Dynamical system

Table 4.4 lists the non-vanishing dynamical system coefficients associated with a least-order Galerkin mode. Figures 4.8, 4.9 and 4.10 display a numerical solution of the dynamic system derived from the POD wake model.

Table 4.1: Mass matrix (left) and viscous term (middle): numerical values for the most relevant expansion modes $i, j \in \{1, 2, 9\}$. The complete list of diagonal terms of the viscous matrix is shown right.

i	j	m_{ij}
1	1	1.000000
1	2	0.000000
1	9	0.000000
2	1	0.000000
2	2	1.000000
2	9	0.000000
9	1	0.000000
9	2	0.000000
9	9	1.000000

i	j	l^{ν}_{ij}
1	1	-2.520396
1	2	0.059919
1	9	0.001294
2	1	0.059919
2	2	-2.635341
2	9	0.001687
9	1	0.001294
9	2	0.001687
9	9	-3.119646

i	j	l^{ν}_{ij}
1	1	-2.520396
2	2	-2.635341
3	3	-9.416582
4	4	-9.421484
5	5	-16.123679
6	6	-16.119000
7	7	-31.647050
8	8	-31.612550
9	9	-3.119646

Table 4.2: Convective term: numerical values for the most relevant expansion modes $i, j, k \in \{1, 2, 9\}$.

i	j	k	q^{c}_{ijk}
1	1	1	-0.000000
1	1	2	0.000003
1	1	9	-0.018066
1	2	1	0.000003
1	2	2	0.000001
1	2	9	-0.045663
1	9	1	0.000497
1	9	2	0.113913
1	9	9	0.000002

i	j	k	q^{c}_{ijk}
2	1	1	0.000001
2	1	2	-0.000004
2	1	9	0.045832
2	2	1	-0.000008
2	2	2	0.000001
2	2	9	-0.020797
2	9	1	-0.114078
2	9	2	0.000049
2	9	9	0.000009

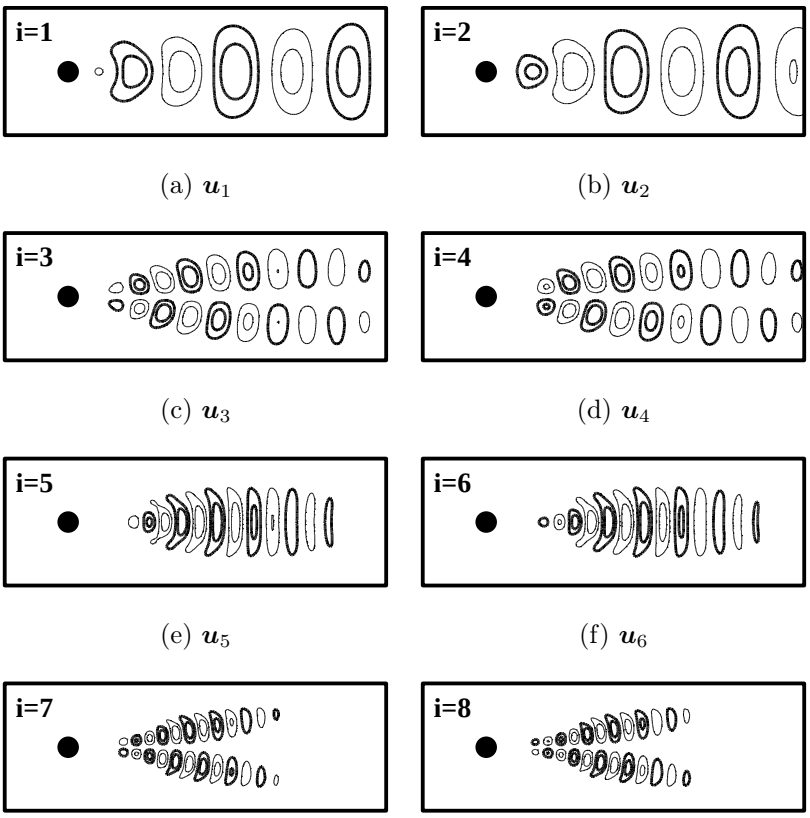
i	j	k	q^{c}_{ijk}
9	1	1	0.017931
9	1	2	-0.045746
9	1	9	-0.000012
9	2	1	0.045376
9	2	2	0.021237
9	2	9	0.000011
9	9	1	-0.000002
9	9	2	-0.000012
9	9	9	0.000085

