Institut P'

CNRS – Université de Poitiers – ENSMA, UPR 3346
Département Fluides, Thermique, Combustion, CEAT

Technical Report 01/2012

xAMC — a Toolkit for Analysis, Modeling and Control of Fluid Flows (Version 3.0)

Bernd R. Noack & **Laurent Cordier**

1 February 2012

43 rue de l'Aérodrome, F-86036 POITIERS Cedex, France

Abstract

This report describes a software package `xAMC` for analysis, modeling and control of fluid flows with steady domains. We discuss standard operating procedures for a fairly 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 `xAMC` for eXecute, Analyis, Modeling and Control. 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 `xAMC` package which allows to reproduce the Galerkin models of **?**.

This 2nd edition of the 2010 report includes details about the format of the flow data and xAMC's internal representation of that data on different grids.

Keywords: proper orthogonal decomposition, Galerkin model, flow control

Contents

1. Introduction

Dynamic reduced-order models (ROM) for understanding and controlling fluid flows are a rapidly evolving area. 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 [?]. 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 one framework for low-order linear and nonlinear Galerkin models. These models gained increasing popularity [?]. Successes include explanations for onset of oscillations and various forms of transition to turbulence.

Meanwhile reduced-order modeling is dominated by empirical Galerkin models [?]. 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, *the toolbox* and numerical programs *the toolkit* and a meticulous order in the file system. Our corresponding endeavor is termed `xAMC` for 'eXecute Analysis, Modeling and Control'. xAMC assumes a multi-author, multi-source, multi-data and multi-Linux platform. In short, we assume a hectic, chaotic research environment of amateur programmers. The implications of these assumptions on the programs and data will be detailed in the report.

The report has four main parts. Part I explains the toolbox, i.e. the SOP from data to the dynamical model. This procedure is exemplified with a cylinder wake model [?]. After this part, the reader may be able to judge if `xAMC` addresses his/her problem. Part II explains the execution of this pipeline for the wake model from installation, execution and first explanation of `xAMC`. This part gives the reader a hands-on feeling of `xAMC`. Part III explains how to apply `xAMC` to construct new Galerkin models from implemented or new flow data. This gives the user the information to apply the program to his/her new problem. The final part IV details the `xAMC` programming philosophy and file sytem. This part is intendend for collaborators who want to participate in the development. In Part V, we conclude with the next action items of `xAMC` development.

Part I

xAMC capabilities

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. Without actuation, 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

$$\begin{aligned} \nabla \cdot \boldsymbol{u} &= 0, & (2.2)\\ \partial_t \boldsymbol{u} + \nabla \cdot (\boldsymbol{u} \otimes \boldsymbol{u}) &= -\nabla p + \nu \triangle \boldsymbol{u} + \sum_{l=1}^{N_V} b_l\, \boldsymbol{g}_l. & (2.3) \end{aligned}$$

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.

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 ??, 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

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

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 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:

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

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

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

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

2.4 Galerkin projection

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

$$\underbrace{\sum_{j=-N_A}^{N} m_{ij}\,\dot{a}_j}_{\text{class}\quad 0} = \underbrace{\nu \sum_{j=-N_A}^{N} l_{ij}^{\nu}\,a_j + \sum_{j=-N_A}^{N} l_{ij}^{+} a_j}_{\text{class}\quad 1} + \underbrace{\sum_{j,k=-N_A}^{N} \left(q_{ijk}^{c} + q_{ijk}^{p}\right)\,a_j\,a_k}_{\text{class}\quad 2}, \quad i = 1, \ldots, N, \tag{2.17}$$

where

$$\begin{aligned} m_{ij} &= (\boldsymbol{v}_i, \boldsymbol{u}_j)_\Omega\,, && (2.18a)\\ l_{ij}^{\nu} &= (\boldsymbol{v}_i, \triangle \boldsymbol{u}_j)_\Omega\,, && (2.18b)\\ q_{ijk}^{c} &= (\boldsymbol{v}_i, \nabla\cdot(\boldsymbol{u}_j \boldsymbol{u}_k))_\Omega\,, && (2.18c)\\ q_{ijk}^{p} &= (\boldsymbol{v}_i, -\nabla p_{jk})_\Omega\,. && (2.18d) \end{aligned}$$

In case of a POD model [?], 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 the 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,

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

Closed-loop control implies

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

We assume that the control law respects that the actuation mode amplitude a_l and its derive $\dot{a}_l$, $l \in \{-N_A, \ldots, -1\}$, 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 $N^+ = N + N_V + 2N_A$:

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

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

$$\dot{a}_i \;=\; f_i^+ \left(a_1, \ldots, a_{N^+}\right), \quad i = 1, \ldots, N \tag{2.22a}$$
$$a_i \;=\; f_i^+ \left(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_N\right) \quad i = N+1, \ldots, N^+ \tag{2.22b}$$

where f_i^+, $i = 1, \ldots, N$ is a polynomial of a_i of second order. In many examples, the control laws $a_i = f_i^+$, $i = N+1, \ldots, N^+$ 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. An example: a cylinder wake ROM

In this chapter, we present an empirical Galerkin model of the cylinder wake. This model will be computed with xAMC in Chapter 6.

3.1 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 $\Omega_{\text{cyl}} = \{(x, y) : x^2 + y^2 \leq 1/2\}$. Details can be inferred from ?. Figures 3.1 and 3.2 provide an impression of the domain, the grid and the periodic flow. In the ROM analysis, the observation domain $\Omega = \Omega_{\text{DNS}}$ is identical with the computational domain.

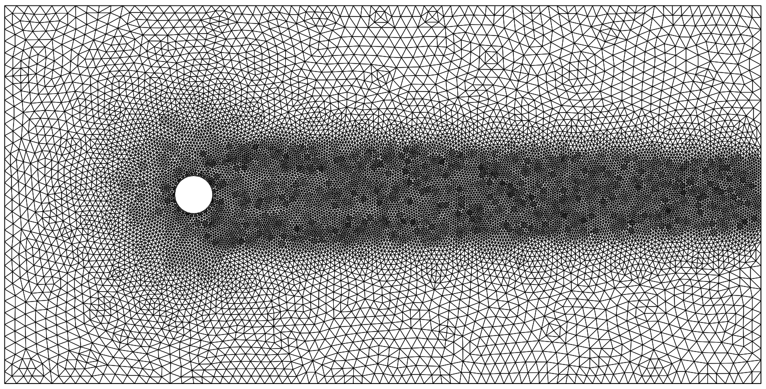


Figure 3.1: Unstructured grid for the FEM direct numerical simulation of cylinder wake.

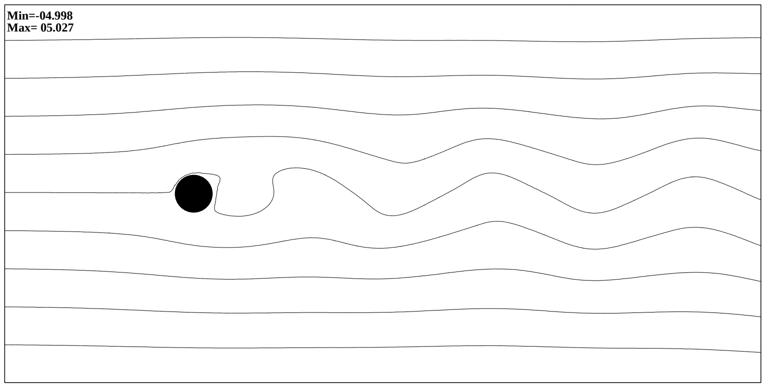


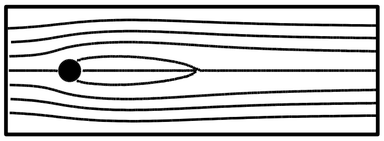
Figure 3.2: Streamlines of a snapshot.

3.2 Galerkin expansion

The POD decomposition with 8 modes resolves the periodic DNS solution (see Fig. 3.2),

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

The basic mode $\boldsymbol{u}_0$ is visualized in Fig. 3.3(b), the POD modes $\boldsymbol{u}_i$, $i = 1, \dots, 8$ in Fig. 3.4, the amplitude evolutions over one period a_i, $i = 1, \dots, 8$ in Fig. 3.5, and the POD spectrum λ_i, $i = 1, \dots, 8$ in Fig. 3.6.


(a) $\boldsymbol{u}_1$

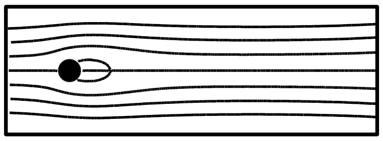
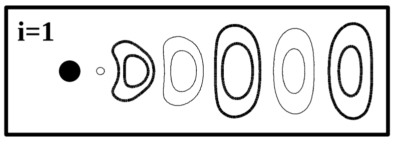
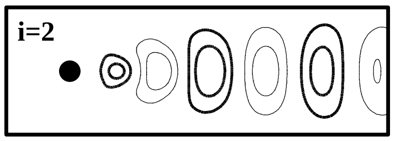

(b) $\boldsymbol{u}_2$

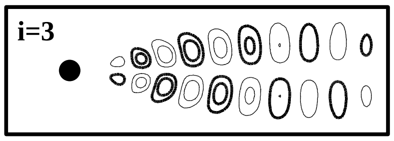
Figure 3.3: Base flows: steady solution (a) and mean flow (b). The flow is visualized with streamlines in a subdomain.



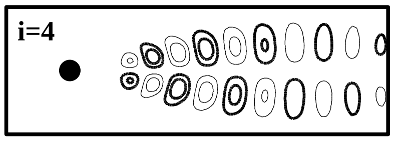
(a) $\boldsymbol{u}_1$



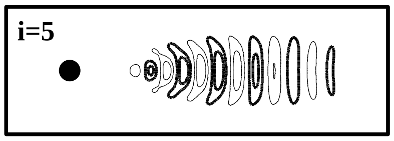
(b) $\boldsymbol{u}_2$



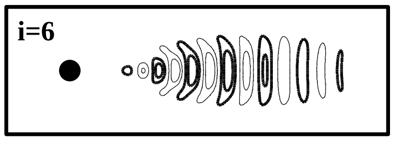
(c) $\boldsymbol{u}_3$



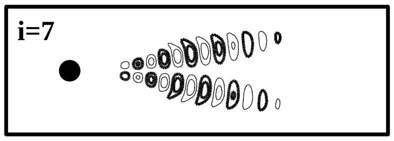
(d) $\boldsymbol{u}_4$



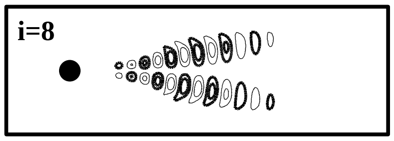
(e) $\boldsymbol{u}_5$



(f) $\boldsymbol{u}_6$



(g) $\boldsymbol{u}_7$



(h) $\boldsymbol{u}_8$

Figure 3.4: POD modes $i = 1, \dots, 8$

The steady solution (see Fig. 3.3(a)) is included in the Galerkin expansion via the shift mode [?] as 9th mode (see Fig. 3.7).

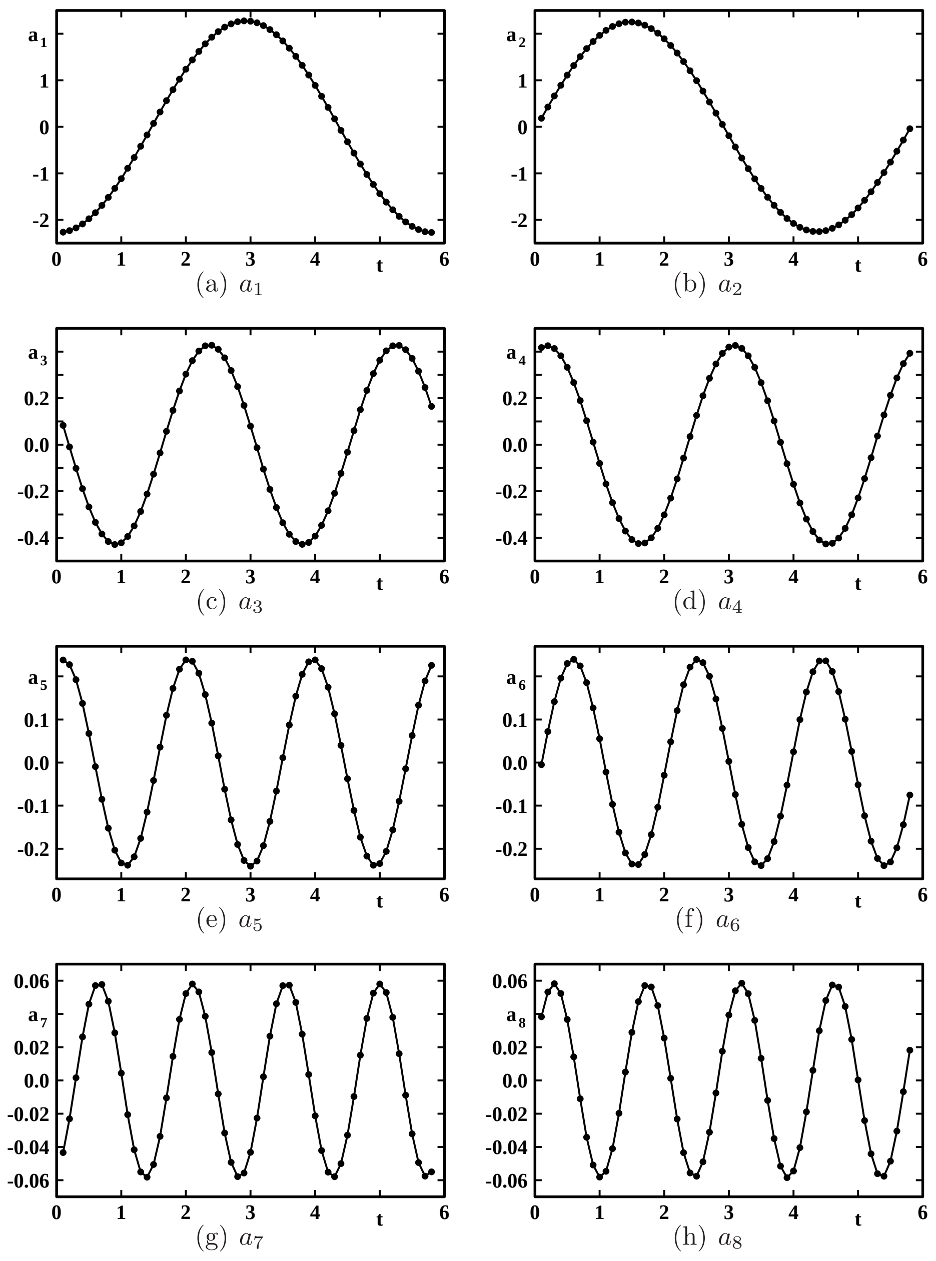
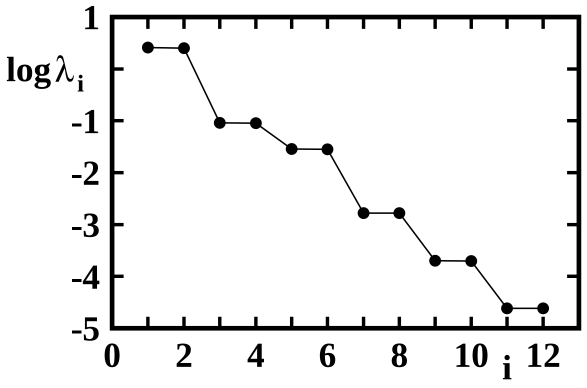


Figure 3.5: Amplitude evolution a_i, $i = 1, \ldots, 8$ corresponding to the POD modes of Fig. 3.4

3.3 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 3.1 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 3.2 and 3.3 displays selected values of q^{c}_{ijk} associated with the convective term. The



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

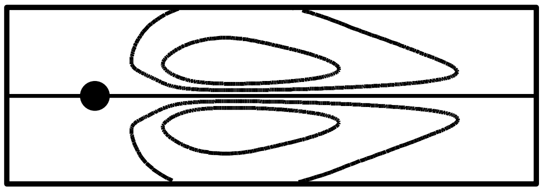


Figure 3.7: Shift mode $\boldsymbol{u}_9$. The flow is visualized in a subdomain by streamlines.

