Berlin Institute of Technology
Department for Fluid Dynamics and Engineering Acoustics
Chair in Reduced-Order Modelling for Flow Control

Technical Report 01/2009

# An introduction to the POD Galerkin method for fluid flows with analytical examples and MATLAB source codes

**D.M. Luchtenburg**, **B.R. Noack** & **M. Schlegel**

August 2009

Berlin Institute of Technology MB1,
Müller-Breslau-Straße 11, D-10623 Berlin

# Abstract

This report provides a "hands-on" introduction to the proper orthogonal decomposition (POD) and the Galerkin method (GM). The POD is a technique to decompose a flow velocity field into spatial modes and time-dependent amplitudes. More specifically, POD extracts deterministic functions, associated with large-scale, energetic structures in a flow. If the governing equation of the field is known, the Galerkin method can be used to derive a system of ordinary differential equations for the evolution of the time-dependent amplitudes.

We focus on the application of the POD and the GM to fluid flows. Three examples are presented: a solution of the one-dimensional convection-diffusion equation, the two-dimensional Stuart solution and the three-dimensional (convecting) ABC flow. These examples serve as "prototypes" to show typical outcomes and problems associated with the POD and the GM. For all these examples, MATLAB source code is available to illustrate the practical implementation of the POD and GM.

Keywords: Proper orthogonal decomposition, Galerkin model, convection-diffusion equation, Stuart vortex, ABC flow

# Contents

# 1. Introduction

The main objective in proper orthogonal decomposition (POD) is to obtain an optimal low-dimensional basis for representing an ensemble of high-dimensional experimental or simulation data. This low-dimensional basis can in turn be used to formulate reduced-order models of complex flows. POD decomposes a given (fluctuating) flow field $\mathbf{u}'(\mathbf{x}, t)$ into an orthonormal system of spatial modes $\mathbf{u_i}(\mathbf{x})$ and corresponding (orthogonal) temporal coefficients $a_i(t)$

$$\mathbf{u}'(\mathbf{x}, t) = \sum_{i=1}^{N} a_i(t)\mathbf{u_i}(\mathbf{x}). \tag{1.1}$$

This basis is optimal in the sense that a truncated series expansion of the data in this basis has a smaller mean square truncation error than a representation by any other basis. The POD provides a natural ordering of the spatial modes by measure of their mean square temporal amplitude (i.e. their kinetic energy). In conjunction with the Galerkin method a system of ordinary differential equations, called the Galerkin system (GS), can be derived for the temporal evolution of the temporal amplitudes.

The term proper orthogonal decomposition was introduced by [Lumley 1967] as an objective definition of coherent structures. The POD is also known as the Karhunen-Loève expansion. Its discrete relatives are called principal component analysis (PCA) and singular value decomposition (SVD). The reader is referred to [Wu *et al.* 2003] for the relations. Tutorials on the discrete and continuous formulations of POD have been presented by [Chatterjee 2000] and [Cordier & Bergmann 2003]. For a thorough treatment of the continuous version of POD see [Holmes *et al.* 1998].

The manuscript is organized as follows. In chapter 2 we consider three analytical solutions of flow prototypes to demonstrate the practical implementation of the POD and the GM in chapter 4. The first type is the one-dimensional convection-diffusion equation, which is presented in more detail in §2.1. The second one is the two-dimensional Stuart solution, see §2.2. The last type is the three-dimensional ABC flow in §2.3. In chapter 3, the POD and the Galerkin method are summarized. In chapter 4, we apply the POD method and GM to all three flow prototypes. In appendix B, MATLAB source code is provided for the computation of the POD and the GS of the presented examples.

# 2. Analytical solutions

In this section, three analytical solutions to model equations are outlined. These solutions serve as prototypes for flow solutions in respectively one, two and three dimensions. The first type is a solution of the convection-diffusion equation. This solution exhibits the typical travelling wave structure. The last two types are solutions of the inviscid Euler equation. An exact two dimensional solution of this equation is the Stuart vortex describing the 'cat's eyes' in a mixing layer. Thirdly, the Arnold-Beltrami-Childress (ABC) flow is introduced. It is used to study a variety of phenomena (e.g. chaos, dynamos).

In §2.1, a solution to the one-dimensional convection-diffusion equation is described. In §2.2 the Stuart vortex is introduced. Lastly, the ABC flow is outlined in §2.3.

## 2.1 A solution of the convection-diffusion equation

The linear convection-diffusion equation is given by

$$\frac{\partial}{\partial t}\,u + \frac{\partial}{\partial x}\,u = \nu\,\frac{\partial^2}{\partial x^2}\,u, \tag{2.1}$$

where $u$ is the velocity, $t$ is the time, $x$ is the spatial co-ordinate and $\nu = 1/Re$ is the reciprocal of the Reynolds number. Boundary conditions are specified at $x = 0$ and $x \to \infty$

$$\begin{aligned} u(0,t) &= \cos(t), && \text{(2.2a)} \\ \lim_{x\to\infty} u(x,t) &= 0. && \text{(2.2b)} \end{aligned}$$

Initial conditions are not prescribed, since we are only interested in converged solutions (not transients).

The converged solution is of the form [Strang 1986]

$$u = e^{\imath(k\,x-t)}, \quad k = \beta + \imath\alpha,$$

with the constraints $\Re\{u(0,t)\} = \cos(t)$ and $\lim_{x\to\infty}\Re\{u(x,t)\} = 0$. Here $\imath$ denotes the imaginary unit, $k$ the complex wave number, $\alpha$ the growth rate, $\beta$ the real wave number, and $\Re$ the real part. The solution is given by [Noack 2006]

$$\begin{aligned} u(x,t) &= e^{-\alpha x}\,\cos(\beta x - t), && \text{(2.3a)} \\ \alpha &= \frac{1}{4\nu}\left[\sqrt{2+2\sqrt{1+16\,\nu^2}}-2\right], && \text{(2.3b)} \\ \beta &= \frac{1}{4\nu}\left[\sqrt{-2+2\sqrt{1+16\,\nu^2}}\right]. && \text{(2.3c)} \end{aligned}$$

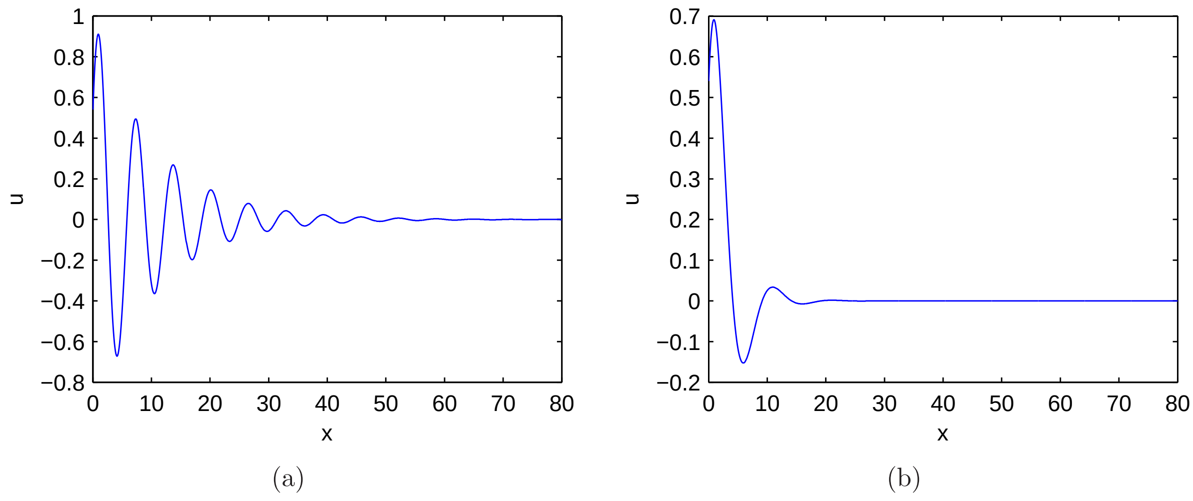


Figure 2.1: Two snapshots at $t = 1$ of the solution (2.3) of the convection-diffusion equation: $\nu = 0.1$ (a) and $\nu = 1$ (b).