Table 4.3: Convective term: numerical values for interactions of the most relevant expansion modes $i, j, k \in \{1, 2, 9\}$ with the base flow j or $k = 0$.

i	j	k	q^{c}_{ijk}
1	0	0	-0.000087
1	0	1	-0.027049
1	0	2	1.024398
1	0	9	0.000196
1	1	0	0.057408
1	2	0	0.074766
1	9	0	0.000028

i	j	k	q^{c}_{ijk}
2	0	0	0.003284
2	0	1	-0.997404
2	0	2	-0.014182
2	0	9	-0.000309
2	1	0	-0.078424
2	2	0	0.048725
2	9	0	0.000350

i	j	k	q^{c}_{ijk}
9	0	0	-0.115342
9	0	1	-0.000151
9	0	2	0.000312
9	0	9	-0.007247
9	1	0	-0.000044
9	2	0	-0.000267
9	9	0	-0.009784



(a) $\boldsymbol{u}_1$ (b) $\boldsymbol{u}_2$

(c) $\boldsymbol{u}_3$ (d) $\boldsymbol{u}_4$

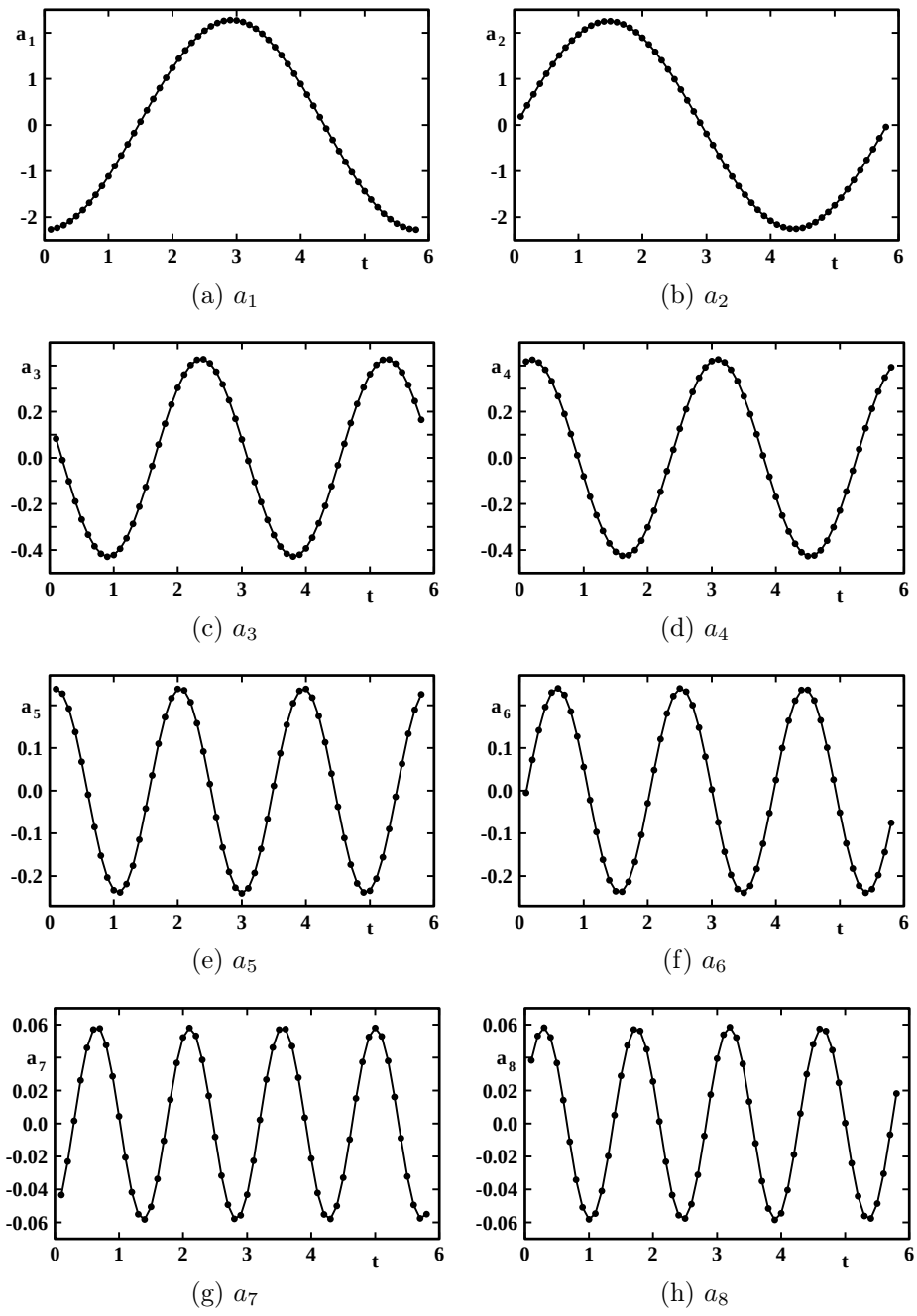
(e) $\boldsymbol{u}_5$ (f) $\boldsymbol{u}_6$