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 [??]. The pressure term is neglected in agreement with literature starting with [?].

Table 3.1: 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 3.2: 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
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
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 3.3: 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
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
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

Table 3.4: 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}
1	0	0	0.000011
1	1	0	0.005155
1	1	1	-0.000000
1	2	0	1.099763
1	2	1	0.000005
1	2	2	0.000001
1	9	0	0.000236
1	9	1	-0.017570
1	9	2	0.068251
1	9	9	0.000002
2	0	0	0.003283
2	1	0	-1.075229
2	1	1	0.000001
2	2	0	0.008189
2	2	1	-0.000012
2	2	2	0.000001
2	9	0	0.000058
2	9	1	-0.068246
2	9	2	-0.020749
2	9	9	0.000009
9	0	0	-0.108340
9	1	0	-0.000181
9	1	1	0.017931
9	2	0	0.000061
9	2	1	-0.000370
9	2	2	0.021237
9	9	0	-0.048227
9	9	1	-0.000015
9	9	2	-0.000000
9	9	9	0.000085

3.4 Dynamical system

Table 3.4 lists the non-vanishing dynamical system coefficients associated with a least-order Galerkin mode. Figures 3.8, 3.9 and 3.10 display a numerical solution of the dynamic system derived from the POD wake model. These results will be reproduced in Sec. 6.

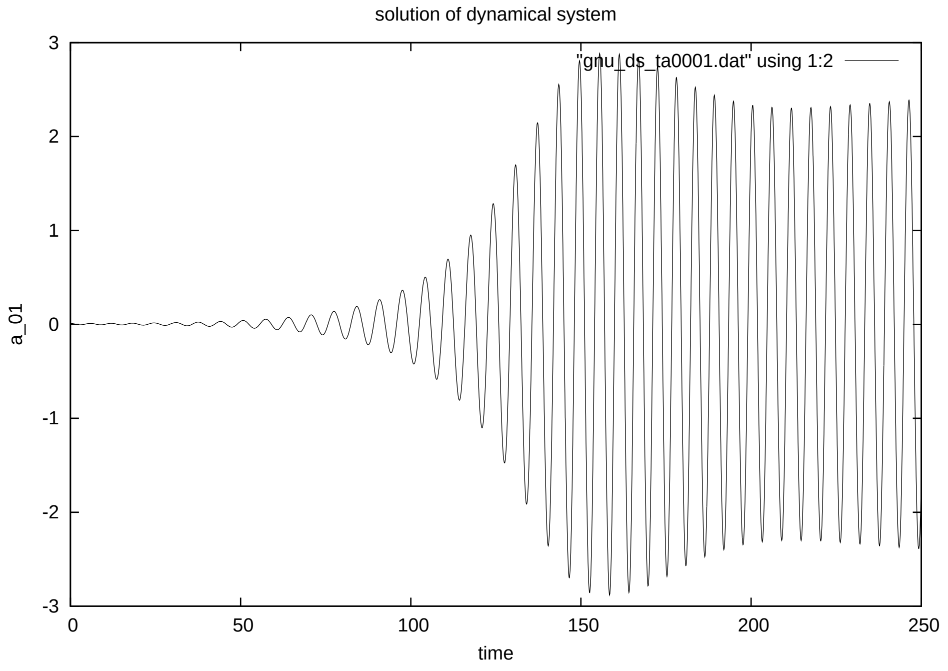


Figure 3.8: Evolution of mode amplitude a_1 of the first POD mode on $t \in [0, 250]$ as predicted by the dynamical system with $N = 9$. The initial condition at $t = 0$ is $a_i = 0.01\delta_{i1}$, $i = 1, \ldots, 9$.

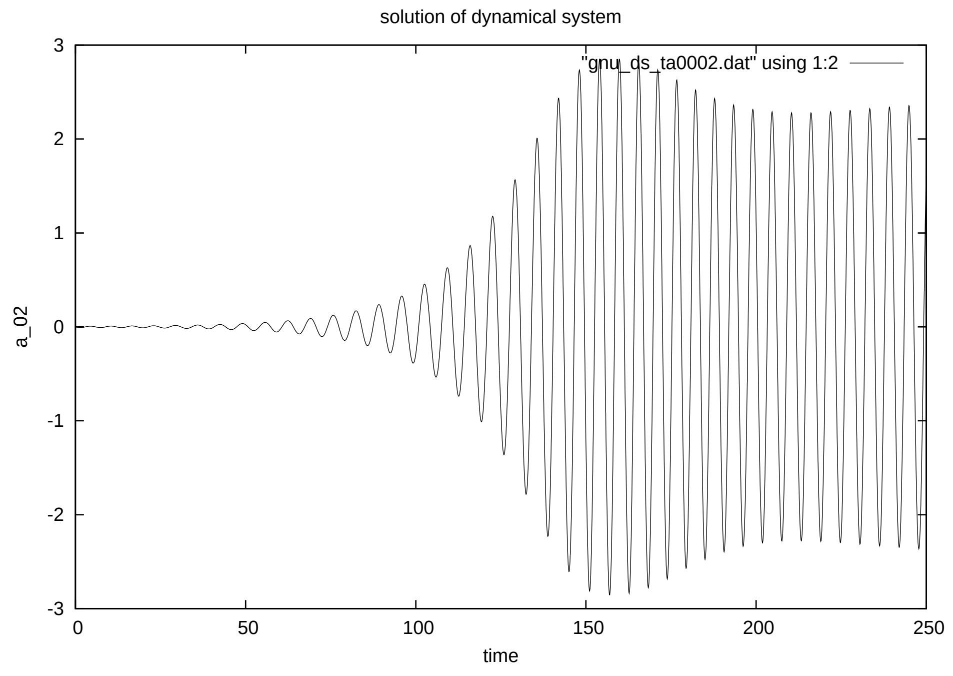


Figure 3.9: Same as figure 3.8, but for the second amplitude a_2.

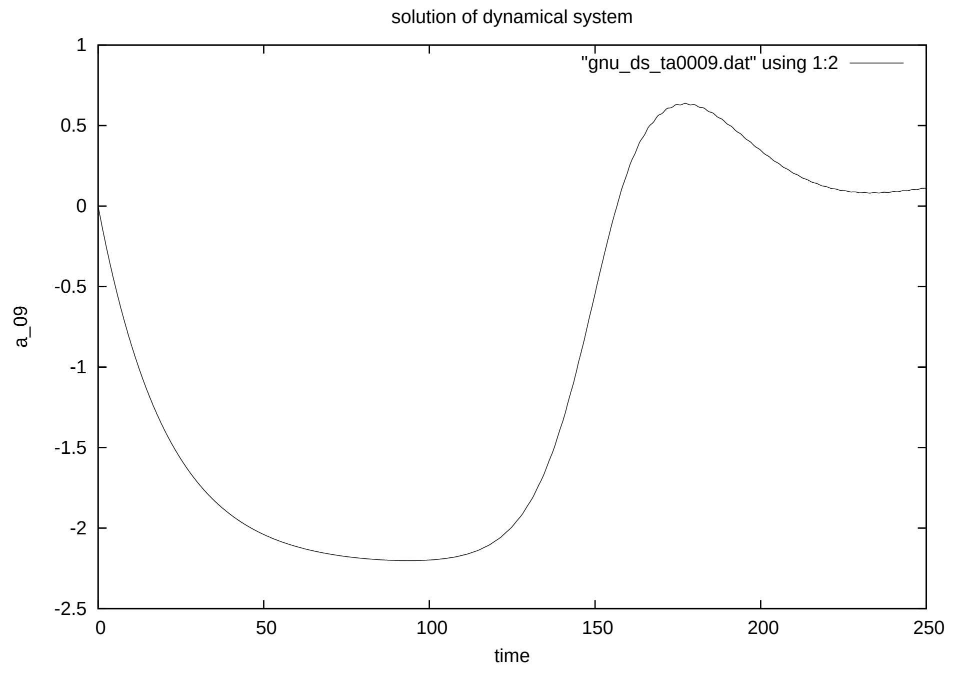


Figure 3.10: Same as figure 3.8, but for the shift-mode amplitude a_9.

3.5 Comment

In Sec. 6, the post-transient Galerkin solution of this example is reproduced starting with a point on the limit cycle. In order to reproduce the transients of this chapter, `y3_State.dat` and y3_Time.dat should be taken from paragraph 4 and 5 of Sec. 8.1. See this and later sections for more information.

These results used here are based on the same snapshots as in ?. However, the Galerkin projection of this reference was based on third-order accurate differentiation and integration. The numerics of the current xAMC version is of one order lower accuracy.

4. Outline of xAMC

This chapter provides a brief outline of xAMC file system and the processes invoked by the main shell script xAMC. The outline shall help to understand the execution of an example ROM in chapter 6.

4.1 Main directories

xAMC separates three categories of files in separate directories (see Fig. 4.1):

xRun is the main directory for program execution. This directory hosts the main shell script xAMC which constructs a ROM with a single command as discussed below. It copies or links files from the following directories.

xTools contains different tools for the execution of algorithms. Each group of tools has its own subdirectory. Examples are listed below:

- x0_GM contains programs acting on the raw data, e.g. for grid or snapshot visualization. Note that different ROMs can be constructed from the same data, e.g. a POD- or DMD-based Galerkin models may use the same snapshots. Moreover, each ROM will generally require different tools. Each subdirectory contains one soft link to the flow data.
- x1_GM contains the programs for the computation of POD, DMD, etc.
- x2_GM effects the Galerkin projection.
- x3_DS computes the solution of the Galerkin system.

yGM contains process-related files, i.e. links the tools with the data for the creation of ROMs. Each subdirectory contains the files for one particular ROM. One example is listed below:

- Noack_2003jfm contains the recipe for creating the 9-dimensional Galerkin model of Noack et al. ?.

zData contains the flow data, one subdirectory for each configuration. The main ingredients may be snapshots for a POD-based ROM or stability eigenmodes for a physical ROM. zData may be a real directory or an imagined directory representing various data locations, e.g. a RAID server, a PC or a harddisk.

This separation between tools, processes and data is a frequently employed concept in scientific programming. Relating to the preparation of a meal, one may consider the tools as the kitchen utensils, the processes as the recipes, and data as the ingredients.

4.2 Main shell script

The ROM is constructed in the xRun directory by a single command, e.g.