This solution describes an exponentially decaying spatial wave with a constant wave number. In fig. 2.1(a) a snapshot is shown for a small viscosity of $\nu = 0.1$, and in fig. 2.1(b) for $\nu = 1$.

A Taylor expansion around $\nu_0$ shows, in the limit of small viscosity $\nu_0 \rightarrow 0$, the decay rate and wave number simplify to

$$\alpha = \nu + O(\nu^3), \tag{2.4a}$$
$$\beta = 1 + O(\nu^2). \tag{2.4b}$$

The wave number converges to the corresponding value of the solution $u = \cos(x - t)$ of the convection equation $\partial_t u + \partial_x u = 0$ with the boundary condition (2.2).

Comparing equations (2.4) and (2.3), we observe that the temporal growth rate $\alpha$ in a moving reference frame with a velocity of 1 is approximately equal to the viscosity coefficient $\nu$ for small viscosities. This observation can be used to compute the spatial dissipation from the temporal decay and vice versa.

## 2.2 The Stuart vortex

The vorticity equation for two-dimensional motion of the inviscid incompressible fluid is of the form

$$\frac{\partial}{\partial t}\triangle\Psi + \frac{\partial\Psi}{\partial y}\frac{\partial}{\partial x}\triangle\Psi - \frac{\partial\Psi}{\partial x}\frac{\partial}{\partial y}\triangle\Psi = 0, \tag{2.5}$$

where $\Psi$ is the stream function, $t$ is the time, $x$ is the co-ordinate in the direction of mean flow and $y$ is the co-ordinate normal to that direction. By definition of the stream function, the $x$

and $y$ velocity components are

$$u = \frac{\partial \Psi}{\partial y}, \tag{2.6a}$$

$$v = -\frac{\partial \Psi}{\partial x}. \tag{2.6b}$$

We consider the following solution of (2.5)

$$\Psi = \ln(C\cosh(y) + A\cosh(x)), \tag{2.7}$$

where the constants $C$ and $A$ are assumed to be positive and related by $A = \sqrt{C^2 - 1}$. As $C$ ranges from $1$ to $\infty$, the flow represented by (2.7) ranges from the laminar shear layer $u = \tanh(y)$ to the flow due to a set of point vortices on the horizontal axis. This solution is derived in [Stuart 1967] and generally known as the Stuart vortex.

The corresponding velocity components are

$$u = \frac{C\sinh(y)}{C\cosh(y) + A\cos(x)}, \tag{2.8a}$$

$$v = \frac{A\sin(x)}{C\cosh(y) + A\cos(x)}. \tag{2.8b}$$

For a non-trivial demonstration of the POD (see eq. (1.1)), a time-dependent solution is required, not only a steady solution. Using the Galilean invariance property (see appendix A) of the Navier-Stokes equation, we can construct the following time-varying solution

$$u = \frac{\sinh(y)}{\cosh(y) + \epsilon\cos(x - ct)} + c, \tag{2.9a}$$

$$v = \frac{\epsilon\sin(x - ct)}{\cosh(y) + \epsilon\cos(x - ct)}, \tag{2.9b}$$

where $c$ is the convection velocity and $\epsilon = A/C$. In fig. 2.2 a snapshot of the Stuart vortex is shown.

## 2.3 The ABC flow

An ideal fluid is governed by the equation of continuity for an incompressible flow

$$\nabla \cdot \mathbf{u} = 0, \tag{2.10}$$

and the (inviscid) Euler equation

$$\frac{\partial}{\partial t}\mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p. \tag{2.11}$$

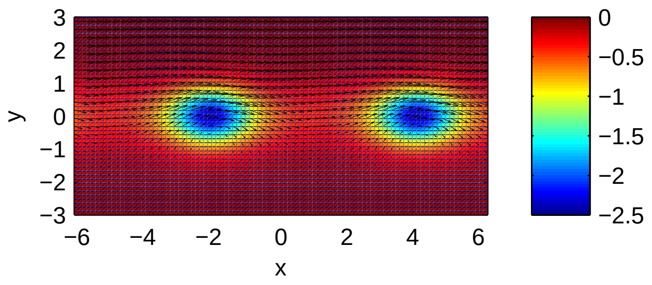


Figure 2.2: A snapshot of the Stuart solution (2.9) at $t = 1$ ($c = 1$ and $\epsilon = 0.42$). The vectors represent the velocity field and the colors indicate the strength of the vorticity.

The ABC flow is a stationary solution to the above equations, with periodic boundary conditions

$$u = A\sin(z) + C\cos(y), \tag{2.12}$$

$$v = B\sin(x) + A\cos(z), \tag{2.13}$$

$$w = C\sin(y) + B\cos(x), \tag{2.14}$$

where $A$, $B$ and $C$ are real constants. The spatial domain for the flow is defined over the cube $[0, 2\pi]^3$. As in §2.2, we use Galilean invariance (see appendix A) to construct a time-varying solution as a non-trivial input for the POD

$$u = A\sin(z - c_3 t) + C\cos(y - c_2 t) + c_1, \tag{2.15a}$$

$$v = B\sin(x - c_1 t) + A\cos(z - c_3 t) + c_2, \tag{2.15b}$$

$$w = C\sin(y - c_2 t) + B\cos(x - c_1 t) + c_3, \tag{2.15c}$$

where $\mathbf{c} = [c_1, c_2, c_2]$ is the convection velocity. In fig. 2.3 a snapshot of the ABC flow is shown.

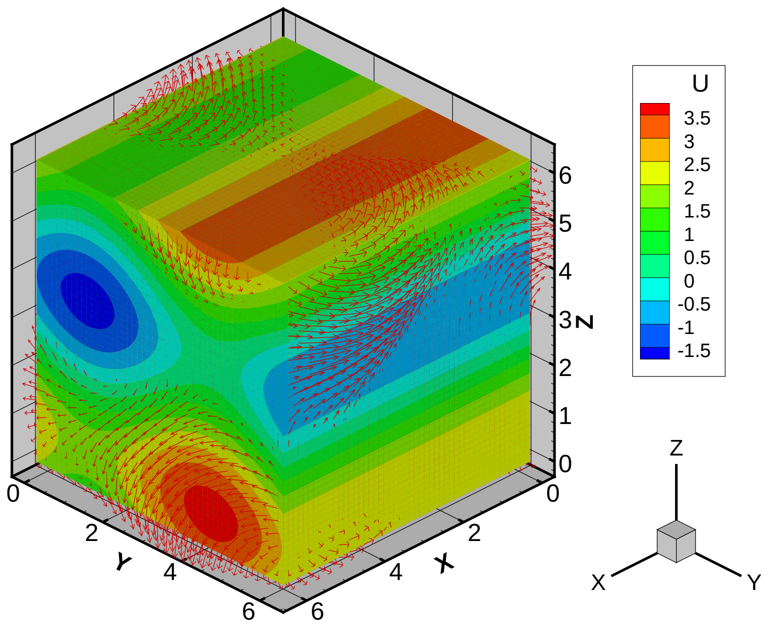


Figure 2.3: A snapshot of the ABC flow (2.15) at $t = \pi$. The constants are $A = \sqrt{3}$, $B = \sqrt{2}$, $C = 1$, $c_1 = 1$, $c_2 = \sqrt{2}$ and $c_3 = \sqrt{3}$. The vectors represent the velocity field and the colors indicate the strength of the u-velocity.

# 3. Methods

## 3.1 Proper Orthogonal Decomposition

In this section, the proper orthogonal decomposition (POD) is summarized. First the POD is described for velocity fields in analytical form. The decomposition can be performed in the spatial and the temporal domain. Secondly, we discuss the method of snapshots, which is indispensable for high-dimensional numerical data. The most important steps for practical implementation and some important properties are recapitulated. For a more detailed introduction to the POD the reader is referred to [Holmes *et al.* 1998, Cordier & Bergmann 2003].