(g) $\boldsymbol{u}_7$ (h) $\boldsymbol{u}_8$

Figure 4.4: POD modes $i = 1, \ldots, 8$

Table 4.4: Dynamical system: complete list of non-vanishing numerical values of q^+_{ijk} $i, j, k \in \{0, 1, 2, 9\}$ and $i \neq 0$.

i	j	k	q^+_{ijk}	i	j	k	q^+_{ijk}	i	j	k	q^+_{ijk}
1	0	0	0.000011	2	0	0	0.003283	9	0	0	-0.108340
1	1	0	0.005155	2	1	0	-1.075229	9	1	0	-0.000181
1	1	1	-0.000000	2	1	1	0.000001	9	1	1	0.017931
1	2	0	1.099763	2	2	0	0.008189	9	2	0	0.000061
1	2	1	0.000005	2	2	1	-0.000012	9	2	1	-0.000370
1	2	2	0.000001	2	2	2	0.000001	9	2	2	0.021237
1	9	0	0.000236	2	9	0	0.000058	9	9	0	-0.048227
1	9	1	-0.017570	2	9	1	-0.068246	9	9	1	-0.000015
1	9	2	0.068251	2	9	2	-0.020749	9	9	2	-0.000000
1	9	9	0.000002	2	9	9	0.000009	9	9	9	0.000085



(a) a_1 (b) a_2

(c) a_3 (d) a_4

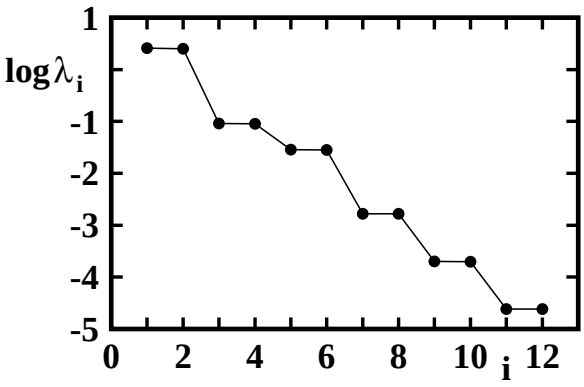
(e) a_5 (f) a_6

(g) a_7 (h) a_8

Figure 4.5: Amplitude evolution a_i, $i = 1, \ldots, 8$ corresponding to the POD modes of Fig. 4.4