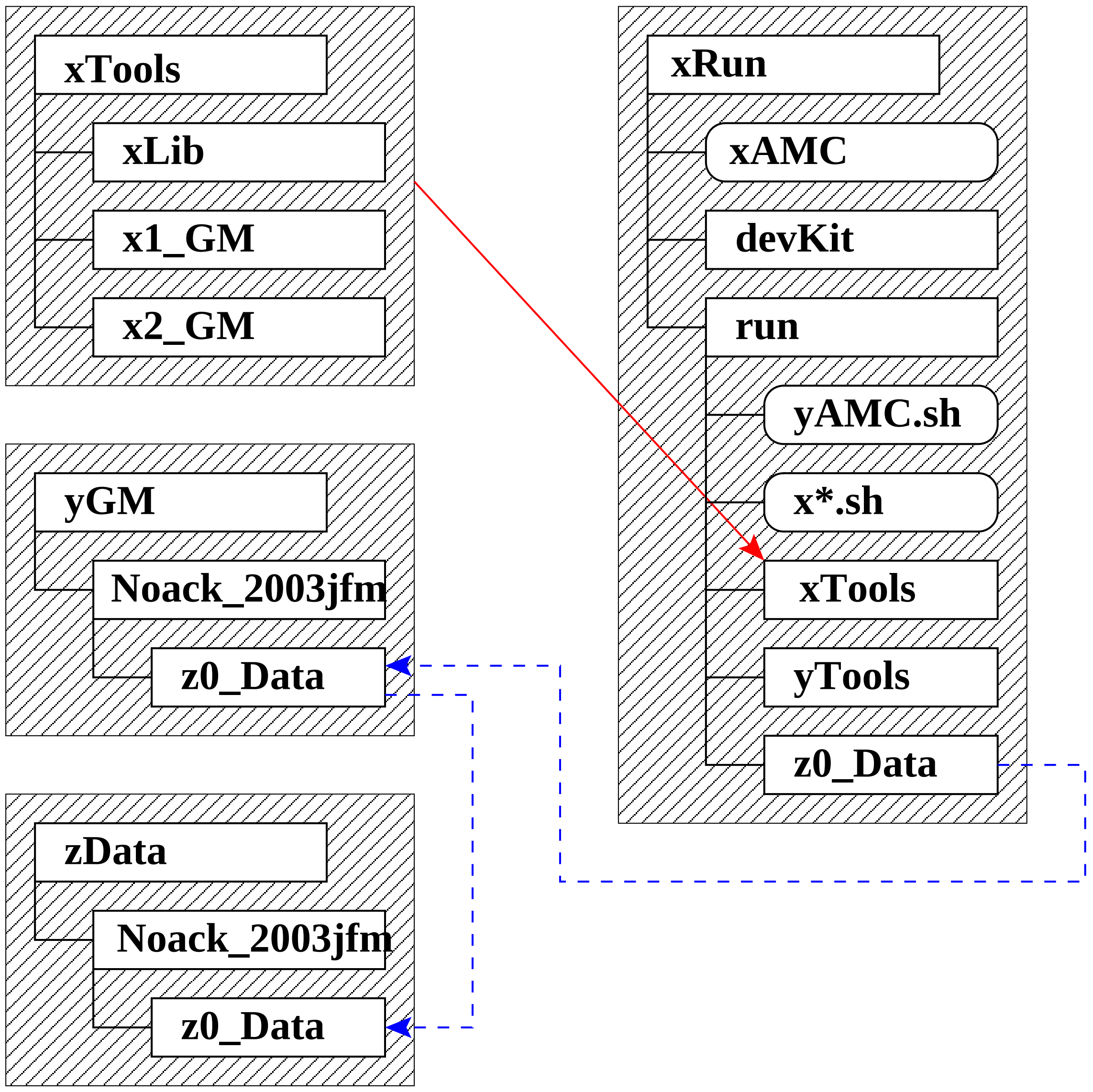


Figure 4.1: Principal (simplified) sketch of the structure of xAMC toolkit. Rounded boxes correspond to shell scripts. The rectangular boxes correspond to directories. The subdirectories are connected to upper-level one via lines. Files are represented by rounded rectangles. The blue line represents a softlink to the directory indicated by the arrow. The red line reminds that subset of the tools is copied in the `run/xTools` directory.

```
> xAMC ../yGM/Noack_2003jfm
```

This command invokes a number of processes:

1. A new `run`-directory is created in `xRun`.
2. The tools, e.g. `x1_POD`, are copied in the `run`-directory and saved in the `run/xTools`-subdirectory.

3. The corresponding binaries, e.g. `Y1_POD`, are created and saved in the `run/yTools`-subdirectory.

4. The process files from the `yGM`-subdirectory, e.g. `yAMC.sh` and `y1_GM.dat`, are copied in the `run` directory (`yGM/Noack_2003jfm` in the considered example).

5. The flow data is made accessible from the `run`-directory via a soft link.

6. Now, all ROM-related files are in the `run`-directory. `xAMC` executes the master shell script `yAMC.sh` copied from the `yGM/Noack_2003jfm`-subdirectory.

7. This master-shell script executes the necessary tools (shell scripts), e.g. one script `x3_solveDS` for the Galerkin system solution.

8. Each tool-related shell script executes the binary.

9. The executable programs read the flow data and processes it corresponding to the contents of the process files. It produces ROM-related data, e.g. the modes or the Galerkin system. Or the executable post-processes ROM-data, e.g. the Galerkin solution from the Galerkin system.

10. All ROM-related data are now in the `run`-directory. The run directory contains also all tools (sources and binaries). Only the data is not copied but accessed via a soft-link to save hard disk space.

4.3 Nomenclature of files

The file names generally obey following rules:

1. The suffixes follow general rules:
   - '`.f`' for a FORTRAN file,
   - '`.include`' for a FORTRAN segment,
   - '`.input`' for a FORTRAN keyboard (input) file,
   - '`.gnuplot`' for a Gnuplot script,
   - '`.sh`' for a shell script,
   - '`.dat`' for a data file, etc.

2. The prefixes of tool-, process- and data-related files start with '`x`', '`y`' and '`z`' respectively. Binary executables start with '`Y`'.

3. The number behind the first character indicates the level of an important source or data file, i.e.
   - '0' for raw data, e.g. `x0_plotGrid.sh`;
   - '1' for the Galerkin expansion, i.e. kinematic compression of the data, e.g. `x1_GalerkinApproximation.sh`;

  - '2' for the Galerkin system, i.e. dynamic compression of the Navier-Stokes equation, e.g. `x2_GalerkinProjection.sh`;
  - '3' for the dynamical system, in which the connection to the Navier-Stokes equations is lost, e.g. `x3_solveDS.sh`;
  - '4' ... higher levels which are not used in this report.

4. A character behind '`x`', '`y`' and '`z`' indicates an auxiliary file, e.g. '`s`' for routine segments and '`v`' for variable segments.

A detailed description of the nomenclature is postponed to Secs. 12 and 13.

Part II

xAMC modelling example

5. Installation of xAMC

This chapter describes the installation of xAMC on a Linux system. The installation is performed in three steps:

(1) installing the freeware (see Sec. 5.1),

(2) unpacking the package (see Sec. 5.2), and

(3) specifying the FORTRAN compiler, editor and viewers (see Sec. 5.3).

5.1 Prerequisites for the Linux system

The `xAMC` toolkit requires

1. a text editor (e.g. `xemacs`, `kwrite` or `gedit`);
2. a FORTRAN compiler (e.g. `gfortran` or `ifort`);
3. a gnuplot version;
4. a graphics viewer for png-files (e.g. `xv`, `firefox` or `gimp`);
5. and a graphics viewer for PostScript files (e.g. `gv` or `evince`).

5.2 Unpacking the `xAMC` package

This report comes with a tar-file `xAMC.tar.gz` containing basic components of `xAMC`. The command

```
> tar xvfz xAMC.tar.gz
```

yields three directories: `xRun`, `xTools`, `yGM` with following contents (see also Fig. 4.1).

`xRun` is the directory for the exection. It contains:

`xAMC`: the master shell script for the reproduction of ROMs.

`CLEAN`: a shell script which removes the changes of `xAMC`, e.g. deletes the `run` directory.

`devKit`: a directory which contains several small auxiliary shell scripts called by `xAMC`.

`docu`: a directory containing a documention of `xAMC`.

`xTools` provides the computational programs. It contains following directories:

`xLib`: subroutines for computational and visualization tasks.

`x0_GM`: programs acting on the raw data.

`x1_GM`: programs creating the Galerkin expansion.

`x2_GM`: programs performing the Galerkin projection.

`x3_GM`: programs analyzing and augmenting the Galerkin model

`x3_DS`: programs for the dynamical system.

`yGM` has a sample Galerkin model (see also Fig. 4.1):

`Noack_2003jfm`: is a directory reproducing the one ROM of ? (see Sec. 3). This directory also contains the snapshots of the periodic vortex street. The inclusion of DNS data deviates from the usual `zData`-link because the amount of data is small and because the file system shall be kept simple.

5.3 Defining the local variables

5.3.1 FORTRAN compiler

The preferred FORTRAN compiler, can be specified in

```
xTools/xLib/x_defineFortranCompiler.sh
```

One example is

```
FortranCompiler="gfortran -O3 -fbounds-check"
export FortranCompiler
```

This script is executed before any compilation. The provided shell script contains many unessential comment lines.

5.3.2 Editors and viewers

The preferred text editor, graphics viewer, PostScript viewer can be determined in

```
xTools/xLib/x_defineLocalVariables.sh
```

For the the editor `gedit`, the image viewer `firefox`, and the PostScript viewer `evince` the shell script may contain following lines:

```
TextEditor=gedit;   export TextEditor
PSViewer=evince;    export PSViewer
PNGViewer=firefox;  export PNGViewer
```

This shell script is executed for running any executable program.

6. Execution of xAMC

This chapter describes a sample execution of `xAMC` for the construction of the ROM provided in Sec. 3. This description includes the main command (Sec. 6.1), an overview of the `run` directory (Sec. 6.2), and an explanation of the master shell script (Sec. 6.3).

6.1 Main command

After the installation of `xAMC` (Sec. 5), the construction is performed with single command:

```
> ./xAMC ../yGM/Noack_2003jfm_cy2u
```

This command creates the `run` directory and executes the master shell script for the ROM — following Sec. 5. On a modern PC, this may take about one minute, mostly for the compilation of the programs and visualization of the flows. Several result pictures for the flow, the modes, and the Galerkin solution pop up with the selected viewers.

6.2 Contents of the run directory

The `run`-directory now contains about 150 files and 4 subdirectories:

- `xTools` with the FORTRAN sources;
- `yTools` with the executable files;
- `z0_Data` with the soft link to the snapshot data in `yGM/Noack_2003jfm/z0_Data`; and
- `z1_Data` with the Galerkin expansion.

A detailed description is postponed to Part IV. For the moment, a listing of few items shall suffice.

`yAMC.sh` is the master shell-script, executing following shell-scripts in the `run` directory:

```
x0_plotGrid.sh
x0_plotFlow.sh
x1_CorrelationMatrix.sh
x1_eigenproblemPOD.sh
x1_POD.sh
x1_GalerkinApproximation.sh
x2_GalerkinProjection.sh
x3_makeDS.sh
x3_solveDS.sh
x3_plotAmplitudes.sh
```

The parameters of the ROM are contained in following files of the run directory:

```
y0_Grid.dat
y1_Domains.dat
y0_Flows.dat
y1_Flows.dat
y1_GM.dat
y1_Modes.dat
y2_GM.dat
y2_Modes.dat
```

6.3 Modeling steps

The construction of the ROM is controlled by the master shell script yAMC.sh. A listing reads:

```
#************************************************************************
#***** @task:       execute AMC pipeline ********************************
#***** @dependency: has to be executed in the xAMC run directory *******
#***** @author:     Bernd R. Noack *************************************
#***** @revised:    2012-01-15 Syracuse (obsolete INIT2u removed) ******
#***** @revised:    2010-08-28 Poitiers ********************************
#***** @created:    2010-04-28 Poitiers ********************************
#************************************************************************

source x_defineLocalVariables.sh
mkdir z1_Data || echo "z1_Data exists!"

#***** Plot grid and first snapshot **************
./x0_plotGrid.sh
./x0_plotFlow.sh < x0_plotFlow.inputFlow0001
$PSViewer Flow.0001_S.ps

#***** Perform POD *******************************
./x1_CorrelationMatrix.sh
./x1_eigenproblemPOD.sh
./x1_POD.sh
./x0_plotFlow.sh < x0_plotFlow.inputMode0001
$PSViewer Mode.0001_S.ps

#***** Add and plot shift mode ******************
./x1_GalerkinApproximation.sh
./x0_plotFlow.sh < x0_plotFlow.inputMode0009
$PSViewer Mode.0009_S.ps
```

```
    #***** Perform Galerkin projection ***************
    ./x2_GalerkinProjection.sh

    #***** Make the dynamical system *****************
    ./x3_makeDS.sh

    #***** Solve dynamical system *******************
    ./x3_solveDS.sh
    ./x3_plotAmplitudes.sh
```

In the following, we explain the actions of this shell script.

1. Initialize the local variables, like the chosen PNG viewer.

   ```
   source x_defineLocalVariables.sh
   mkdir z1_Data || echo "z1_Data exists!"
   ```

   The z1_Data directory is created for the Galerkin expansion modes.

2. Display the grid.

   ```
   ./x0_plotGrid.sh
   ```

   After this script, the PostScript viewer pops up with a plot of the grid.

3. Visualize one snapshot.

   ```
   ./x0_plotFlow.sh < x0_plotFlow.inputFlow0001
   $PSViewer Flow.0001_S.ps
   ```

   After these tasks, a PostScript file with the streamlines of the first snapshot is shown. This script lists the main parameters of the Galerkin model to be constructed, e.g. the snapshots to be used, the modes to be constructed, etc.

4. Compute the POD.

   ```
   ./x1_CorrelationMatrix.sh
   ./x1_eigenproblemPOD.sh
   ./x1_POD.sh
   ./x0_plotFlow.sh < x0_plotFlow.inputMode0001
   ```

   The first three scripts perform the POD. First, the correlation matrix is computed. Then, the eigenproblem is solved. And finally, the POD modes are assembled as linear combinations of the snapshot fluctuations. In the last line, yAMC.sh visualizes the first POD mode like the previous flow fields.

5. Add the non-POD mode(s).

   ```
   ./x1_GalerkinApproximation.sh
   ./x0_plotFlow.sh < x0_plotFlow.inputMode0009
   ```

In these lines, the shift mode is added and visualized.

6. Perform the Galerkin projection.

   ```
   ./x2_GalerkinProjection.sh
   ```

   Here, the script produces the Galerkin system coefficients l_{ij}^{ν} and q_{ijk}^{c} for $i = 1, \ldots 9$ and $j, k = 0, \ldots, 9$.

7. Compute the dynamical system.

   ```
   ./x3_makeDS.sh
   ```

   The dynamical system computes the pure constant, linear and quadratic term.

8. Compute and show the Galerkin solution.

   ```
   ./x3_solveDS.sh
   ./x3_plotSolution.sh
   ```

   At the end of this process, a PDF file with the temporal evolution of the mode amplitudes is displayed.

6.4 Main results

The `xAMC` command produces a Galerkin model for periodic vortex shedding. Apart from the shown figures, following results may be inspected and compared with the corresponding data of Sec. 3.

6.4.1 Galerkin expansion

These are contained in `z1_Data` and may be visualized by executing `x0_plotFlow.sh` and responding with the file name.

The spectrum is contained in `z1_SpectrumPOD.dat`.

```
1    2.5866981398658608
2    2.5271145550389322
3   9.12016916624641544E-002
4   9.04780322226650668E-002
5   2.85384891708289450E-002
6   2.84441544839046899E-002
7   1.67187672134170803E-003
8   1.66878202850828791E-003
```

6.4.2 Galerkin system

A listing of the viscous term is provided z2_ViscousTerm.dat.

```
    1  0  9.76931251858641214E-003
    1  1  -2.5203958980433834
    1  2  5.99194421326098292E-002
    1  3  8.13139790514311178E-004
    1  4  3.16551336001615058E-003
    1  5  4.22156737113055652E-002
    1  6  1.85963396942649128E-002
    1  7  2.05427084425675172E-003
    1  8  1.26754905906375018E-003
    1  9  1.29415616415176626E-003
    2  0 -1.77924158874104288E-005
 ...
    9  9  -3.1196455199655206
```

Each line contains i, j and l_{ij}^{ν}.