### 3.1.1 Decomposition in the spatial domain

We start with some definitions. The inner product of two vector fields $\mathbf{u}$ and $\mathbf{v}$ is defined as

$$(\mathbf{u}, \mathbf{v})_\Omega := \int_\Omega d\mathbf{x}\, \mathbf{u} \cdot \mathbf{v}, \tag{3.1}$$

and the induced norm by

$$\|\mathbf{u}\|_\Omega := \sqrt{(\mathbf{u}, \mathbf{u})_\Omega}\,. \tag{3.2}$$

The time average of a quantity $a$ over a period $T$ is defined as

$$\overline{a} := \frac{1}{T} \int_0^T dt\, a. \tag{3.3}$$

The velocity field is decomposed as follows

$$\mathbf{u}(\mathbf{x}, t) = \mathbf{u}_0(\mathbf{x}) + \mathbf{u}'(\mathbf{x}, t), \tag{3.4}$$

where $\mathbf{u}_0$ is a steady base flow and $\mathbf{u}'$ the fluctuating velocity field. The goal is to represent the fluctuating velocity field by the proper orthogonal decomposition

$$\mathbf{u}'(\mathbf{x}, t) = \sum_{i=1}^{N} a_i(t)\, \mathbf{u}_i(\mathbf{x}), \tag{3.5}$$

where $\{\mathbf{u}_i\}_{i=1}^N \subset \mathcal{L}_2(\Omega)$ is the POD basis and time dependency is described by the amplitudes $a_i$.

Premise for the POD in the spatial domain is spatial correlation (coherence) of the velocity flow field. The two-point autocorrelation tensor for the flow field is defined by

$$\mathbf{R}(\mathbf{x}, \mathbf{y}) = \overline{\mathbf{u}'(\mathbf{x}, t) \otimes \mathbf{u}'(\mathbf{y}, t)}, \tag{3.6}$$

or in index notation

$$R_{\alpha\beta} = \overline{u'_\alpha(\mathbf{x},t)u'_\beta(\mathbf{y},t)}, \tag{3.7}$$

where the greek symbols refer to the velocity components, i.e. in three dimensions $\mathbf{u}' = [u'_1, u'_2, u'_3]$. The following eigenvalue problem needs to be solved to obtain the $i$-th POD mode $\mathbf{u}_i$ with eigenvalue $\lambda_i$

$$\int_\Omega d\mathbf{y}\; \mathbf{R}(\mathbf{x},\mathbf{y})\mathbf{u}_i(\mathbf{y}) = \lambda_i \mathbf{u}_i(\mathbf{x}). \tag{3.8}$$

(This is the very definition of a POD mode). The autocorrelation tensor is self-adjoint and positive semidefinite [Holmes *et al.* 1998]. The first property implies implies that the modes are orthogonal. The modes are ordered with respect to the decreasing real positive eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \ldots > 0$. Note that zero eigenvalues are not considered since they do not contribute to the velocity. Using the orthonormality of the modes

$$(\mathbf{u}_i, \mathbf{u}_j)_\Omega = \delta_{ij}, \tag{3.9}$$

the time-dependent amplitudes follow from the projection

$$a_i(t) = (\mathbf{u}'(\mathbf{x},t), \mathbf{u}_i(\mathbf{x}))_\Omega. \tag{3.10}$$

The first and second statistical moments of the temporal coefficients are

$$\overline{a_i} = 0, \tag{3.11a}$$
$$\overline{a_i a_j} = \lambda_i\, \delta_{ij}. \tag{3.11b}$$

The second statistical moment shows that the modal amplitudes are orthogonal in time.

The autocorrelation function can be expanded using the POD modes

$$\mathbf{R}\,(\mathbf{x},\mathbf{y}) := \sum_{i=1}^{\infty} \lambda_i\, \mathbf{u}_i(\mathbf{x}) \otimes \mathbf{u}_i(\mathbf{y}). \tag{3.12}$$

Note that the Reynolds stress is a special case of the autocorrelation tensor when $\mathbf{x} = \mathbf{y}$. The trace of the Reynold tensor is twice the turbulent kinetic energy

$$q(\mathbf{x}) := \frac{1}{2}\,\overline{\mathbf{u}'(\mathbf{x},t)\cdot\mathbf{u}'(\mathbf{x},t)} = \frac{1}{2}\sum_{\alpha=1}^{3} R_{\alpha\alpha}(\mathbf{x},\mathbf{x}).$$

The turbulent kinetic energy in the complete domain is

$$\mathcal{K}_\Omega = \frac{1}{2}\,\overline{(\mathbf{u}',\mathbf{u}')_\Omega} = \frac{1}{2}\sum_{i=1}^{\infty}\lambda_i. \tag{3.13}$$

## Calculation of the POD modes using a basis ansatz

Assume that the fluctuating velocity field is given by

$$\mathbf{u}'(\mathbf{x},t) = \sum_{i=1}^{N} b_i(t)\, \mathbf{v}_i(\mathbf{x}), \tag{3.14}$$

where $\{\mathbf{v}_i\}_{i=1}^{N} \subset \mathcal{L}_2(\Omega)$ is a basis and $b_i$ are the temporal amplitudes.

POD mode $i$ can be written as a linear combination of the basis vectors in $(3.14)$

$$\mathbf{u}_i = \sum_{j=1}^{N} T_{ij}\, \mathbf{v}_j. \tag{3.15}$$

Substitution of $(3.14)$ and $(3.15)$ in $(3.8)$ yields

$$\sum_{j=1}^{N}\sum_{k=1}^{N} Q_{jk}\, T_{ik}\mathbf{v}_j(\mathbf{x}) = \lambda_i \sum_{j=1}^{N} T_{ij}\mathbf{v}_j(\mathbf{x}), \tag{3.16}$$

where the matrix $\mathbf{Q}$ is

$$Q_{jk} = \sum_{m=1}^{N} \overline{b_j(t) b_m(t)}\, (\mathbf{v}_k, \mathbf{v}_m)_\Omega\, . \tag{3.17}$$

Since the basis vectors are linearly independent, eq. $(3.16)$ must hold for each $j$ separately

$$\sum_{j=1}^{N}\left(\sum_{k=1}^{N} Q_{jk}\, T_{ik} - \lambda_i T_{ij}\right)\mathbf{v}_j(\mathbf{x}) = 0. \tag{3.18}$$

This leads to an eigenvalue problem and can be written in matrix notation

$$\mathbf{Q}\, \mathbf{t}^{[i]} = \lambda_i \mathbf{t}^{[i]}, \tag{3.19}$$

where the weights are defined by the $j$-th entry of the $i$-th eigenvector, $T_{ij} = \mathbf{t}^{[i]}\big|_j$.

The modes can now be constructed with $(3.15)$. It is common practice to normalize the modes, see $(3.9)$, hence eq. $(3.15)$ is replaced by

$$\mathbf{u}_i = \sum_{j=1}^{N} T'_{ij}\, \mathbf{v}_j, \tag{3.20}$$

where $T'_{ij}$ is the scaled weight $T_{ij}$ such that $\|\mathbf{u}_i\|_\Omega = 1$.

### 3.1.2 Decomposition in the temporal domain

The decomposition $(3.5)$ can be read as an expansion with spatial modes $\mathbf{u}_i$ and temporal coefficients $a_i$ or alternatively as an expansion with temporal modes $a_i$ and spatial coefficients $\mathbf{u}_i$. This symmetry implies that time and space can be interchanged. Integration over space is replaced by integration over the time domain $[0, T]$. Thus the orthonormality in space and time, respectively expressed by eq. $(3.9)$ and $(3.11\text{b})$ is analogue.

Similarly the spatial correlation $(3.7)$ is exchanged for the temporal correlation function

$$C\,(t,s) = \left(\mathbf{u}'(\mathbf{x},t), \mathbf{u}'(\mathbf{x},s)\right)_\Omega . \tag{3.21}$$

The eigenproblem for the eigenfunction $a_i$ with eigenvalue $\mu_i$ is

$$\frac{1}{T}\int_0^T ds\, C\,(t,s)\;a_i(s) = \mu_i\, a_i(t). \tag{3.22}$$

As in §3.1.1, the modes are ordered with respect to the decreasing real positive eigenvalues $\mu_1 \geq \mu_2 \geq \mu_3 \geq \ldots > 0$. These eigenvalues are identical to the ones obtained from the decomposition in the spatial domain, i.e. $\mu_i = \lambda_i$.

The self-adjointness of the temporal autocorrelation function implies the orthogonality of the temporal coefficients. The coefficients are scaled such that

$$\overline{a_i a_j} = \frac{1}{T}\int_0^T dt\, a_i(t)\, a_j(t) = \lambda_i\, \delta_{ij}. \tag{3.23}$$

Thus we have the exact same scaling as before (compare with $(3.11\text{b})$).