4.5 Comment

These results used here are based on the same snapshots as in Noack et al. [2003]. However, the Galerkin projection of this reference was based on third-order accurate differentiation and integration. The numerics of the current `xROM` version is of one order lower accuracy.



i	λ_i
1	2.586698
2	2.527115
3	0.091202
4	0.090478
5	0.028538
6	0.028444
7	0.001672
8	0.001669

Figure 4.6: POD spectrum. The left figure displays the range $i = 1, \ldots, 12$ and the right table lists the values of the employed modes.

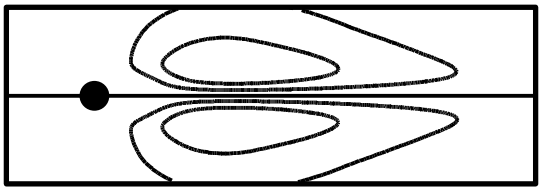


Figure 4.7: Shift mode $\boldsymbol{u}_9$. The flow is visualized in a subdomain by streamlines.

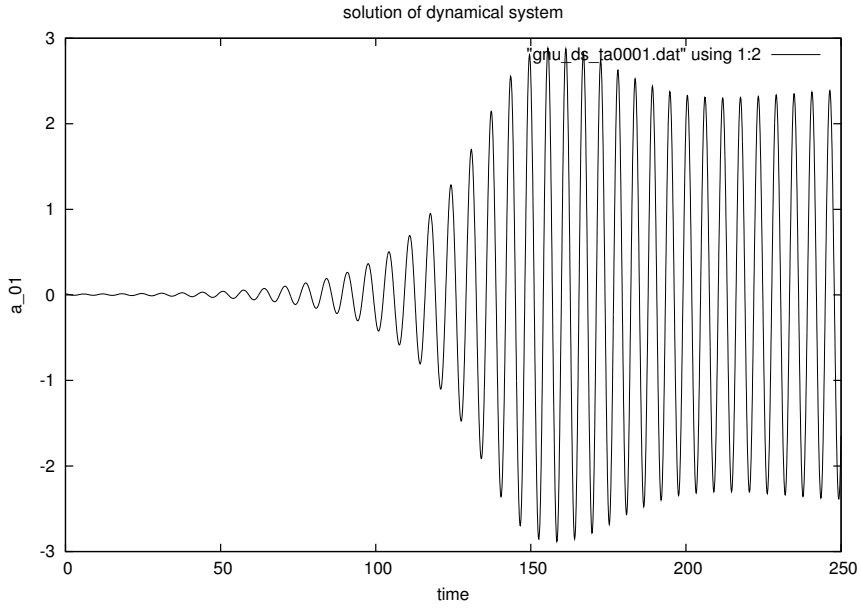


Figure 4.8: Evolution of mode amplitude a_1 of the first POD mode on $t \in [0, 250]$ as predicted by the dynamical system with $N = 9$. The initial condition at $t = 0$ is $a_i = 0.01\delta_{i1}$, $i = 1, \ldots, 9$.