Similarly, the convective term is listed in z2_ConvectiveTerm.dat:

```
    1  0  0 -8.70147661623209279E-005
    1  0  1 -2.70492265120131983E-002
    1  0  2   1.0243975091889415
 ...
    1  0  9  1.95770821520876125E-004
    1  1  0  5.74082867526624838E-002
 ...
    1  1  9 -1.80664565670516503E-002
    1  2  0  7.47659177997665181E-002
 ...
    9  9  9  8.48331704298415507E-005
```

Each line contains i, j, k, and q_{ijk}^{c}.

The coefficients q_{ijk}^{+} are contained in y3_DS_Qijk.dat following the listing format of z2_ConvectiveTerm.dat.

6.4.3 Galerkin solution

z3_Times.dat contains the sampling times of the Galerkin solution $m \mapsto t_m$. Each line contains m and t_m.

z3_Amplitudes.dat contains the amplitude values $m \mapsto a_i(t_m)$. Each line has the format m, i and $a_i(t_m)$. For visualization purposes, the amplitude history of the first mode are stored in gnu_ds_ta0001.dat. Similarly, $a_i(t)$, $i = 2, \ldots, 9$ is saved.

7. Program descriptions

This chapter describes the actions of the programs employed in `yAMC.sh` (see Sec. 6.2) in some detail. This includes visualization programs for the grid `x0_plotGrid.sh` (Sec. 7.1), for the flow `x0_plotFlow.sh` (Sec. 7.2), and for the amplitudes `x3_plotAmplitudes.sh` (Sec. 7.10). The Galerkin expansion is constructed with `x1_CorrelationMatrix.sh` (Sec. 7.3), `x1_eigenproblemPOD.sh` (Sec. 7.4), `x1_POD.sh` (Sec. 7.5), and `x1_GalerkinApproximation.sh` (Sec. 7.6). The dynamical system is derived with `x2_GalerkinProjection.sh` (Sec. 7.7), and `x3_makeDS.sh` (Sec. 7.8) and solved with `x3_solveDS.sh` (Sec. 7.9).

7.1 Plotting the grid

The program `x0_plotGrid` reads the grid from `y0_Grid.dat`. In fact, this file only contains the name of the file `z0_Data/Grid.dat` with the nodes and vertices. This link is implied by the mapping symbol '$\mapsto$' in Tab. 7.1. The grid visualization is shown in `Grid.ps` and is, or course, grid-dependent. Tab. 7.1 summarizes the I/O.

Table 7.1: I/O for `x0_plotGrid`

Input:	• `y0_Grid.dat` $\mapsto$ `z0_Data/Grid.dat`
Output:	• `Grid.ps`

7.2 Plotting the flow

The program `x0_plotFlow` is illustrated for the first snapshot. The program asks if a snapshot (option 1) or a mode (option 2) is to be visualized. After choosing option 1, the number m of the snapshot is requested. $m = 1$ is inserted. In `yAMC.sh`, this user input is provided in the keyboard file `x0_plotFlow.inputFlow0001`.

Thereafter, the grid is loaded from `y0_Grid.dat`, like in Tab. 7.1. Similarly, the name of the mth snapshot `z0_Data/Flow.0001` is found in `y1_Flows.dat`.

This flow is visualized with streamlines in `Flow.0001_S.ps`, with the x-component of the velocity in `Flow.0001_u.ps`, and with the y-component of the velocity in `Flow.0001_v.ps`. The visualization is grid-dependent. Tab. 7.2 summarizes the I/O.

7.3 Computing the correlation matrix

The tool `x1_CorrelationMatrix` assumes a domain decomposition in several grids. In `y0_Domains.dat`, it finds a single domain `z0_Data/Grid.dat`, like in the previous sections.

Table 7.2: I/O for x0_plotFlow

User:	• Choose between snapshots and modes • Specify number of snapshot 'm' or mode 'i'
Input:	• y0_Grid.dat $\mapsto$ z0_Data/Grid.dat • y1_Flows.dat $\mapsto$ z0_Data/Flow.0001 for choosing the first snapshot
Output:	• Flow.0001_S.ps .... for streamlines • Flow.0001_u.ps .... for iso-contours of u • Flow.0001_v.ps .... for iso-contours of v

First, the correlation matrices are computed on the subdomains (here only z1_CorrelationMatrix.0001) before they are summed up for the whole domain (here: z1_CorrelationMatrix.dat). Evidently, both matrices must be equal.

Tab. 7.3 summarizes the I/O.

Table 7.3: I/O for x1_CorrelationMatrix

Input:	• y0_Domains.dat $\mapsto$ z0_Data/Grid.dat • y1_Flows.dat $\mapsto$ z0_Data/Flow.0001..58 Both, M and the snapshot files, are inferred from y1_Flows.dat.
Output:	• z1_CorrelationMatrix.dat • z1_CorrelationMatrix.0001 (for subdomain)

7.4 Solving the eigenvalue problem

The program x1_eigenproblemPOD reads the number of modes N from y1_GM.dat and the correlation matrix (2.13) from z1_CorrelationMatrix.dat. The dimension M is inferred from the latter file.

The output is the spectrum λ_i, $i = 1, \ldots, N$ in z1_SpectrumPOD.dat, the snaphot mode amplitudes $a_i(t\_m)$, $i = 1, \ldots, N$, $m = 1, \ldots, M$ in z1_Amplitudes.dat, and the coefficients of the POD modes in terms of snapshots in z1_TransformationMatrixPOD.dat. Note that the information is truncated at mode number $N < M$.

Tab. 7.4 summarizes the I/O.

7.5 Assembling the POD modes

x1_POD assembles the POD modes as linear combinations of the snapshots with the input of x1_eigenproblemPOD. It works on the whole domain as specified by y0_Grid.dat. The grid is

Table 7.4: I/O for x1_eigenproblemPOD

Input:	• y1_GM.dat • z1_CorrelationMatrix.dat
Output:	• z1_SpectrumPOD.dat • z1_Amplitudes.dat • z1_TransformationMatrixPOD.dat

needed for the dimensions of the flow field.

Tab. 7.5 summarizes the I/O.

Table 7.5: I/O for x1_POD

Input:	• y0_Grid.dat $\mapsto$ z0_Data/Grid.dat • y1_Flows.dat $\mapsto$ z0_Data/Flow.0001..58 • y1_Modes.dat ....only for the file names • x1_TransformationMatrixPOD.dat
Output:	• z1_Data/Mode.0000..8

7.6 Adding the shift mode

x1_GalerkinApproximation adds non-POD modes (here: the shift mode). It works on the whole domain. The construction of the shift mode is specified in y1_GM.dat (see Part IV of this report for details).

Tab. 7.6 summarizes the I/O.

Table 7.6: I/O for x1_GalerkinApproximation

Input:	• y1_GM.dat • y0_Grid.dat $\mapsto$ z0_Data/Grid.dat • y1_Modes.dat $\mapsto$ z1_Data/Mode.0000..8
Output:	• z1_Data/Mode.0009

7.7 Performing the Galerkin projection

Like x1_CorrelationMatrix, the tool x2_GalerkinProjection assumes a domain decomposition in several grids.

The output consists of the mass matrix m_{ij} in z2_MassMatrix.dat[1], the viscous term l^{ν}_{ij} in z2_ViscousTerm.dat, and the convective term q^{c}_{ijk} in z2_ConvectiveTerm.dat. For each term, analogues for subdomain number 1 with extension 0001 are saved. In this case, these subdomain and whole domain coefficients are identical.

Tab. 7.7 summarizes the I/O.

Table 7.7: I/O for x2_GalerkinProjection

Input:	• y1_GM.dat .... for N
	• y2_GM.dat .... for Re, etc.
	• y0_Domains.dat $\mapsto$ z0_Data/Grid.dat
	• y1_Modes.dat $\mapsto$ z0_Data/Mode.0000..9
Output:	• z2_MassMatrix.dat .... (m_{ij})
	• z2_MassMatrix.0001 .... contribution on subdomain
	• z2_ViscousTerm.dat .... l^{ν}_{ij}
	• z2_ViscousTerm.0001 .... contribution on subdomain
	• z2_ConvectiveTerm.dat .... q^{c}_{ijk}
	• z2_ConvectiveTerm.0001 .... contribution on subdomain
	○ z2_PressureTerm.dat .... q^{p}_{ijk}, currently not used
	○ z2_PressureTerm.0001 .... contribution on subdomain, not used

7.8 Creating the dynamical system

The program x3_makeDS reads the number of modes N_A, N from y1_GM.dat, the number of volume forces N_V and $\nu = 1/Re$ from y2_GM.dat, and the Galerkin system coefficients l^{ν}_{ij} and q^{c}_{ijk} from z2_ViscousTerm.dat and z2_ConvectiveTerm.dat, respectively. The mass-matrix inversion, pressure-term represenation and volume-force representation is not currently included.

The output is the descriptor system (2.22) with q^{+}_{ijk} in y3_DS_Qijk.dat, z_i in y3_Modes.dat, the initial condition $a_i(t_0)$ in y3_State.dat, and the integration parameters in y3_Times.dat. Initial conditions and time interval are chosen to match the full plant as specified in z1_Times.dat and z1_Amplitudes.dat.

Tab. 7.8 summarizes the I/O.

7.9 Solving the dynamical system

The program x3_solveDS reads the descriptor system of x3_makeDS and writes the solution in z3_Times.dat and z3_Amplitudes.dat.

Tab. 7.9 summarizes the I/O.

[1]This should be approximately the identity matrix.

Table 7.8: I/O for x3_makeDS

Input:	• y1_GM.dat .... for N • y2_GM.dat .... for $\nu = 1/Re$, N_V ∘ z2_MassMatrix.dat .... m_{ij}, currently not used • z2_ViscousTerm.dat .... l^{ν}_{ij} • z2_ConvectiveTerm.dat .... q^{c}_{ijk} ∘ z2_PressureTerm.dat .... q^{p}_{ijk}, currently not used
Output:	• y3_DS.dat • y3_DS_Qijk.dat .... q^{+}_{ijk} • y3_Modes.dat .... z_i • y3_State.dat .... $a_i(t_0)$ • y3_Time.dat .... $t_0, t_1, \Delta t, \ldots$

Table 7.9: I/O for x3_solveDS

Input:	• y3_DS.dat .... N • y3_DS_Qijk.dat .... q^{+}_{ijk} • y3_Modes.dat .... z_i • y3_State.dat .... $a_i(t_0)$ • y3_Time.dat .... $t_0, t_1, \Delta t, \ldots$
Output:	• z3_Times.dat .... sampling times $t \in [t_0, t_1]$ • z3_Amplitudes.dat .... corresponding amplitudes $a_i(t)$

7.10 Visualizing the Galerkin solution

The program x3_plotAmplitudes reads the solution of x3_solveDS and plots the $a_i(t)$ in PNG files and a cumulative PDF. The dimensions of the state space N and of the snapshots M' are inferred from the input files.

Tab. 7.10 summarizes the I/O.

Table 7.10: I/O for x3_plotAmplitudes

Input:	• z3_Times.dat] .... sampling times $t \in [t_0, t_1]$ • z3_Amplitudes.dat .... corresponding amplitudes $a_i(t)$
Output:	• gnu_ta000[1..9].dat .... gnuplot data for $a_i(t)$ • gnu_ta000[1..9].png .... corresponding PNG pictures • gnu_ta.tex .... LaTeX-file comprising all pictures • gnu_ta.pdf .... cumulative PDF with all pictures

Part III

xAMC application guide

8. Changing the Galerkin model

We assume that the `run` directory contains all source files for a ROM. In this chapter, we describe ways of changing the ROM path without re-compiling all FORTRAN sources. One option is changing the parameters in the process-related files as described in Sec. 8.1. Another option is adding or omitting existing tools in the modeling pipeline defined by `yAMC.sh` (Sec. 8.2). A third option is changing the source files (Sec. 8.3). This chapter provides few examples and a brief overview of these options. These examples are appetizers for detailed file content and program descriptions in Part IV.

8.1 Changing parameters

The following list contains examples of changes in the ROM via parameter files

1. The snapshots for POD may be changed in `y1_Flows.dat`
2. The number of POD modes are controlled in `y1_GM.dat`. The file corresponding to Sec. 6 reads:

   ```
   0     [N_A:   number of actuation modes]
   9     [N:     total number of modes]
   8     [N_POD: number of POD/DMD/... modes]

   1  8      ! identity
   9         ! initialize u_D=0
   9  0  1.0 ! linear combination u_D = u_0-u_s
   9 10 -1.0 ! linear combination u
   9  1  8   ! orthonormalize     u_D w.r.t. u_1...8
   ```

   The final lines refer to operations necessary to compute the shift mode as orthonormalized mean-field correction. For details see Sec. 13.

   The parameters for a pure POD model with 8 POD modes reads

   ```
   0     [N_A:   number of actuation modes]
   8     [N:     total number of modes]
   8     [N_POD: number of POD/DMD/... modes]
   ```

   No instructions for non-POD modes are necessary.

   The parameters for a mean-field model with 2 POD modes and one shift mode read, for instance:

   ```
   0     [N_A:   number of actuation modes]
   3     [N:     total number of modes]
   2     [N_POD: number of POD/DMD/... modes]
   ```

```
1  2      ! identity
3         ! initialize u_D=0
3  0  1.0 ! linear combination u_D = u_0-u_s
3  4 -1.0 ! linear combination u
3  1  2   ! orthonormalize     u_D w.r.t. u_1...8
```

Again, the final lines refer to operations necessary to compute the shift mode as orthonormalized mean-field correction.

3. The Reynolds number of the Galerkin system is specified in the first line of `y2_GM.dat`:

   ```
   100     [Re:  Reynolds number]
     0     [N_V: number of volume forces]
   ```

4. The initial condition of the dynamical system is specified in `y3_State.dat`:

   ```
   1      0.01000      [a_1 = 0.01]
   ```

   The listing contains only non-vanishing mode amplitudes.