Using this scaling, the POD modes are calculated by the projection

$$\mathbf{u}_i(\mathbf{x}) = \frac{1}{\lambda_i}\overline{a_i(t)\;\mathbf{u}'(\mathbf{x},t)}. \tag{3.24}$$

The autocorrelation function is expanded, in complete analogy to $(3.12)$, as follows

$$C(t,s) = \sum_{i=1}^{\infty} a_i(s)\, a_i(t). \tag{3.25}$$

In summary, we have obtained the same information in the time domain as in the spatial domain. For numerical problems this equivalence can be advantageous. Depending on the number of samples and the grid size, the eigenproblem may be computed in the spatial or temporal domain. The discretization in the temporal domain is discussed in the next section.

### 3.1.3 Method of snapshots

The method of snapshots [Sirovich 1987] is a discretization of the POD procedure in the temporal domain. Let an ensemble of $M$ snapshots be given at the discrete times $t_m$,

$$\mathbf{u}^{(m)}(\mathbf{x}) = \mathbf{u}(\mathbf{x}, t_m), \quad m = 1, 2, \ldots M. \tag{3.26}$$

The base flow $\mathbf{u}_0$ is defined as the ensemble average

$$\mathbf{u}_0(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} \mathbf{u}^{(m)}(\mathbf{x}). \tag{3.27}$$

Each snapshot is Reynolds decomposed into the mean and instantaneous fluctuation

$$\mathbf{u}(\mathbf{x}, t_m) = \mathbf{u}_0(\mathbf{x}) + \mathbf{u}'(\mathbf{x}, t_m). \tag{3.28}$$

The discrete form of the correlation function is a matrix $\mathbf{C}$, which in index notation reads

$$C_{mn} := \frac{1}{M} \left(\mathbf{u}'(\mathbf{x}, t_m), \mathbf{u}'(\mathbf{x}, t_n)\right)_\Omega , \tag{3.29}$$

where $m, n = 1, \ldots, M$. This spatial inner product can be approximated by a suitable quadrature rule.

Similarly to the continuous case, an eigenproblem needs to be solved

$$\mathbf{C}\, \mathbf{a}^{[i]} = \lambda_i\, \mathbf{a}^{[i]} . \tag{3.30}$$

The eigenvectors of the correlation matrix are the temporal coefficients $\mathbf{a}^{[i]} = \left(a_1^{[i]}, \ldots, a_M^{[i]}\right)$. The symmetry of $\mathbf{C}$ implies non-negative eigenvalues and the orthogonality of the eigenvectors. The time-dependent amplitudes are scaled such that

$$\overline{a_i a_j} = \frac{1}{M} \sum_{m=1}^{M} a_m^{[i]}\, a_m^{[j]} = \lambda_i\, \delta_{ij}. \tag{3.31}$$

Using this scaling, the POD modes are computed as

$$\mathbf{u}_i = \frac{1}{M\lambda_i} \sum_{m=1}^{M} a_m^{[i]} \left(\mathbf{u}^{(m)} - \mathbf{u}_0\right) . \tag{3.32}$$

These last two equations are the discrete equivalents of $(3.23)$ and $(3.24)$. The time average is estimated by the ensemble mean.

Note that the POD discretization in the temporal domain yields a correlation matrix with has size $M \times M$. The same discretization in the spatial domain would yield a matrix with size $(N_g N_d) \times (N_g N_d)$, where $N_g$ is the number of grid points and $N_d$ is the number of velocity components. The method of snapshots is the method of choice for CFD data, whereas the discretization in the spatial domain is more suited for long time samples of few experimental sensor measurements.

## 3.2 Galerkin system

An ordinary differential equation governing the temporal coefficients $a_i(t)$ is obtained by substituting the velocity field expansion (3.5) into the governing equation $\mathcal{N}$ and projecting onto the subspace spanned by the modes $\mathbf{u}_i$

$$(\mathcal{N}\left[\mathbf{u}(\mathbf{x},t)\right], \mathbf{u}_i(\mathbf{x}))_\Omega = 0 \quad \text{for } i = 1, \ldots, N, \tag{3.33}$$

where $N$ is the number of modes. Let $\mathcal{N}$ be the Navier–Stokes operator

$$\mathcal{N}\left[\mathbf{u}\right] = \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p - \nu \nabla^2 \mathbf{u} = 0. \tag{3.34}$$

Invoking (3.4) and (3.5), the resulting Galerkin system is quadratically nonlinear

$$\frac{d}{dt} a_i = \nu \sum_{j=0}^{N} l_{ij}\, a_j + \sum_{j,k=0}^{N} q_{ijk}\, a_j\, a_k + f_i^p \quad \text{for } i = 1, \ldots, N. \tag{3.35}$$

Here $a_0 \equiv 1$, $l_{ij}$ is the dissipation term

$$l_{ij} = (\mathbf{u}_i, \triangle \mathbf{u}_j)_\Omega, \tag{3.36}$$

$q_{ijk}$ the convection term

$$q_{ijk} = -(\mathbf{u}_i, (\mathbf{u}_j \cdot \nabla)\mathbf{u}_k)_\Omega \tag{3.37}$$

and $f_i^p$ the term that follows from projection of the pressure gradient

$$f_i^p = -(\mathbf{u}_i, \nabla p)_\Omega. \tag{3.38}$$

The incompressibility of the modes can be used to express this term in a boundary integral only

$$f_i^p = -\left(\mathbf{u}_i,\, \nabla p\right)_\Omega = -\left(\nabla \cdot [p\,\mathbf{u}_i]\right)_\Omega = -\int_{\partial\Omega} d\mathbf{A}\, p\mathbf{u}_i \cdot \mathbf{n} = -\left[p\,\mathbf{u}_i\right]_{\partial\Omega}. \tag{3.39}$$

This term is zero for the presented examples in chapter 4. Note that this is not generally the case, see [Noack *et al.* 2005] for examples.

For numerical purposes (i.e. to reduce discretization errors associated with approximation of the spatial derivatives), Green's first identity can be used to rewrite the dissipation term such that it only contains derivatives of first order

$$l_{ij} = -\left(\nabla \otimes \mathbf{u}_i :\ \nabla \otimes \mathbf{u}_j\right)_\Omega + \left[\mathbf{u}_i \cdot \nabla \otimes \mathbf{u}_j\right]_{\partial\Omega}, \tag{3.40}$$

where the $:$ symbol denotes a double contraction. For matrices $\mathbf{A} = (A_{lm})$ and $\mathbf{B} = (B_{lm})$ the contraction is defined as

$$\mathbf{A} : \mathbf{B} \quad := \quad \sum_{l,m} A_{lm}\, B_{ml}.$$

The second term in (3.40) is an integral over the boundary $\partial\Omega$ of the domain $\Omega$.

# 4. Applications

## 4.1 Convection diffusion equation

In this section, the proper orthogonal decomposition and the Galerkin projection are demonstrated for the one dimensional convection-diffusion equation and its solution, which were introduced in §2.1. Since the analytics are straightforward, we present all results in analytical form. In appendix B, MATLAB source code is listed to compute the POD and the GS.

### 4.1.1 Proper orthogonal decomposition

The solution $(2.3)$ to the one dimensional diffusion equation is expanded as follows

$$u(x,t) = \cos(t)\, e^{-\alpha x} \cos(\beta x) + \sin(t)\, e^{-\alpha x} \sin(\beta x) = \sum_{i=1}^{2} b_i(t)\, v_i(x), \tag{4.1}$$

where $\alpha = \alpha(\nu)$ and $\beta = \beta(\nu)$ are respectively defined by $(2.3b)$ and $(2.3c)$. Note that the base solution equals zero. This form of the solution can be directly used to compute the POD as outlined in §3.1.1.