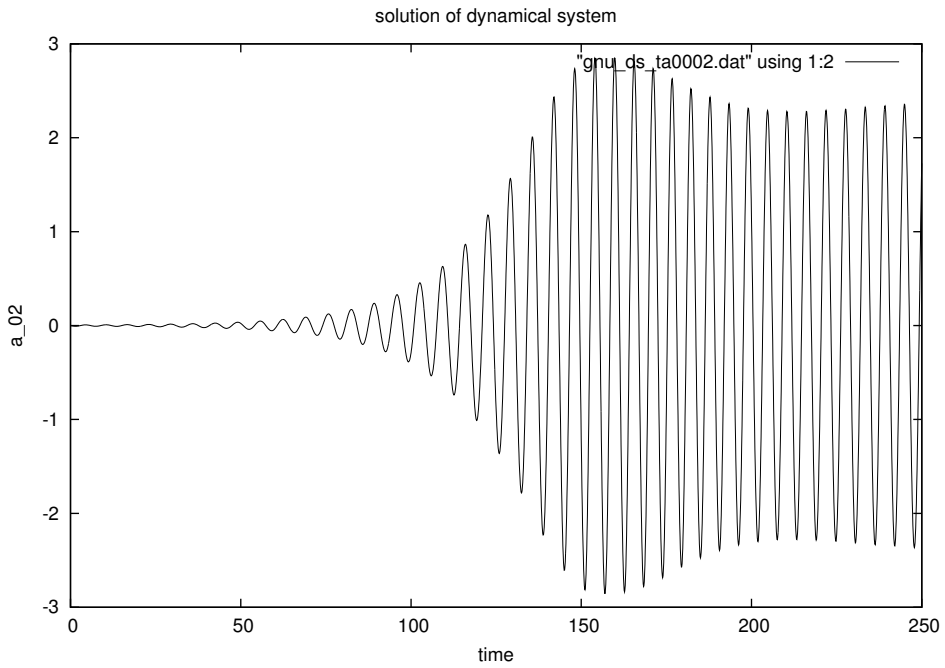


Figure 4.9: Same as figure 4.8, but for the second amplitude a_2.

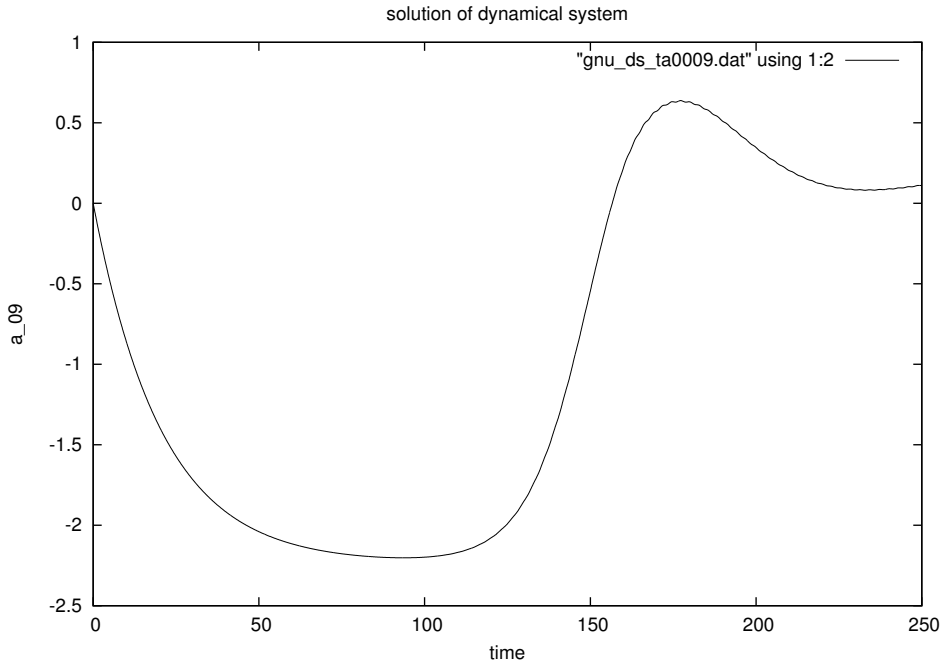


Figure 4.10: Same as figure 4.8, but for the shift-mode amplitude a_9.