5. The time integration of the dynamical system is controlled in `y3_Time.dat`:

   ```
     0.00000     [t_0 ]
   250.00000     [t_1 ]
     0.00100     [dt, 1000 steps per unit]
     0.10000     [dt_Show, 10 steps per unit]
     0.10000     [dt_Save, 10 steps per unit]
   ```

   Evidently, the first and final instant are listed in the first 2 lines.

8.2 Changing the modeling pipeline

Examples for performing new tasks in ROM are listed below.

1. The command

   ```
   > ./x3_FixedPoint.sh
   ```

   determines the fixed point of the dynamical system. `x3_FixedPoint.f`. Here, the fixed point is specified in `z3_FixedPoint.dat`, like in `y3_State.dat`. The input and output are specified in the header of the corresponding FORTRAN file.

2. The command

   ```
   > ./x3_StabilityAnalysis.sh
   ```

   effects the stability analysis of the fixed point of the dynamical system. Again, the input and output are specified in the header of the corresponding FORTRAN file.

3. The DMD may be effected with

```
> ./x3_eigenproblemDMD.sh
> ./x3_DMD.sh
> ./x3_plotSpectrumDMD.sh
> ./x3_plotModesDMD.sh
```

4. Full versions of `xAMC` contain also tools for a FTT prediction [??]. See the directory `xTools/x4_FTT`.

5. Future versions of `xAMC` allow to bypass the Galerkin projection by a model identification [?].

8.3 Changing the source files

`xAMC` is designed as simple program package for Galerkin methods described in the literature. Hence, many new modeling ideas and options require to change the source files. This is best done in the `run` directory, or by developing and testing new plugins. Few examples are listed below:

1. Development of a new subgrid turbulence model accounting for neglected modes.

2. Adding identified forcing terms to the dynamical system.

3. Adding an identified constant and linear term while keeping the quadratic term of the Galerkin projection [?].

4. Developing the effect of new actuation modes.

5. Testing new sets of projection modes.

All these tasks are minor variations of the standard operating procedure. After convincing tests they may later be included in the `xAMC` package.

9. Setting up a new Galerkin model

In this chapter, the file management related to the Galerkin model is discussed. Some changes require the re-compilation of the sources. Options for this compilation are discussed

9.1 Compilation

In this section, the compilation of the `xAMC` toolbox is discussed.

9.1.1 Compilation of a single program after an `xAMC` execution

The script `xAMC` copies all required source files in the `run`-directory. The compilation is done in this directory. After the compilation most source files are moved in the `run/xTools`-subdirectory. Let `xL_Name.f` be the name of a single program to be re-compiled. Here, L is a place-holder for a number from 0 to 4, and `Name` represents the core program name, e.g. `makeDS`. Re-compilation this program can be performed in the `run/xTools`-subdirectory with

```
> ./xL\_Name\_0.sh
```

The added token '`_0`' marks a compilation shell script. The executable has to be manually moved in the proper directory with

```
> mv YL\_Name ../yTools
```

9.1.2 Compilation of all programs after `xAMC` execution

If all programs are to be re-compiled, this is performed with following commands in this subdirectory:

```
> cp ../z0.dat .
> ./aToolsCompile.sh
> mv Y* ../yTools
```

The first command makes sure that `aToolsCompile` finds its only input file `z0.dat`. Based on the grid type specified in `z0.dat`, `aToolsCompile` decides which shell script of the format `xL_Name_0.sh` needs to be executed and which not. For instance, a Galerkin model for data on a 2D unstructured grid, does not require `x0_plotGrid3c` for visualization of a 3D Cartesian grid. The last commands moves the binary executables in the proper directory.

9.1.3 Compilation of all programs in a user-created `run`-directory

Now, we assume that no `run`-directory has been created. Perform following commands in the `xRun`-directory

```
> mkdir run
> cd run
> cp ../../xTools/xLib/*.* .
> cp ../../xTools/x0_GM/*.* .
> cp ../../xTools/x1_GM/*.* .
> cp ../../xTools/x2_GM/*.* .
> cp ../../xTools/x3_GM/*.* .
> cp ../../xTools/x3_DS/*.* .
> ./aTools.sh
```

The first lines have obvious implications. `aTools.sh` creates all FORTRAN and process-related files for a standard POD Galerkin model by a user dialogue. This implies compilation and creation of the subdirectories `xTools` and `yTools`. Thereafter, changes of the process files or programs may be effected. Sec. 13 provides more information for such operations.

9.2 Galerkin expansion

The Galerkin expansion is initialized in following steps:

1. Adjust `xv.include`, i.e. determine the maximum number (physical dimension) of actuation modes (`NAMP`) and the maximum number of expansion modes (`NEMP`).
2. Update the parameter file `y1_GM.dat`, as explained in Sec. 13.2.
3. Specify the names of the modes in `y1_Modes.dat`. Note that the list must contain all expansion modes, i.e. the mapping from $i = -N_A, \ldots, N$ to file names. But it may also contain auxiliary files. In the example of Sec. 6, the steady solution has been added as $N+1$th mode for the construction of the shift mode.

9.3 Galerkin projection

The Galerkin projection is initialized in following steps:

1. Adjust `xv.include`, i.e. determine the maximum number of volume forces N_V (`NVFP`).
2. Update the parameter file `y2_GM.dat`, as explained in Sec. 13.2.
3. Specify the file names of the test functions in `y2_Modes.dat` for Petrov-Galerkin method. Up to now, only the traditional Galerkin method is implemented, i.e. `y2_Modes.dat` is not used.
4. Specify the file names for the volume forces in `y2_Forces.dat` (only if $N_V > 0$).

9.4 Dynamical system

Determine the maximum dimension N_a (`NAP`) of the descriptor system in `xv_DS.include`. Note that

$$N_a = N + 2N_A + N_V$$

defines a lower bound for N_a. Some auxiliary models, e.g. for subgrid turbulence, add additional states to the descriptor system.

`x3_makeDS.sh` initializes all necessary parameter files:

1. `y3_DS.dat` (as umbrella),
2. `y3_DS_Qijk.dat` with q_{ijk},
3. `y3_Modes.dat` with z_i,
4. `y3_State.dat` with $a_i(t_0), i = 1, \ldots, N_a$, and
5. `y3_Time.date` with $t_0, t_1, \Delta t$ etc.

See Sec. 13.2 for a detailed description.

10. Setting up new flow data

This chapter outlines the required `xAMC` format of flow data for modelling. The first section (Sec. 10.1) outlines how `xAMC` copes with flow data on various grids. Sec. 10.2 lists the grid-dependent segments. Sec. 10.3 specifies the data files which need to be created. Sec. 10.4 describes implemented data formats for various grids. Sec. 10.5 outlines the format of the I/O routines for reading the grid and flow data. Sec. 10.6 explains the `xAMC` setup for domain decompositions.

10.1 How `xAMC` processes flow data on various grids

In principle, the flow data to be modelled may be of many kinds:

1. finite-dimensional phase space, e.g. for reading of pressure culites;
2. one-dimensional, e.g. for solutions of the 1-D Ginzburg-Landau equation;
3. two-dimensional, e.g. for 2-D PIV data; and
4. three-dimensional, e.g. for a 3-D Navier-Stokes simulation of turbulence.

In case of fields, the flow may be stored on different grids. The three-dimensional data, for instance, may be stored on an unstructured grid, a general structured grid, a Cartesian grid, or even on a Cartesian equidistant grid.

Evidently, some parts of the `xAMC` toolkit have to be grid-dependent, e.g. the computation of an inner product depends on the type of grid. The I/O of data also depends on the grid with its dimensions. However. evidently, it is also desirable to minimize the number of the grid-dependent variables and elements. `xAMC` has been designed to minimize grid dependence. Internally, the flow is represented in a pseudo-3D manner, i.e. N_f dependent flow variables are written in an $N_x \times N_y \times N_z$ dimensional array. For 2-D data, $N_z = 1$. For an unstructured grid, $N_y = N_z = 1$. Thus, many elementary operations, e.g. linear combinations of flow fields, are computed in a grid-independent manner.

Only three tasks are explicitely grid-dependent:

1. I/O of the grid;
2. I/O of a single flow field; and
3. few elementary computational routines associated with the inner product and the Navier-Stokes terms (viscous, convective and pressure).

Often, the computations are performed on a set of subdomains. Hence, segments for I/O of subdomain must be included as well.

10.2 Grid-dependent segments

Table 10.1 shows the minimal list of grid-dependent segments. The variable segments define dimensions of the flow data. `xv3X_Index.include`, for instance, contains the physical dimensions for N_x, N_y, N_z and N_f. The token `ioGrid` and `ioFlow` refers to I/O routines for the grid and the flow. The token `Sub` implies operation on the subdomain. For instance, all Galerkin modelling operations are performed on the subdomain with the routines provide in `xs3X_gxCORESub.f`. The following chapters detail the contents.

Table 10.1: List of grid-dependent `xAMC` segments. Note that `xv3X_Flow.include` and `xv3X_FlowSub.include` employ the pseudo 3-D format and a grid-independent.

segment	whole domain	subdomain
for variables	`xv3X.include`	Ditto
	`xv3X_Grid.include`	`xv3X_GridSub.include`
for subroutines	`xs3X_ioGrid.f`	`xs3X_ioGridSub.f`
	`xs3X_ioFlow.f`	`xs3X_ioFlowSub.f`
		`xs3X_gxCORESub.f`

It should be noted that `xAMC` is based on extended Fortran 77 and does not make use of dynamic arrays. This implies that all files of Tab. 10.1 must be adapted for the data to be modelled. The `xLib` directory has templates for various grid formats. The tokens `2u`, `2c`, and `3c` indicate segments for 2-D unstructured grids, 2-D Cartesian grids and 3-D Cartesian grids, respectively. The shell scripts `aTools[2u,2c,3c].sh` copy their contents in the corresponding `3X`-file.

10.3 Flow data

At minimum, four type of files need to be provided:

`z0_Data/Grid.dat`: This file contains the grid. The name is a suggestion.

`z0_Data/Flow.XXXX`: Each flow snapshot is stored in a separate file. `XXXX` refers to the number of the snapshot. The nomenclature is a suggestion.

`z0_Times.dat`: This ASCII file contains the snapshot number and the corresponding time.

`y0_Flows.dat`: This file contains the complete list of snapshots.

Grid-dependent data is stored in a subdirectory `z0_Data`. There exist desirable file names and file contents for that directory, but no mandatory one. More information is provided in Chapter 13.

10.4 Recommendations for flow data formats on different grids

In the following subsections, recommendations for flow data formats are provided for different grids. This recommendations are obeyed in the templates of the grid-dependent routines in `xTools/xLib`. The format is ultimately dictated by the I/O routines.

10.4.1 2-D unstructured grids

A sample listing of `z0_Data/Grid.dat` for the cylinder wake model (Sec. 3.) reads:

```
15838 1 1 2 ! NX, NY, NZ, NVarGrid
  15.  5.
  14.53125  5.
  15.  4.375
  14.0625  5.
  ....
 31352 3 ! NElements, NPoints
 2 4002 4003
 4003 4002 4001
 4001 4002 1
 ....
```

The first line contains the dimensions of the pseudo 3-D grid `NX`, `NY`, `NZ`, and the dimension of the space `NVarGrid`=2. This line is followed by `NX` nodes x_n, y_n, $n = 1, \ldots,$`NX`, one per line.

The fact that an effectively one dimensional array (`NY`=`NZ`=1) shall describe a two-dimensional space implies the need for connectivity information. This mesh is provided is provided thereafter via `NElements`=31352 triangles characterized by `NPoints`=3 vertices. Each line contains the node indices of the vertex edges. This file describes a standard 2-D unstructured grid (see Fig. 3.1). Note that more points per triangle may be used for higher-order differentiations and interpolations.

A typical listing of a snapshot (here: `z0_Data/Flow.0001`) reads:

```
15838 1 1 2
  1.027  0.02504
  1.03  0.02314
  1.042  0.0178
```

The first line contains `NX`, `NY`, `NZ` and `NVarFlow`=2. The last number determines the number of dependent flow variables. In the following, the 2-D velocity components u_n, v_n are listed for all `NX` grid nodes. If, for instance, the pressure is added, `NVarFlow`=3. Hence, `NVarFlow` may be larger than `NVarGrid`.

10.4.2 2-D Cartesian grids

A sample listing of a typical grid in z0_Data/Grid.at is similar to the analogue in Sec. 10.4.1:

```
200 48 1 2 ! NX NY NZ NVarGrid
0.0
0.2
...
```

After the line with grid dimensions, the x values followed by the y values are listed.

A sample listing of the first snapshot in z0_Data/Flow.0001 is similar to the analogue in Sec. 10.4.1::

```
200 48 1 2 ! NX NY NZ NVarFlow
  0.49999954375000005        0.0000000000000000
  0.49971606458333340        0.0000000000000000
  0.49940091041666673        0.0000000000000000
....
```

The velocity component is written out by

```
     do iy=1,NZ
     do iy=1,NY
     do ix=1,NX
       (fFlowG(ix,iy,iz,iDummy),iDummy=1,NVarFlow)
     end do
     end do
     end do
```

10.4.3 3-D Cartesian grids

The recommendation of the 3-D Cartesian grid is in complete analogy to the 2-D Cartesian grids. A large data volume may make a binary storage of flow fields desirable. In future versions of the xAMC-toolkit, an implementation of CGNS is planned.

10.4.4 Other grids

It does not take much imagination to write down the formats for 1D grids, 2D structured grids or 3D unstructured grids. We shall not pause to explicit these formats.

10.5 I/O routines

10.5.1 Grid

The xAMC grid interface for new data is implemented in two steps:

1. Determine the right physical dimensions in `xv3X.include`.

2. Write one

   ```
       subroutine readGrid
   ```

   in `xs3X_ioGrid.f` which brings `z0_Data/Grid.dat` into the format specified by `xv3X_Grid.include`. There exist templates for different grids (see Sec. 10.2).

Note that the grid is stored in a common block. Only variables which change during the program run may be included in argument lists.