We set the period $T = 2\pi$ and the spatial domain is the non negative real axis $\Omega = [0, \infty]$. The first step is the computation of the matrix $\mathbf{Q}$ (eq. $(3.17)$). As an example, element $Q_{11}$ is computed as

$$\begin{aligned} Q_{11} = & \left(\frac{1}{2\pi}\int_0^{2\pi} \cos^2(t)\, dt\right) \left(\lim_{x_0\to\infty}\int_0^{x_0} e^{-2\alpha x}\cos^2(\beta x)\, dx\right) \\ & + \left(\frac{1}{2\pi}\int_0^{2\pi} \cos(t)\sin(t)\, dt\right)\left(\lim_{x_0\to\infty}\int_0^{x_0} e^{-2\alpha x}\sin(\beta x)\cos(\beta x)\, dx\right). \end{aligned} \tag{4.2}$$

The second term is zero and the spatial integral can be easily calculated using complex notation.

In summary, the complete matrix $\mathbf{Q}$ is given by

$$\mathbf{Q} = \frac{1}{8\,\alpha\,(\alpha^2+\beta^2)} \begin{pmatrix} \beta^2 + 2\,\alpha^2 & \alpha\,\beta \\ \alpha\,\beta & \beta^2 \end{pmatrix}.$$

The second step is the solution of the eigenproblem associated with $\mathbf{Q}$, see $(3.19)$. The eigenvectors contain the weights $t_{ij}$ that are used to construct the POD modes. This leads to the third step in which the POD modes are calculated by $(3.15)$. After normalization the POD modes are given by

$$u_1(x) = \frac{\sqrt{2\alpha}}{\beta}\, e^{-\alpha x}\left[\beta\,\cos(\beta x) + \left(-\alpha + \sqrt{\alpha^2+\beta^2}\right)\,\sin(\beta x)\right], \tag{4.3a}$$

$$u_2(x) = -\frac{\sqrt{2\alpha}}{\beta}\, e^{-\alpha x}\left[\beta\,\cos(\beta x) + \left(-\alpha - \sqrt{\alpha^2+\beta^2}\right)\,\sin(\beta x)\right]. \tag{4.3b}$$

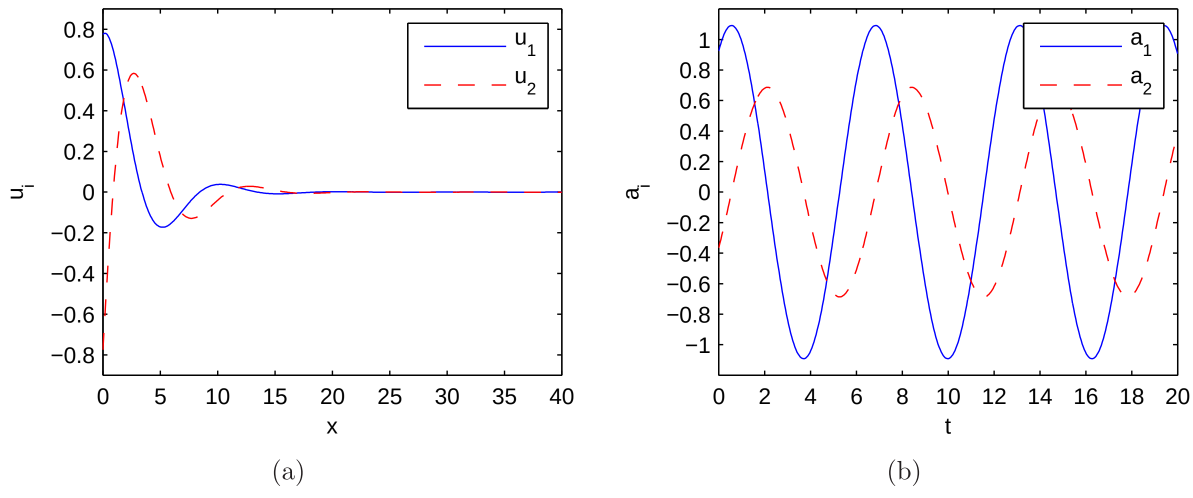


Figure 4.1: The POD for the solution (2.3) of the convection-diffusion equation, $\nu = 1$: the POD modes (a) and the temporal coefficients (b).

The corresponding temporal coefficients follow from the projection (3.10)

$$a_1(t) = \frac{\sqrt{2}}{4\sqrt{\alpha}\sqrt{\alpha^2+\beta^2}} \left[\left(\alpha + \sqrt{\alpha^2+\beta^2}\right)\cos t + \beta \sin t\right], \tag{4.4a}$$

$$a_2(t) = -\frac{\sqrt{2}}{4\sqrt{\alpha}\sqrt{\alpha^2+\beta^2}} \left[\left(-\alpha + \sqrt{\alpha^2+\beta^2}\right)\cos t - \beta \sin t\right]. \tag{4.4b}$$

In fig. 4.1(a) the POD modes are shown for $\nu = 1$, and in fig. 4.1(b) the corresponding temporal coefficients.

In the limit of small dissipation the POD simplifies to (compare with (2.4))

$$\lambda_{1,2} = \frac{1 \pm \nu + O\left(\nu^2\right)}{8\nu}, \tag{4.5a}$$

$$u_{1,2}(x) = \sqrt{\nu}\, e^{-\nu x} \left[\cos\left(x \mp \pi/4\right) + O\left(\nu^2\right)\right], \tag{4.5b}$$

$$a_{1,2}(t) = \frac{1}{4\sqrt{\nu}} \left[\cos\left(t \mp \pi/4\right) + O\left(\nu^2\right)\right]. \tag{4.5c}$$

The modes $u_{1,2}$ are decaying waves with a phase shift of $90°$. This phase shift guarantees the orthonormality of the modes $(u_1, u_2)_\Omega = 0$. For decreasing viscosity $\nu$ the spatial modes are stretched with $1/\nu$. Thus the factor $\sqrt{\nu}$ follows from the normalization $\|u_i\|_\Omega = 1$. In fig. 4.2(a) the POD modes are shown for a small viscosity ($\nu = 0.1$), and in fig. 4.2(b) the corresponding temporal coefficients. The eigenvalues $\lambda_{1,2}$ characterize the energy content of modes $u_{1,2}$ (see (3.13)). This energy content increases with decreasing viscosity, since the transversal waves exhibit slower decay (compare fig. 4.1(a) with 4.2(a)). Similarly, [Deane *et al.* 1991] found increasing energy levels for the modes of a Karman vortex street for increasing Reynolds numbers.

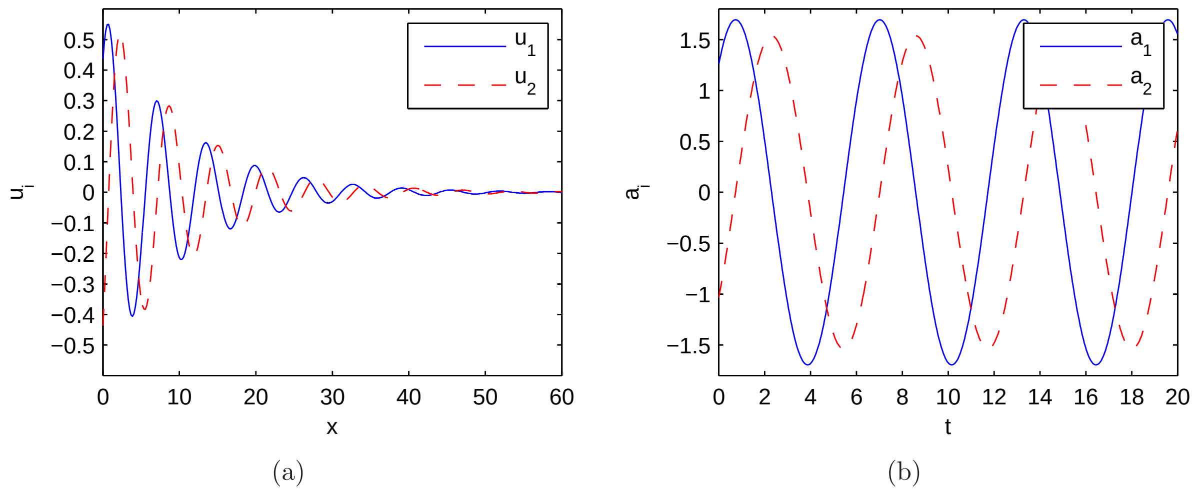


Figure 4.2: Same as fig. 4.1 but for $\nu = 0.1$.