A. ConfigFile used for the ROM of chapter 4

```
# Steps to execute
# ~~~~~~~~~~~~~~~~
  GalerkinExpansion : yes          # (yes, no)
 GalerkinProjection : yes          # (yes, no)
    DynamicalSystem : yes          # (yes, no)
    #useExistingRun : continue     # (continue, copy)
             #useRun : Run6

# Pre-processing
# ~~~~~~~~~~~~~~
           #RunName : cylinder-15POD
         DataFormat : cgns              # (cgns, dat)
         tStartSnap : 0                 # [s]
           tEndSnap : 5.7               # [s]
             dtSnap : 0.1               # [s]

           #GridSpec : Equidistant      # (Cartesian, Equidistant)

                #PIV : LaVision         # (LaVision, DynamicStudio)

       #reduceDomain : Sphere           # (Sphere, Box)
             #Center : (2,0,0)
             #Radius : 2
           #MinPoint : (-3,-3,0)
           #MaxPoint : ( 3, 3,0)

# Galerkin Expansion
# ~~~~~~~~~~~~~~~~~~
            BaseFlow : MeanFlow     # (MeanFlow, SteadyFlow)
     #SteadyFlowName : steady.cgns
     # FirstSnapshot : 10
     #  LastSnapshot : 25

                NPOD : 8
           ShiftMode : yes          # (yes, no)

# Galerkin Projection
# ~~~~~~~~~~~~~~~~~~~
                  Re : 100
      NPODProjection : 8

# Dynamical System
# ~~~~~~~~~~~~~~~~
```

```
            tStartDS : 0              # [s]
              tEndDS : 250             # [s]
                dtDS : 0.01           # [s]
```

Bibliography

S. L. Brunton and B. R. Noack. Closed-loop turbulence control: progress and challenges. *Applied Mechanics Reviews*, 67(5):050801, 2015.

A. E. Deane, I. G. Kevrekidis, G. E. Karniadakis, and S. A. Orszag. Low-dimensional models for complex geometry flows: Application to grooved channels and circular cylinders. *Phys. Fluids A*, 3:2337–2354,, 1991.

C. A. J. Fletcher. *Computational Galerkin Methods*. Springer, New York, 1st edition, 1984.

P. Holmes, J. L. Lumley, G. Berkooz, and C. W. Rowley. *Turbulence, Coherent Structures, Dynamical Systems and Symmetry*. Cambridge University Press, Cambridge, 2nd paperback edition, 2012.

H. J. Lugt. *Introduction to Vortex Theory*. Vortex Flow Press, Potomac (Maryland, USA), 1996.

B. R. Noack and L. Cordier. xAMC — a Toolkit for Analysis, Modeling and Control of Fluid Flows (Version 3.0). Technical Report 2012-01, CEAT, Institute PPRIME, France, 2012.

B. R. Noack, K. Afanasiev, M. Morzyński, G. Tadmor, and F. Thiele. A hierarchy of low-dimensional models for the transient and post-transient cylinder wake. *J. Fluid Mech.*, 497: 335–363, 2003.

B. R. Noack, G. Tadmor, and M. Morzyński. Low-dimensional models for feedback flow control. Part I: Empirical Galerkin models. *AIAA Paper 2004-2408 (invited contribution)*, June 28 – July 1 2004.

B. R. Noack, M. Morzynski, and G. Tadmor. *Reduced-Order Modelling for Flow Control*. Springer Science & Business Media, 2011.

J. Östh, B. R. Noack, S. Krajnović, D. Barros, and J. Borée. On the need for a nonlinear subscale turbulence term in pod models as exemplified for a high-Reynolds-number flow over an Ahmed body. *J. Fluid Mech.*, 747:518–544, 2013.

R. Semaan, P. Kumar, M. Burnazzi, G. Tissot, L. Cordier, and B. R. Noack. Reduced-order modelling of the flow around a high-lift configuration with unsteady coanda blowing. *J. Fluid Mech.*, 800:72–110, 2016.

G. Tadmor, O. Lehmann, B. R. Noack, and M. Morzyński. Mean field representation of the natural and actuated cylinder wake. *Phys. Fluids*, 22(3):034102–1..22, 2010.