10.5.2 Flow data

The `xAMC` interface for new flow data requires two subroutines for reading and writing the flow.

```
subroutine readFlow(fn,fnLength,fFlowG)
subroutine writeFlow(fn,fnLength,fFlowG)
```

Here `fFlowG` represents the flow as specified in `xv3X_Flow.f`, `fn` the `character*120` file name, and `fnLength` the length of this file name.

10.5.3 Programs operating on the whole domain

Now, `x0_plotGrid3X.sh` (or grid-dependent variants thereof) will visualize the grid.

`x0_plotFlow3X.sh` (or grid-dependent variants thereof) will visualize the flow field.

Note that the reduced-order modelling is performed on the subdomain. This subdomain may be identical to the whole domain, if all snapshots and modes can be stored in the RAM of the PC.

10.6 Setting up the domain decomposition

If the snapshots and modes cannot be stored in the RAM of the PC, a domain decomposition is necessary. The domain decomposition is implemented in 7 steps:

1. Initialize the dimensions of the subdomain in `xv3X.include`. All snapshots on that subdomain must fit in the RAM.

2. Initialize `y1_Domains.dat`. The contents of this file is strongly data dependent. One suggestion is the list

   ```
       1 z0_Data/Grid.0001
       2 z0_Data/Grid.0002
   ```

where the specified files contain the complete information. However, the number of lines is interpreted as the number of domains and used in corresponding loops. As one rule of thumb,

$$\text{size of all snapshots/number of domains} \approx 50\% \text{ of RAM}$$

3. Write `y1_Flows.dat` in analogy to `y0_Flows.dat`. The analysis may be performed on a subset of the original data, e.g. only the post-transient phase or every 5-th snapshot.

4. Adjust `z1_Times.dat` correspondingly.

5. Write one

   ```
   subroutine readGridSub(iDomain)
   ```

   in `xv3X_ioGridSub.f` for reading the grid of the `iDomain`-th subdomain.

6. Write one

   ```
   subroutine readFlowSub(iDomain,fn,fnLength,fFlowSubG)
   ```

   in `xv3X_ioFlowSub.f` for reading the flow of the `iDomain`-th subdomain — in complete analogy to the corresponding routine of the whole domain.

7. The computational heart of `xAMC` is contained in `xs3X_gxCORESub.f` with subroutines for the inner product and Navier-Stokes related operators, all operating on the subdomain. There exist grid-related templates, e.g. `xs2u_gxCORESub.f` for the 2-D unstructured grid. For some operations — not for standard POD Galerkin models — a routine for the inner product on the whole domain is included. Re-visit this segment in case of necessary adaptions.

No output routines are necessary for the grid and the flow on the subdomains. The grid does not change during program execution and any flow (including the expansion modes) is stored on the original domain.

Part IV

xAMC programming guide

11. Desiderata

The current version of the AMC toolkit is the product of many considerations for its intended use. The side constraints are elaborated for the employed software (Sec. 11.1), for the user friendliness (Sec. 11.2), and for the programmer friendliness (Sec. 11.3).

11.1 Software specifications

We assume that a single ROM-development is performed by several groups on autonomous Linux PCs. Following assumptions describe the environment of the authors.

Distributed execution: Different parts of the ROM pipeline are performed by different groups on their own PCs. Quick exchange of data and tools is key to an efficient execution.

Multi platform: Each group has their own Linux platform, e.g. Ubuntu (in France), SuseLinux (in Germany) and RedHat (in USA).

Multi source: Each group uses their own programming tools, e.g. Fortran77, Fortran90, C++, MatLab, an so own.

Multi applications: The same data is subjected to several different modeling strategies.

Multi data: The same tools are applied to different data, e.g. 0-D sensor data, 1-D Ginzburg-Landau Equation as well as 2-D and 3-D flows. The data may arise from analytical formulae, experimental sources or heavy-duty CFD simulations requiring O(1) TeraByte.

Multi idea: The platform shall open the possibility to quickly test new ideas outside the standard operation procedure.

Law of equivalent stupidity:[1] A smart solution becomes stupid if it cannot be easily communicated to other groups or if it is forgotten by the inventor a few months later.

The described environment practically excludes

- a makefile (each group has their own sources and compilation options);
- libraries for tasks (each group has their own programming language);
- a web-based user interface (conflict with versatility);
- a version management (CVS,...), since each group maintains their own versions.

The described environment suggests a Linux-inspired programming solution: a smart umbrella for the execution of the whole pipeline, and many simple supporting shell scripts. Moreover, a meticulous nomenclature and order of the file system is required so that the whole ROM development from original data to final model can be read by all groups.

[1]To the best of the first author's knowledge, the inventor of this law is Andrzej Banaszuk (UTRC).

11.2 Good practices for user friendliness

Simple reproducibility: A user should be able to perform a complete xAMC analysis with a single command.

Documentation of programs: The programs should be documented — both in the source files and in a separate documentation (like this report).

Documentation of data: The data should be documented. Such a description may include

1. ID: A short identifier, re-used, e.g. in the directory name `Comte_2004_ml`;
2. Source: e.g. Comte, 2004, LES (including order of scheme);
3. References: publications most related to the configuration;
4. Configuration: complete description of the configuration, e.g. evolution equations, order parameters (Re, Ma, ...), domain size;
5. Grid: discretization of the domain;
6. Snapshots: sampling times and stored quantities;
7. Data format: ASCII, big endian, little endian, CGNS, ... This includes a simple reading routine for a single snapshot.
8. Location: name of raid server, directory, ...

Comprehensibility of the tasks: The whole pipeline should be specified in a single file. The task of a program should be clear by the file names.

Comprehensibility of the results: The produced data and figures should be easily interpretable.

11.3 Good practices for programmer friendliness

The `xAMC` toolbox is a compromise between user and programmer friendliness. A large user friendliness may imply many measures which are difficult to maintain by a programmer, e.g. a graphical user interface and meticulous checking routines for the input. Programmer friendliness should imply the following features.

Documented standard operating procedure: This implies a standard path from the data to the model with a complete description of the analytics and possibly few alternatives.

Data readability: Small amounts of data should be saved in easily readable ASCII format. Large amounts of data should be stored a readable binary format. Other binary formats frequently lead to trouble. For instance, there exist some binary formats which are accessible by Intel Fortran, but not by gfortran.

Documentation: The purpose, input, output and algorithms of the file should be comprehensible from the source code alone. Ideally, the comment lines alone should tell the informed researcher the algorithm.

Linux / Unified Modeling Language (UML) philosophy: The philosophy of Linux and UML should be appreciated, e.g. an intelligent umbrella but many small stupid shell scripts and programs.

Separate data, processes, tools and meta-tools: A general programming strategy is to separate

- data (including raw and processed data),
- tools used for processing the data,
- processes for the execution of several tools on the data,
- and meta-tools for selecting these processes and additional tasks

as good as possible, e.g. by putting them in different main directories.

Let Mr. ASCII sort the files: We adopt the flat-file hierarchy and flat-name philosophy. Thus, files may be easily found in few directories. A strongly hierarchical file structure is good for the main program developer but generally bad for comprehensibility by a novice. On the downside, the lack of a sophisticated directory implies that many files have to be sorted by Mr. ASCII and must obey a clean nomenclature.

To some extend user and programmer friendliness share the same documentation standards. The compromise made is the lack of a graphical user interface and other strongly PC dependent features. The realization of the requirements mentioned in this chapter will be elaborated in Part III.

12. File system

This chapter describes the file system of `xAMC`, briefly sketched in Sec. 4. In particular, the files will be related to their role in the reduced-order modeling pipeline and to their role for the executables. First, the reduced-order modeling process is partitioned in 5 level (Sec. 12.1) This consideration leads to a classification of files for tool-related (Sec. 12.2), for data-related (Sec. 12.3) and for processes-related files (Sec. 12.4).

This chapter focuses on the ordering methodology behind the file system, a kind of 'structure identification' of the `xAMC` software. The next chapter discusses the content of important files.

12.1 The reduced-order modeling pipeline

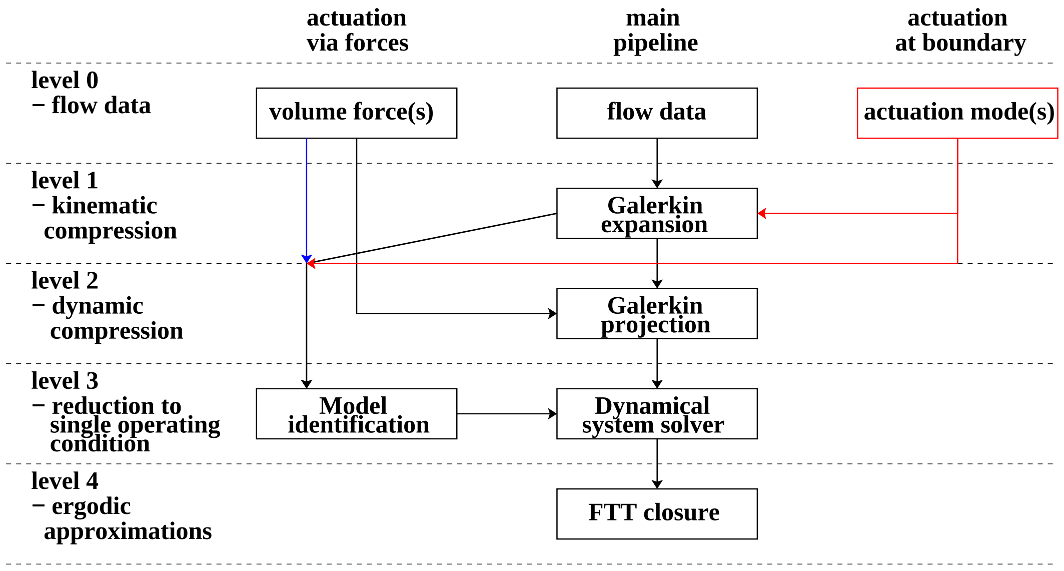


Figure 12.1: Reduced-order modeling pipeline

The reduced-order modeling process (see Sec. 2 and Fig. 12.1) consists of following levels:

Level 0 – Flow data: The starting point of the ROM is numerical or experimental flow data. Generally, this data consists of snapshots. In case of actuation, the employed volume forces or actuation modes need to be added. The snapshot data may be replaced / augmented by physical modes.

Level 1 – Galerkin expansion (kinematic compression): On this level, the snapshots are compressed to a Galerkin expansion. The flow is analyzed in a new phase space (modal piano).

Level 2 – Galerkin system (dynamic compression): In this level, the modal phase space is equipped with its physical propagator, the projection of the Navier-Stokes equation. It

Table 12.1: Some information flows for the construction of a POD Galerkin model without actuation.

Level	Parameter	Tasks / Tools	Data
0	...	• DNS/experiment	$\{\boldsymbol{u}^m\}$
1	M	• correlation matrix	$\boldsymbol{C} = C^{mn}$
	N	• POD eigenproblem	λ_i, α_i^m $i = 1, \ldots, N$ $m = 1, \ldots, M$
	N	• POD modes	$\boldsymbol{u}_i$, $i = 1, \ldots, N$
2	Re	• Galerkin projection	l_{ij}^{ν}, q_{ijk}^{c} $i = 1, \ldots, N$, $j, k = 0, \ldots, N$
3		• make dynamical system	q_{ijk}^{+} $i = 1, \ldots, N$, $j, k = 0, \ldots, N$
	$a_i(0)$, t_0, t_1 $i = 1, \ldots, N$	• solve dynamical system	$a_i(t)$, $t \in [t_0, t_1]$ $i = 1, \ldots, N$

should be emphasized that the resulting Galerkin system keeps track of the physical origin of all its terms. This implies that the Reynolds number and other aspects of the flow can be varied in the Galerkin system.

Level 3 – Dynamic system: In this level, the propagator is simplified in constant, linear and quadratic terms for a single Reynolds number. The constant term, for instance, aggregates the convective, viscous or pressure term. The relation to Navier-Stokes terms is lost.

Level 4 – Ergodic theory: In this level, statistical closures, like FTT [?] are employed. These closures are not subject of this report.

Figure 12.1 illustrates this pipeline. Boundary actuation effects the Galerkin expansion (level 1) and volume forces are required in the Galerkin projection (level 2). Model identification (see, e.g. [??]), bypasses the Galerkin projection and leads directly to the dynamical systems level. Identified models do not contain information of the Galerkin system level, e.g. the effect of the Reynolds number, the effect of the pressure term, etc.

On each level, predetermined parameters of the full plant / ROM are required (structure identification) and data of that model are produced (parameter identification). ROM data of level L is needed for the construction of level $L + 1$. These information flows shall be exemplified for a POD Galerkin model without actuation (see Tab. 12.1). The construction of this model requires 7 tasks from the production of the full-plant data to the dynamical system solution. For the POD, M snapshots need to be selected and the order N of the POD has to be chosen. The Reynolds number Re enters as a parameter for the Galerkin system.

From this consideration, it is natural to separate all information in three types of files:

Table 12.2: Main directories of xAMC. Directories are framed. Some file names are placeholders.

$\boxed{\texttt{xTools}}$ contains all tools in subdirectories.	$\boxed{\texttt{yGM}}$ contains all ROM parameters in subdirectories.	$\boxed{\texttt{zData}}$ contains all full-plant data in subdirectories.
Example files for a subdirectory $\boxed{\texttt{x1\_Tool}}$: • x1_Tool_0.sh • x1_Tool.sh • x1_Tool.f • x1_Tool.gnuplot	Example files for a subdirectory $\boxed{\texttt{Noack\_2003jfm}}$: • yAMC.sh • y1_GM.dat,... • xv[3X].include, • plug-ins	Example files for a subdirectory $\boxed{\texttt{CFD data}}$: • z0_Data with snapshots • z0_Times.dat • z0_Amplitudes.dat

Tool-related (x-)files: These files are related to the computational algorithm, can be applied for many ROM and for many data sources. In xAMC, the names of all tool-related files start with the letter 'x'.

Process-related (y-)files: These items define the structure of the ROM, independent of the data source and the details of the computational algorithms. Generally, the parameter identification is done by the user, e.g. the choice of M snapshots or the selection of the order N. Hence, process-related files can be considered to define a *structure identification* of the ROM. The names of all process-related files start with the letter 'y'. Executable files start with Y.