### 4.1.2 Galerkin system

The expansion for the Galerkin system is

$$u(x,t) = a_1(t)u_1(x) + a_2(t)u_2(x), \tag{4.6}$$

where $u_{1,2}$ and $a_{1,2}$ are given by (4.3) and (4.4). The governing equation is the convection-diffusion equation

$$\mathcal{N}[u] = \partial_t u + \partial_x u - \nu \partial_{xx} u = 0. \tag{4.7}$$

The Galerkin projection (3.33) results in the following terms

$$(u_i, \partial_t [a_1\, u_1 + a_2\, u_2])_\Omega = \dot{a}_i, \tag{4.8}$$

$$(u_i, -\partial_x [a_1\, u_1 + a_2\, u_2])_\Omega = c_{i1}\, a_1 + c_{i2} a_2, \tag{4.9}$$

$$\left(u_i, \partial^2_{xx} [a_1\, u_1 + a_2\, u_2]\right)_\Omega = d_{i1}\, a_1 + d_{i2}\, a_2, \tag{4.10}$$

where the matrices are defined as

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} := \begin{pmatrix} \alpha & \alpha + \sqrt{\alpha^2+\beta^2} \\ \alpha - \sqrt{\alpha^2+\beta^2} & \alpha \end{pmatrix},$$

and

$$\begin{pmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{pmatrix} := \begin{pmatrix} \alpha^2 - \beta^2 & 2\alpha^2 + 2\,\alpha\,\sqrt{\alpha^2+\beta^2} \\ 2\alpha^2 - 2\,\alpha\,\sqrt{\alpha^2+\beta^2} & \alpha^2 - \beta^2 \end{pmatrix}.$$

In summary the Galerkin system reads

$$\dot{a}_i = \sum_{j=1}^{2} l_{ij}\, a_j, \tag{4.11}$$

where the matrix $\mathbf{L}$ is

$$\begin{pmatrix} l_{11} & l_{12} \\ l_{21} & l_{22} \end{pmatrix} := \begin{pmatrix} c_{11} + \nu\, d_{11} & c_{12} + \nu\, d_{12} \\ c_{21} + \nu\, d_{21} & c_{22} + \nu\, d_{22} \end{pmatrix} = \begin{pmatrix} 0 & c_{12} + \nu\, d_{12} \\ c_{21} + \nu\, d_{21} & 0 \end{pmatrix}.$$

This linear Galerkin system describes the evolution of the temporal coefficients (4.4). The matrix $\mathbf{L}$ is anti-symmetric and its diagonal entries are zero. This implies that its two eigenvalues are purely imaginary and conjugates of each other. Straightforward calculation yields $\lambda_{1,2} = \pm\imath$. Thus the solution of the Galerkin system is given by

$$\mathbf{a} = c_1 \mathbf{v}_1 \cos t + c_2 \mathbf{v}_1 \sin t, \tag{4.12}$$

where $\mathbf{a} = [a_1, a_2]$, $\mathbf{v}_{1,2}$ are the eigenvectors corresponding to $\lambda_{1,2}$ and the constants $c_{1,2}$ are determined by the initial conditions $a_{1,2}(t = 0)$. Comparing this result with the POD amplitudes given by $(4.4)$, the linear GS with the specific form of the matrix $\mathbf{L}$ could be anticipated, since the time-dependent amplitudes describe a purely convective motion. Solving for $c_{1,2}$, it is found that the solution of the GS is equal to the result of the POD given by $(4.4)$, which verifies that the GS is correct.

## 4.2 Stuart solution

In this section, the proper orthogonal decomposition and the Galerkin system are computed for the Stuart solution (see §2.2). This solution cannot be readily separated into temporal and spatial basis functions. Therefore we use the method of snapshots to compute the POD. In appendix B, MATLAB source code is listed to compute the POD and the GS.

### 4.2.1 Proper orthogonal decomposition

Here, we use the method of snapshots, see §3.1.3, to compute the POD of the Stuart solution $(2.9)$. Note that the solution is periodic in $x$-direction. In the $y$-direction the velocity field extends to $\pm\infty$. Whence, the spatial domain is defined by the rectangle $\Omega = [-2\pi, 2\pi] \times [-\infty, \infty]$.

For the numerical example, we set the convection velocity to $c = 1$, and the vortex strength to $\epsilon = 0.42$. The spatial integrals are approximated with a midpoint rule. For all practical calculations the domain can be bounded to $y = \pm 3$. This domain is uniformly discretized in $100 \times 50$ points in $x$- and $y$-direction. The ensemble for the POD consists of $50$ uniformly distributed snapshots over the period $T = 2\pi$. A characteristic snapshot is shown in fig. 2.2.

In fig. 4.3 the resulting (normalized) eigenspectrum of the correlation matrix $(3.29)$ is shown. The eigenvalues occur in pairs with equal energy content. This is generally the case for convecting structures. Fig. 4.4 shows the temporal evolution of the first six mode amplitudes. They also appear in pairs of equal energy, and frequency as well. The frequencies of the mode pairs are

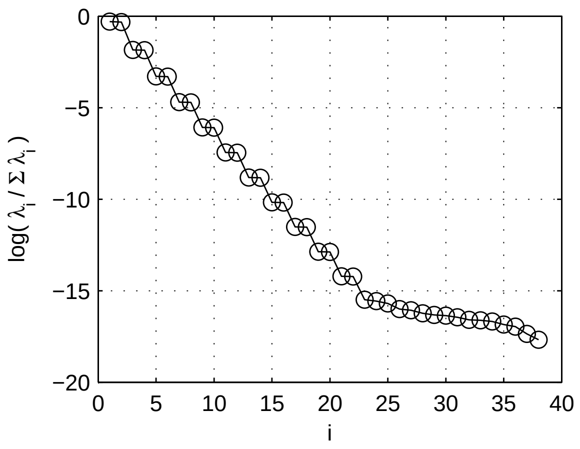


Figure 4.3: The POD spectrum corresponding to the Stuart solution (2.9) ($c = 1$ and $\epsilon = 0.42$).

simply integer multiples of the frequency $f = 1/2\pi$ defined by the convection velocity $c = 1$. This is most easily inferred from fig. 4.4 and the phase portraits in fig. 4.5. In other words, we have empirically shown that the POD yields the same result as a Fourier decomposition. The mean flow velocity field (ensemble average of the snapshots) is shown in fig. 4.6. The POD modes occur in pairs with the same wave number. The first six modes are shown in fig. 4.7.

Mode pairs that span a travelling wave have also been found in more realistic flows, for instance, in the boundary layer [Remfer & Fasel 1994] and the periodic cylinder wake [Deane *et al.* 1991].

### 4.2.2 Galerkin system

For the Galerkin system, we consider a truncated expansion of the velocity field using the first six modes

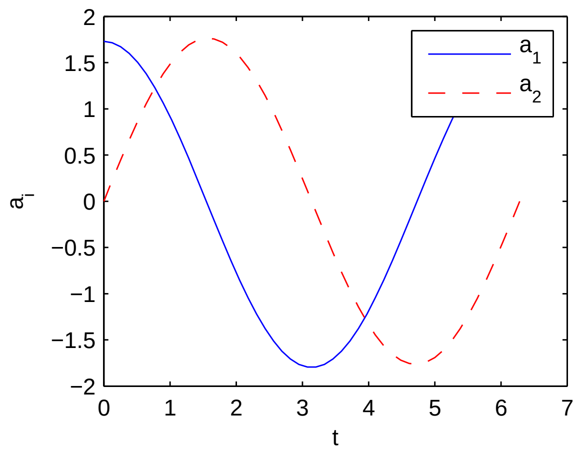
$$\mathbf{u}(\mathbf{x}, t) = \mathbf{u}_0(\mathbf{x}) + \sum_{i=0}^{6} a_i(t)\, \mathbf{u}_i(\mathbf{x}). \tag{4.13}$$

The Galerkin system is obtained by substituting this expansion into the Euler equation and projecting the residual onto the first six POD modes. Following §3.2 the Galerkin system is given by