Data-related (z-)files: These files contain information about the full plant (e.g. snapshots) or about the resulting ROM. The names of all data-related files start with the letter 'z'. Generally, these files contain computer-generated numbers, e.g. the correlation matrix. Data-related files can be considered to result from a *parameter identification* of the ROM.

x-, y-, z-files are cleanly separated in three main directories (see Fig. 4.1 and Tab. 12.2). They are merged by the xAMC script in the run-directory.

12.2 Tool-related files (x-files)

Tools are aggregated in subdirectories, e.g. xL_GM, of xTools. In xL_$*$, $L = 0, \ldots, 4$ represents the modeling level (the lowest level in case of several tools) and 'Tool' is a placeholder for a descriptive name.

12.2.1 Main x-files

The subdirectory xL_Tool should contain at minimum four files:

<u>xL_Tool_0.sh</u>, a shell script which compiles xL_Tool.f.

xL_Tool.sh, a shell script which executes the resulting binary YL_Tool, and, possibly, visualizes some results via a gnuplot script xL_Tool.gnuplot. One example is the shell script x1_eigenproblemPOD.sh $\in$ x1_GM for the spectral analysis of the correlation matrix, the corresponding FORTRAN program x1_eigenproblemPOD.f, and x1_eigenproblemPOD.gnuplot for the visualization the POD eigenvalue spectrum.

xL_Tool.f, a FORTRAN program. It is compiled with xL_Tool_0.sh using the compiler and compiler options specified during the xAMC installation (Sec. 5). We deliberately do not use a makefile, because this complicates the multi-user, multi-source side constraints (Sec. 11). Every user has its own favorite FORTRAN, C++ or or other comppiler, and the run may mix these different sources.

xL_Tool.gnuplot, a script for gnuplot. This is executed with xL_Tool.sh via the command

```
> gnuplot ./xTool/xL_Tool.gnuplot
```

Note that the gnuplot script is expected in the subdirectory xTool. The data files for gnuplot visualizations are generally ROM data files zL_⋆.dat or start with gnu_⋆.dat if the data is of more temporary nature. The files gnu_⋆.dat can be deleted, since they can always be re-generated with the appropriate tools.

12.2.2 Auxiliary x-files

The subdirectory xL_Tool generally contains a number of auxiliary x-files used in xL_Tool.f. The auxiliary files are not linked to a modeling level, as a variety of variables and routines, e.g. for input/output, are used in several levels. Following kinds of files are employed:

xv⋆.include are FORTRAN files with variable declarations, which make sure that lists of parameters and index ranges are the same in all modules. xv_⋆.include host grid-independent variables. xv3X_⋆.include contain flow fields.

This distinction is important, since the grid-dependent files might(!)[1] have to be replaced as one changes the type grid, e.g. from 2-D unstructured to 3-D equidistant Cartesian type.

xs⋆.f contain FORTRAN subroutines. The prefix xs_ marks grid independent routines, while xs3X_ the grid-dependent complement.

ns_⋆.f contains FORTRAN subroutines for matrix operations. Most routines are adopted from ? with minor variations.

gs2d_PostScript.f contains FORTRAN subroutines for the generation of PostScript files, like the streamline flow plots.

gv2d_⋆.include represent variable segments used in gs2d_PostScript.f.

[1]This is currently the case. A later version of xAMC attempts to minimize these replacements.

Auxiliary files which are used by several tools are placed in the `xLib` subdirectory.

The file structure mimics features of object-oriented programming. The variable segments `xv⋆.include` represent objects, and the subroutine segments `xs⋆.f` operators. However, we avoid object-oriented programming, since it requires expert programmers and the desire to invest deeply in understanding the `xAMC` package. In short, object-oriented programming would exclude too many amateur programmers with valuable contributions to this effort.

12.3 Data-related files (z-files)

Data-related files contain the flow data of the full-state plant (experiment or CFD) on level 0 and parameters or coefficients of the ROM on higher levels.

12.3.1 Files for the flow field phase space

The discretized flow fields may have different characteristics. These are presently coded in two letters in the `x`-files, e.g. `2u` for 2-D flow on an unstructured grid.

A more general list of options reads:

Dimension: The flow fields may be 1-D, 2-D or 3-D. The dimension refers to the number of space coordinates, not to the number of independent variables.

Grid type: The grid may be Cartesian equidistant '`e`', Cartesian '`c`', structured '`d`', or unstructured '`u`'. The first three types represent increasing generalizations. For computational reasons, it is important to discriminate between equidistant, more general Cartesian and structured grids, because the computational algorithms for the Cartesian equidistant grid have a much lower workload than for corresponding unstructured grids.

Flow field nodes: The flow fields may be stored in collocated, cell-centered, staggered or many other formats. The current version of `xAMC` is restricted to collocated formats, since this is the standard for experimental data and often used in CFD as well.

Single versus multi-domain: The domain may treated in a single, non-overlapping multi-domain or overlapping multi-domain. Currently, only the first two options are implemented.

In the long-term, we target one `CGNS` format for all grid types. Currently, however, all 2-D flows are stored in ASCII format and all 3-D flows in binary format. The ASCII format has the advantage of immediate readability on all PCs at the price of 3 times more storage space after unpacking. These price appears justifiable, since 2-D data are generally not overly storage intensive. The 3-D data may easily require 100s of MB, i.e. the inflation by a factor 3 is generally no option. Reading binary files of new sources is generally a pain and may require many days of testing, installation of a new FORTRAN compiler and the like. In short, transfers of binary files clearly violates the maximum 10 minutes interface time required in Sec. 11.

12.3.2 Files for Galerkin phase space

All ROM-related data files are saved in ASCII format and as numbered lists. For example, the matrix

$$\begin{pmatrix} 1.3 & 0 & 2.7 \\ 0 & -3.2 & 0 \end{pmatrix}$$

is saved as

```
1 1  1.3
1 3  2.7
2 2 -3.2
```

Missing elements are assumed to be zero. Saving elements in lists

```
1.3 0 0 -3.2 2.7 0
```

has severe disadvantages:

1. The order of the elements is not clear apriori. FORTRAN programmers are used to the first index running first, while C programmers are used to the opposite order. A convention may or may not be obeyed by the programmer. At minimum, such a convention is source of potential errors.
2. The zeros have to be saved. This can be particularly space consuming for sparse matrices or very sparse quadratic terms q_{ijk} of spectral Galerkin models.
3. The file cannot easily be interpreted with an ASCII file editor. It is much more pleasing to the eyes (and the linked brain).

These disadvantages appear to justify the space for the added index numbers.

The ROM-related parameters may be vectors, matrixes or 3-tensors. They are stored in indexed lists as discussed above.

1. 1-D fields $(i,\ f^1,\ \ldots,\ f^I)$
2. 2-D fields $(i, j,\ f^1,\ \ldots,\ f^I)$
3. 3-D fields $(i, j, k,\ f^1,\ \ldots,\ f^I)$

The number of elements per index I is generally 1.

12.4 Process-related files (y-files)

Each `yAMC` subdirectory contains 2 kinds of process related files:

<u>`yAMC.sh`</u> , the master shell script. For a pure POD Galerkin model (see Tab. 12.2), this script reads

```
./x1_CorrelationMatrix.sh
./x1_eigenproblemPOD.sh
./x1_POD.sh
./x2_GalerkinProjection.sh
./x3_makeDS.sh
./x3_solveDS.sh
```

with self-explanative tasks of the listed shell scripts. This shell-script is considered to belong to category `y` as opposed `x`, because the chosen tasks are tied to the target data.

<u>`y[0..4]_⋆.dat`</u> , the parameter files (left side of Tab. 12.2). The number $L = 0, \ldots, 4$ refers to the modeling level, at which these parameters are needed. For example, `y1_GM.dat` contains the parameter N.

<u>`YL_Tool`</u> are binary executables which are created from the corresponding `xL_Tool.f` program.

13. File contents

In this chapter, we describe important files for running xAMC and for interpreting or changing the ROM results. This includes subsets of the x-, y- and z-files, namely shell scripts (Sec. 13.1), parameter files (Sec. 13.2), and data files (Sec. 13.3).

13.1 Shell scripts

Almost all compilation shell scripts xL_Tool_0.sh share a very similar procedure:

1. First, the local variables, e.g. the FORTRAN compiler, are imported from x_defineFortanCompiler.sh.
2. The new executable is created from the source files. The shell script terminates in case the compilation is not successful.

An edited listing of x1_eigenproblemPOD_0.sh shall serve as example:

```
#*******************************************************************
#***** @author:  Bernd Noack ****************************************
#***** @task:    see x1_eigenproblemPOD.f ***************************
#***** @revised: 2012-01-14 -> Syracuse (split up of 1 in 2 files) *****
#***** @version: 2006-06-26 Poitiers *********************************
#*******************************************************************
source x_defineFortranCompiler.sh

echo "Compiling x1_eigenproblemPOD ..."
$FortranCompiler x1_eigenproblemPOD.f xs_ioTERMINAL.f [...] || exit

echo "*** Fini ***"
```

Almost all execution shell scripts xL_Tool.sh share a very similar procedure:

1. First, the local variables, e.g. the PNG viewer, are imported from x_defineLocalVariables.sh.
2. The program is executed.
3. Finally, some data may be visualized with gnuplot or another auxiliary program.

x1_eigenproblemPOD.sh is displayed as one example of such a script.

```
#*******************************************************************
#***** @author:  Bernd Noack ****************************************
#***** @task:    see x1_eigenproblemPOD.f ***************************
```

```
#***** @revised: 2012-01-14 -> Syracuse (split up of 1 in 2 files) *****
#***** @version: 2006-06-26 Poitiers ************************************
#************************************************************************
source x_defineLocalVariables.sh

echo "Executing the program ..."
time ./yTools/Y1_eigenproblemPOD || exit

echo "Visualizing the results ..."
gnuplot ./xTools/x1_eigenproblemPOD.gnuplot || exit
echo "$PNGViewer gnu_SpectrumPOD.png" >> SHOW.sh

echo "*** Fini ***"
echo ">>> See figure in last line of << SHOW.sh >>:"
tail -1 SHOW.sh
```

13.2 Parameter files

We attempt to list the files in a form, which allows an easy link with the analytics of Sec. 2.

13.2.1 Files for the grid

Whole domain .... `y0_Grid.dat`

This file contains the name of the grid for the whole domain. A sample listing reads:

```
z0_Data/Grid.dat
```

This grid is loaded by subroutine `readGrid` and used by tools operating on the whole domain or on single snapshots. Examples are:

- `x0_plotGrid`;
- `x0_plotFlow`;
- `x1_POD`;
- `x1_GalerkinApproximation`.

Subdomain .... `y1_Domains.dat`

This file contains the names of the grids for the subdomains. A sample listing reads:

```
1 z0_Data/Grid.0001
2 z0_Data/Grid.0002
 ...
```

This grid is loaded by subroutine `readGridSub` and used by tools operating on <u>all</u> snapshots or <u>all</u> modes simultaneously. Examples are:

- `x1_CorrelationMatrixSub`;

- `x2_GalerkinProjectionSub`.

13.2.2 Files for the Galerkin expansion

Eq. (2.8), e.g. N_A, N .... `y1_GM.dat`

This file controls the dimensions of the Galerkin expansion and the construction of the modes. A sample listing for the Galerkin model of ? with 1 actuation mode, 1 shift mode and 2 POD modes reads:

```
1      [N_A:   number of actuation modes]
3      [N:     total number of modes]
2      [N_POD: number of POD/DMD/... modes]

1  2       ! identity
3          ! initialize u_D=0
3  0  1.0 ! linear combination u_D = u_0-u_s
3  4 -1.0 ! linear combination u
3  1  2   ! orthonormalize     u_D w.r.t. u_1...2
```

N_A and N are specified in the first and second line, respectively. The third line determines the number of POD (or DMD) modes $N_{\rm POD} \le N$.

The following lines contain the recipes according to which non-empirical modes shall be constructed. Lines with

i_{min}, i_{max} ! identity

describe the range of modes $i \in [i_{min}, i_{max}]$ which have been already determined, generally the empirical POD or DMD modes. These modes are loaded in the program. A line of the format

i ! initialize

implies that the i-th mode is zero-ed $\boldsymbol{u}_i \equiv 0$.

A line of the format

i j α ! linear combination ...

implies that α times $\boldsymbol{u}_j$ is added to the $\boldsymbol{u}_i$.

The line

i j k ! orthonormalize ...

implies that $\boldsymbol{u}_i$ is orthonormalized with respect to $\boldsymbol{u}_l$, $l = j, \ldots, k$.

The indices of the modes refer to modes mentioned in `y1_Modes.dat`. These indices may be out of the range of Galerkin expansion $i \in [-N_A, N]$. For instance, the construction of the shift mode uses the steady solution a (temporary) $N+1$ mode.

y1_GM.dat has to be created by the user. For a standard POD model, aToolsInitGM creates this file after a short user dialogue. y1_GM.dat is read in subroutine readYGA by following programs:

- x0_plotFlows: N is needed for the visualization of all modes.
- x1_CorrelationMatrixSub: N_a is needed to subtract the contributions of the actuation modes.
- x1_eigenproblemPOD: N is needed for the output. Only the first N POD eigenvalues and eigenvectors are written out.
- x1_GalerkinApproximation: In this tool, non-POD modes, like the shift mode, are added as specified.
- x1_GalerkinProjection: N_a and N are needed for computation of the right coefficients and for the output.

A complete listing of these programs can be obtained with

```
> grep "readYGA" x*.f
```

M .... y1_Flows.dat

This file contains the list of snapshots used to compute the empirical modes,

$$m \quad \mapsto \quad \texttt{file name of snapshot.}$$

The number of employed snapshots may be a subset of the number of available snapshots. This file is used to determine M for POD, DMD, etc. A sample listing reads:

```
1 z0_Data/Flow.0001
2 z0_Data/Flow.0002
...
```

y1_Flows.dat is typically created by the user. aToolsInitGM also creates this file after a short user dialogue. From the modelling example in Sec. 6, following tools read this list:

- x0_plotFlow which reads the file name of the selected mth snapshot from this list.
- x1_CorrelationMatrix which reads all snapshots for the correlation matrix.
- x1_POD which composes the POD modes from linear combinations of all snapshots for the correlation matrix.