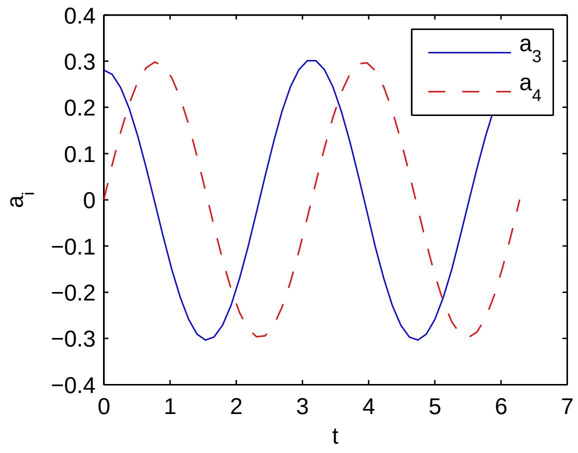
$$\frac{d}{dt} a_i = \sum_{j,k=0}^{N} q_{ijk}\, a_j\, a_k \quad \text{for } i = 1, \ldots, 6, \tag{4.14}$$

where

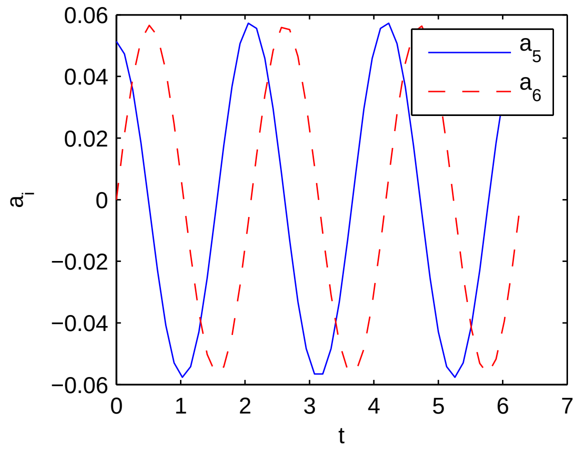
$$q_{ijk} = -(\mathbf{u}_i, (\mathbf{u}_j \cdot \nabla)\mathbf{u}_k)_\Omega \tag{4.15}$$



(a)



(b)



(c)

Figure 4.4: Temporal evolution of the modal amplitudes of the Stuart solution (2.9) ($c = 1$ and $\epsilon = 0.42$). Each subfigure shows a pair of coefficients with the same frequency.

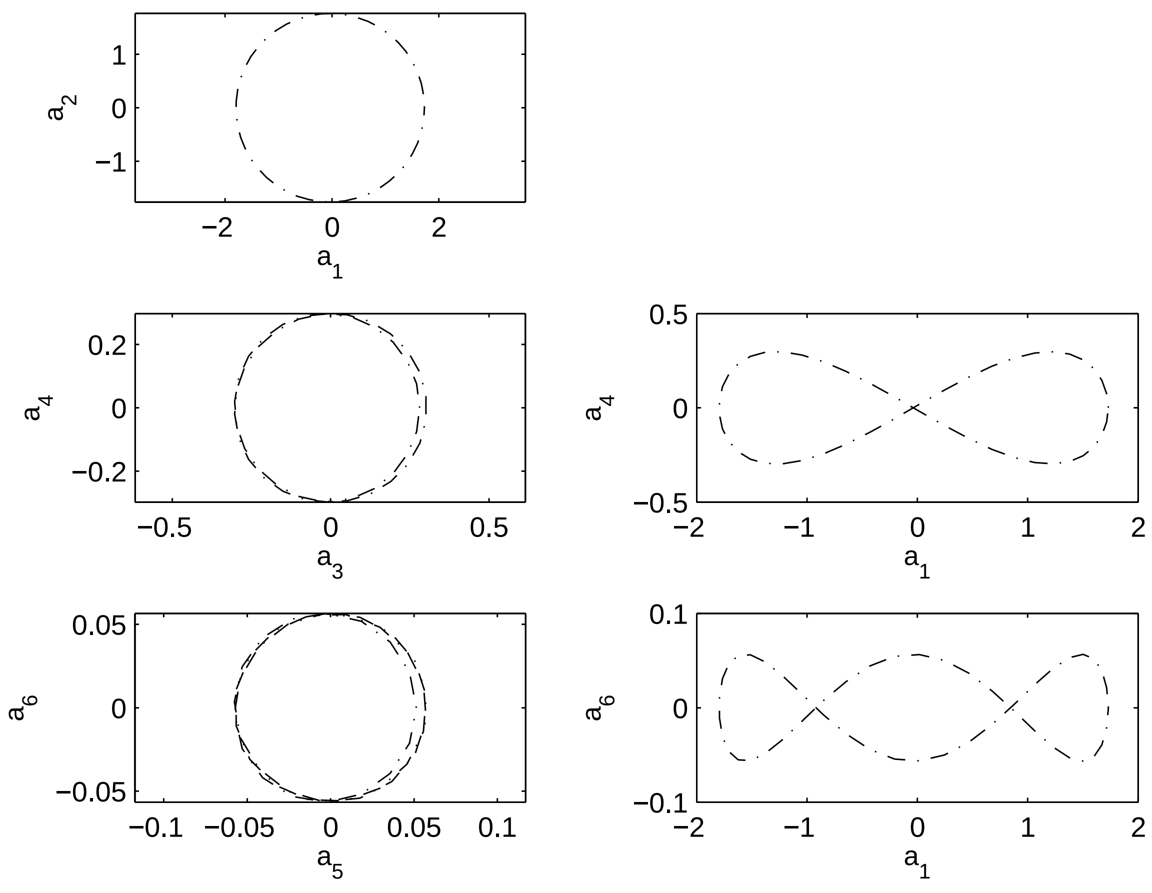


Figure 4.5: Phase portraits (Lissajous figures) of the temporal coefficients in fig. 4.4.

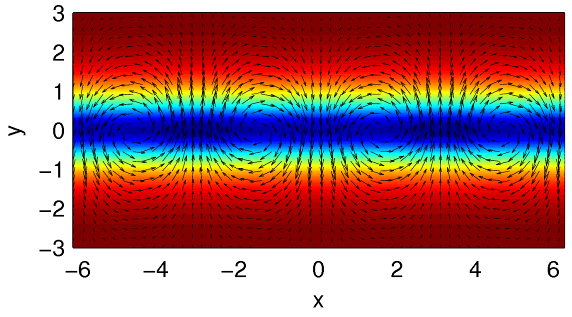
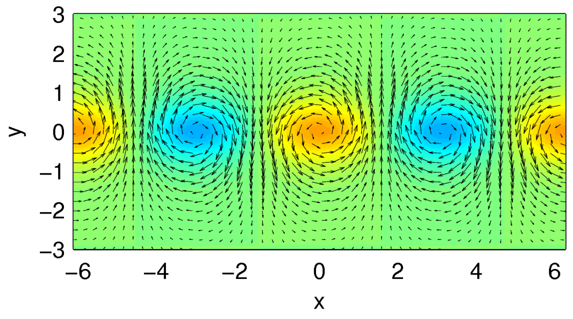
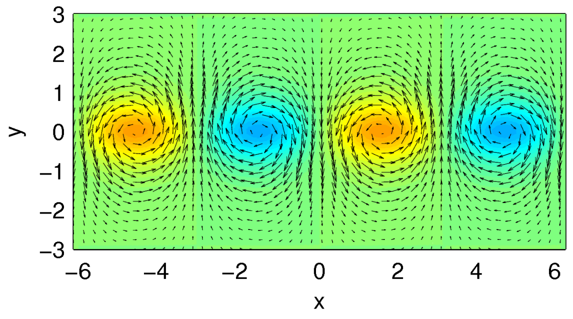


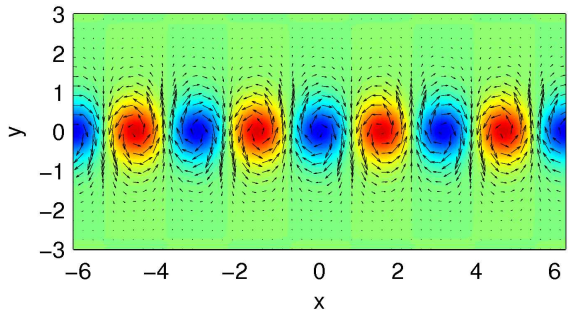
Figure 4.6: The mean flow corresponding to the Stuart solution (2.9) ($c = 1$ and $\epsilon = 0.42$). The vectors represent the velocity field and the colors indicate the strength of the vorticity.