A complete listing of these programs can be obtained with

```
> grep "readFilenameFlow1" x*.f
```

This subroutine translate m into the filename.

13.2.3 Files for the Galerkin system

Eq. (2.17)**, e.g.** Re, N_V . . . . y2_GM.dat
controls the composition of the Galerkin system. A sample listing for the example of Sec. 6 reads:

```
    100      [Re:  Reynolds number]
      0      [N_V: number of volume forces]
```

The first line determines the Reynolds number Re and the second number of volume forces N_V. Note that the volume forces enter at level 2 in the Galerkin projection, not earlier. The volume forces enter the dynamics (level 2), the Navier-Stokes equation, not the kinematics (level 1).

y2_GM.dat is generally created by the user. For a standard POD model, aToolsInitGM creates this file after a short user dialogue. y2_GM.dat is read in subroutine readYGP by following programs:

- x2_GalerkinProject: N_V is needed for the Galerkin representation of the volume force term
- x3_makeDS: $\nu = 1/Re$ is needed to construct the dynamical system (2.19) from (2.17).

A complete listing of these programs can be obtained with

```
> grep "readYGP" x*.f
```

$\boldsymbol{v}_i$ . . . . y2_Modes.dat
This file maps the index i to the file name of the corresponding projection modes $\boldsymbol{v}_i$ in (2.17),

$$i \quad \mapsto \quad \texttt{file name of projection mode},$$

e.g.

```
    1   z2_Data/Test.0001
    2   z2_Data/Test.0002
    .....
```

For the traditional Galerkin method, the test functions are the expansion modes, $\boldsymbol{u}_i = \boldsymbol{v}_i$, $i = 1, \ldots, N$. Currently the Petrov-Galerkin method with test functions different from the expansion modes is not fully implemented.

$\boldsymbol{g}_l$ . . . . y2_Forces.dat . . . . z0_Data/Force.0001 (example)
y2_Forces.dat contains the mapping from $l = 1 \,.\, N_V$ to the file name. A sample listing of y2_Forces.dat reads:

```
    1 z0_Data/Force.0001
```

y2_Forces.dat is generally created by the user. y2_Forces.dat is read in subroutine readFilenameForce by following program:

- x2_GalerkinProjection[Sub]: The force fields are needed for the Galerkin representation of the volume force term.

A more complete listing of these programs can be obtained with

```
> grep "readFilenameForce" x*.f
> grep "readForcesSub" x*.f
```

readForcesSub uses readFilenameForce.

13.2.4 Files for the dynamical system

z_i, $i = 1, \ldots, N$ .... y3_Modes.dat

This file contains the mapping

$$i \quad \mapsto \quad z_i = \pm 1, \quad i = 1, \ldots, N$$

for the descriptor system. A sample listing reads:

```
1  1
2  1
3  1
4 -1
```

This corresponds to a descriptor system with 3 differential equations and one algebraic one. In ?, such a system arises by a mean-field Galerkin model (a_1, a_2, a_3) for a cylinder wake stabilized by a volume force ($a_4 = b$). The volume force amplitude is determined by a control law $b = b(a_1, a_2)$, i.e. an algebraic equation.

A simple version of y3_Modes.dat ($z_i \equiv 1$, only differential equations) is created by x3_makeGS. xAMC provides no tools for setting up algebraic equations from manifolds and control laws. The set of possibilities is too rich.

y3_Modes.dat is read by subroutines readMode which is used in turn by other subroutines, like readQijk or initFlow. y3_Modes.dat is used by the tool x3_solveDS.

Eq. (2.23), e.g. q^+_{ijk}, $i = 1, \ldots, N$, $j, k = 0, \ldots, N$ .... y3_DS_Qijk.dat

This file defines the propagator of the descriptor system

$$i, j, k, \quad q^+_{ijk} \quad \text{for} \quad \begin{cases} i & = 1, \ldots, N \\ j & = 0, \ldots, N \\ k & = 0, \ldots, N \end{cases}$$

A sample listing reads:

```
1  0  0  0.1
1  1  0 -0.3
1  1  9  0.07
1  2  0 +0.4
...
9  9  9 -0.001
```

y3_DS_Qijk.dat is created from the Galerkin system (2.17) by x3_makeGS. The file is read by the tool x3_solveDS. x3_solveDS loads the descriptor system via calling subroutine initFlow, which calls readQijk.

t . . . . y3_Time.dat

This file controls the time integration in the interval $t_0 \leq t \leq t_1$. A sample listing reads:

```
    0.0        !   t_0
  100.0        !   t_1
    0.001      ! dt
    0.1        ! dt_Show
    0.1       !   dt_List
```

The first two numbers denote the integration interval. The last three numbers specify the integration time step, the frequency of run-time information and the time step for post-processing.

y3_Time.dat is created by x3_makeGS. If x3_makeGS finds z0_Times.dat, than t_0 corresponds to time of the first snapshot t^1 and t_1 corresponds to the time of the last snapshot t^M. The listing frequency (dt_List) for z3_Amplitudes.dat is inferred from z0_Times.dat as well. Thus, z1_Times.dat and z3_Times.dat should be similar. The other parameters (dt and dt_Show) are chosen by experience which may or may not apply to the considered example. If x3_makeGS does not find z0_Times.dat, default values are taken.

The file is read by the tool x3_solveDS. x3_solveDS loads the descriptor system via calling subroutine initFlow, which in turn calls readQijk.

$a_i(t_0), i = 1, \ldots, N$ . . . . y3_State.dat

This file determines the initial condition for the integration. A sample listing for $a_1(t_0) = 0.02$, and $a_i = 0$, $i > 1$ reads:

```
1 0.02
```

Only non-zero values are listed.

y3_State.dat is created by x3_makeGS. If x3_makeGS finds z1_Amplitudes.dat, than $a_i(t_0)$ corresponds to amplitudes of the first snapshot. Thus, z1_Amplitudes.dat and z3_Amplitudes.dat should be similar for an accurate Galerkin model. If x3_makeGS does not find z1_Amplitudes.dat, default values are taken.

The file is read by the tool x3_solveDS via calling subroutine initState.

N is determined as the largest index i from y3_DS_Qijk.dat. Note that this N corresponds to N^+ in (2.22) and my be larger than the N of (2.8).

13.3 Data files

The following subsections list important data files. Section 7 has shown how these files are created or post-processed.

13.3.1 Files related to the IBVP

$\boldsymbol{x}, \boldsymbol{u}$ z0.dat

This file describes grid discretization. A sample listing reads:

```
2u
...
```

for a 2-D unstructured grid. Currently, only aToolCompile reads this file. It executes only compilation shell scripts x[0..4,X]_⋆_0.sh which are either grid-independent or contain the right grid specifier (here: '2u'). Future versions of xAMC may provide more information in z0.dat.

13.3.2 Files with the flow data

$\boldsymbol{u}^m$ .... z0_Flows.dat .... z0_Data/Flow.0001 (example)

maps $m = 1, \ldots, M$ to the corresponding name of the data file,

$$m \quad \mapsto \quad \texttt{file name of snapshot}$$

A sample listing reads:

```
1 z0_Data/Flow.0001
2 z0_Data/Flow.0002
.....
2 z0_Data/Flow.0058
```

a_i^m, b_l^m .... z0_Times.dat

z0_Times.dat records the sampling instants,

$$m \quad \mapsto \quad t_m$$

e.g.

```
1  0.00
2  0.10
...
```

a_i^m, b_l^m .... z0_Amplitudes.dat

z0_Amplitudes.dat contains the actuation amplitudes of volume forces and boundary actuators,

$$m, l \quad \mapsto \quad \begin{cases} b_l^m & l > 0 \\ a_l^m & l < 0 \end{cases} \quad \text{for} \quad \begin{cases} m = 1, \ldots, M \\ l = -N_A, \ldots, -1, 1, \ldots, N_V \end{cases}$$

e.g.

```
1 -1 0.47 ! a_-1
1 +1 0.50 ! b_1
2 -1 0.43 ! a_-1
2 +1 0.54 ! b_1
...
```

The first index identifies the snapshot m and the second index l corresponds to the volume force amplitude b_l if $l > 0$ and to a boundary actuator amplitude a_l if $l < 0$. The amplitude value is the last entry in each row.

13.3.3 Files for the Galerkin expansion

$\boldsymbol{u}_i$ .... z1_Modes.dat

This file maps i to the corresponding file name of the mode,

$$i \quad \mapsto \quad \texttt{file name of expansion mode},$$

including possibly also auxiliary quantities, like the steady Navier-Stokes solution for the shift mode.

```
1 z1_Data/Mode.0001
2 z1_Data/Mode.0002
...
```

a_i^m .... z1_Amplitudes.dat

This files contains the modal amplitudes for all snapshots,

$$m, i \quad \mapsto \quad a_i^m, \quad \text{for} \quad \begin{cases} m = 1, \ldots, M \\ i = N_A, \ldots, N \end{cases}$$

e.g.:

```
1  -1  0.47
1   0  1.0
1   1  1.47
1   2 -0.71
...
1  10 -0.01
2  -1  0.43
...
```

The first, second, and third number corresponds to m, i and a_i^m, respectively.

13.3.4 Files for the Galerkin system

m_{ij} .... z2_MassMatrix.dat

$$i, j, m_{ij} \quad \text{for} \quad \begin{cases} i = 1, \ldots, N \\ j = -N_A, \ldots, N \end{cases}$$

Only non-vanishing m_{ij} are listed.

l^{ν}_{ij} . . . . z2_ViscousTerm.dat

$$i,j, \quad l^{\nu}_{ij} \quad \text{for} \quad \begin{cases} i = 1,\ldots,N \\ j = -N_A,\ldots,N \end{cases} .$$

q^{c}_{ijk} . . . . z2_ConvectiveTerm.dat

$$i,j,k, \quad q^{c}_{ijk} \quad \text{for} \quad \begin{cases} i = 1,\ldots,N \\ j = -N_A,\ldots,N \\ k = -N_A,\ldots,N \end{cases} .$$

q^{p}_{ijk} . . . . z2_PressureTerm.dat

$$i,j,k, \quad q^{p}_{ijk} \quad \text{for} \quad \begin{cases} i = 1,\ldots,N \\ j = -N_A,\ldots,N \\ k = -N_A,\ldots,N \end{cases} .$$

The pressure term representation is not implemented in this current xAMC release. An implementation is planned in the near future.

Part V

xAMC – future developments

14. Summary and outlook

This report outlines the road-map for `xAMC` as a unifying modeling platform for control-oriented Galerkin models [??]. The current version allows to understand existing ROMs which presented in `xAMC` format. The version does not include detailed informations for programmers. Nor is the description complete with respect to current capabilities of `xAMC`.

Future versions of this report may contain a selection of following so far undocumented capabilities.

Domain decomposition: `xAMC` can be parallelized via a domain decomposition. This capability shall be described in a later version of the report.

DMD: a detailed description of DMD [??]. This is included already in `xAMC`.

Dynamical system solver: a more detailed description of the descriptor system, including center-manifold models, control design, etc. In addition, the fixed-point solver and stability solver may be described.

Finite-time thermodynamics solver: an inclusion and description of this package.

A synopsis of many important shell scripts: This list may contain for each shell script: (1) the task, (2) the input files, (3) the output files, (4) the employed algorithm and (5) general comments on the execution or performance.

A description of the FORTRAN programs including a listing of the main subroutines with I/O parameters and their purpose.

An appendix chapter on the development of `xAMC`: This includes rules for ownership and version management.

An appendix chapter with a general trouble shooting guide: This may include a listing of physical dimensions and corresponding files.

An appendix chapter on the version history.

15. Future milestones of `xAMC` development

Milestone 1: Include finite-dimensional state spaces and 1-D flow data. Existing `xAMC` toolkits for the mentioned state spaces shall be integrated in the new package. The packages may be validated with

1. Lorenz equation for finite-dimesional state spaces.
2. Ginzburg-Landau solutions for 1-D configurations;
3. The Stuart solution for 2-D shear flow;
4. The ABC solution for 3-D flows;

following partially the report by ?.

Milestone 2: Employ CGNS. The different grids shall be integrated in a CGNS-based `xAMC` toolkit. Resulting tasks are related to:

1. a data viewer (ADFviewer),
2. a visualization software (VisIt or ParaView),
3. a routine for reading the grid,
4. a routine for reading the flow, and
5. CGNS analogs for `x0_plotGrid` and `x0_plotFlow`.

Milestone 3: Implementation of model identification tools. This implies physics-based identification tools of 2007 and advanced methods of [?], possibly a combination of both.

Milestone 4: Mixing-layer and jet Galerkin models. Numerous application of `xAMC` are perceivable, once the previous milestones are achieved.

Milestones 2 and 4 of § 15 appear the most important and urgent for the team.

16. Acknowledgements

The development of xAMC as new modelling platform started in a visionary meeting at TU Berlin in January 2004. Since then, numerous collaborators have significantly contributed to the current version:

- Christoph Airiau tested xAMC in February 2010 and found incompatibilities for his Linux system, e.g. for file names. This observation has entered many routines and prevented a lot of future trouble.
- Oliver Lehmann advocated the strict separation in tool, processes and data files. He shaped the initial structure of xAMC in 2 weeks in Poitiers 2006 and immensely contributed in all developments.
- Guillaume Lehnasch suggested us in 2010 to include pseudo 3-D data structure, which reduced the number of grid-dependent routines to a minimum.
- Mark Luchtenburg tested the CGNS data format in 2009 in his alpha version of xAMC.
- Marek Morzyński generously provided his large CFD wisdom in many decisions regarding xAMC. Moreover, he integrated his Navier-Stokes solver UNS3 in a xAMC-based flow control studies.
- Clarence W. Rowley introduced one of the authors in the world of DMD and thus helped in the inclusion of another capability.
- Michael Schlegel had the ingenious idea to introduce ASCII lists (with indices) for all Galerkin model data. This made the xAMC much more readable.
- Gilead Tadmor wholeheartedly supported the xAMC project with many applications and in many discussions. Most of the applications have his signature.
- Tino Weinkauf shaped the choice of CGNS for future readable binary data formats.

Markus Abel, Guy Ben-Dov, Pierre Comte, Joël Delville, Peter Jordan, Franck Keherve and others motivated us in developing xAMC by emphasizing the need and suggesting new applications.