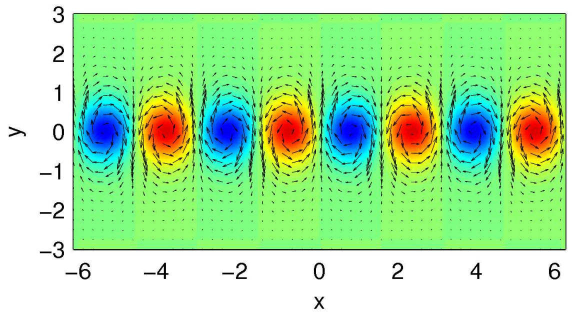
(a) mode 1



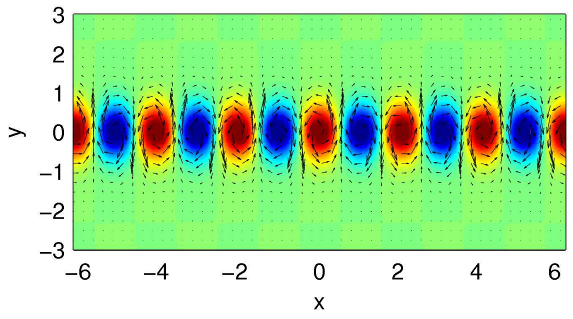
(b) mode 2



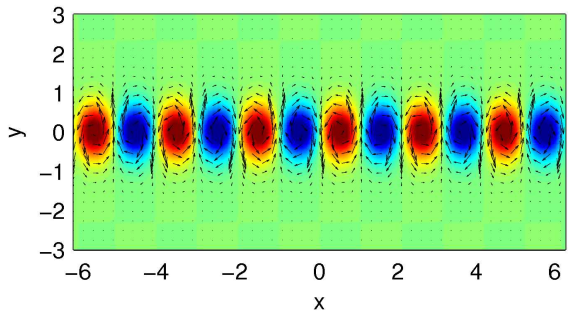
(c) mode 3



(d) mode 4



(e) mode 5



(f) mode 6

Figure 4.7: The POD modes corresponding to the time-dependent amplitudes in fig. 4.4. Each row shows a pair of modes with the same wave number. The vectors represent the velocity field and the colors indicate the strength of the vorticity.

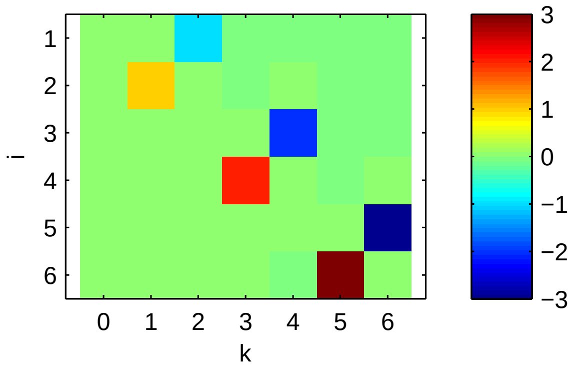


Figure 4.8: The coefficient $q_{i0k}$ of the Galerkin system of the Stuart solution.

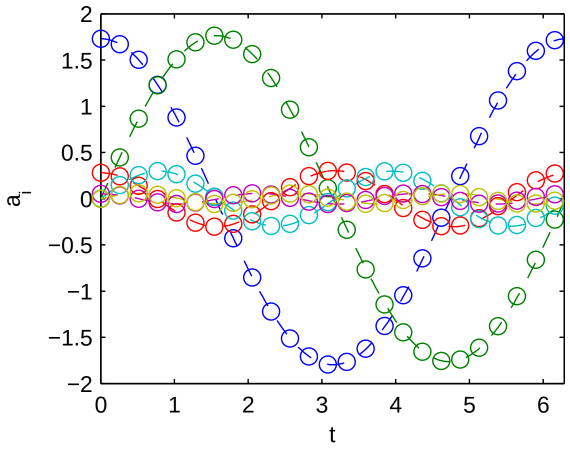


Figure 4.9: A comparison of the POD (o) and Galerkin system integrated (–) temporal coefficients of the Stuart solution.

Note that the pressure term vanishes because of the homogeneous boundary conditions (compare with eq. (3.39)). The numerical results show that only the $q_{i0k} = -(\mathbf{u}_i, (\mathbf{u}_0 \cdot \nabla)\mathbf{u}_k)_\Omega$ coefficient has entries that are non-zero (up to neglible small contributions). This term describes the convection of the fluctuations by the mean flow. More specifically only the off diagonal entries surrounding the diagonal are non-zero, see fig. 4.8. These are exactly the angular velocities of the temporal coefficients in fig. 4.4. The computed Galerkin system is integrated for one period. The result is shown in fig. 4.9. Small errors are observed that are mainly related to numerical discretization errors and not so much to truncation of the velocity field expansion.

## 4.3 ABC flow

In this section, the proper orthogonal decomposition and the Galerkin system are computed for the ABC flow. (see §2.3). This solution is readily separated into temporal and spatial basis functions. The POD is analytically computed and for comparison also via the method of snapshots. Peculiarities associated with short time horizons and incommensurable frequencies are illustrated with examples. In appendix B, MATLAB source code is provided for all examples.

### 4.3.1 Proper orthogonal decomposition and Galerkin system

The convecting ABC flow $(2.15)$ is decomposed into a constant base flow and a fluctuation part

$$\begin{aligned}\mathbf{u} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} &+ B\cos(c_1 t)\begin{bmatrix} 0 \\ \sin(x) \\ \cos(x) \end{bmatrix} + B\sin(c_1 t)\begin{bmatrix} 0 \\ -\cos(x) \\ \sin(x) \end{bmatrix} \\ &+ C\cos(c_2 t)\begin{bmatrix} \cos(y) \\ 0 \\ \sin(y) \end{bmatrix} + C\sin(c_2 t)\begin{bmatrix} \sin(y) \\ 0 \\ -\cos(y) \end{bmatrix} \\ &+ A\cos(c_3 t)\begin{bmatrix} \sin(z) \\ \cos(z) \\ 0 \end{bmatrix} + A\sin(c_3 t)\begin{bmatrix} -\cos(z) \\ \sin(z) \\ 0 \end{bmatrix}.\end{aligned} \tag{4.16}$$

The spatial inner product is defined over the cube $[0, 2\pi]^3$. It is easily observed that the spatial Fourier modes, in the expression above, are orthogonal. The temporal inner product is defined over the time interval $[0, 2\pi]$.

For the numerical examples, we set: $A = \sqrt{3}$, $B = \sqrt{2}$, $C = 1$. In the following we consider four examples. The first example shows a typical POD outcome. The second example reveals a problem that arises when the second moments are not adequately resolved. In the last example the first and second order statistics are both insufficiently described. These problems can result in erroneous interpretations of the physics.

#### Example 1, harmonic flow

Consider the following values:

- $c_1 = 1$, $c_2 = 2$ and, $c_3 = 3$.

Substitution in $(4.16)$ yields a decomposition with first, second and third harmonics and a constant mean flow. The POD merely sorts the Fourier modes with respect to their energy content and the basis is unchanged. This is confirmed by the computed weights $T'_{ij}$ in expression $(3.20)$, which are all equal. The normality of the modes implies that the amplitudes of the temporal

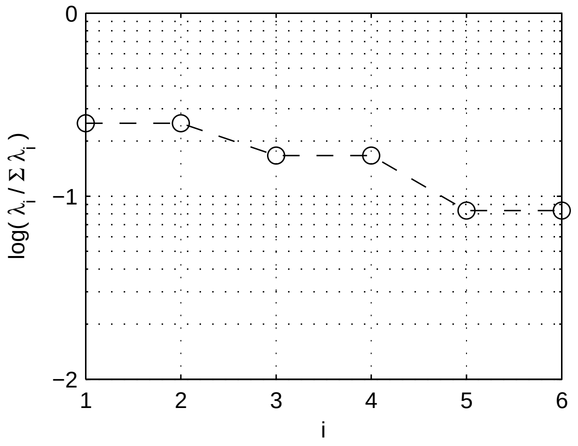


Figure 4.10: ABC flow example 1. The POD spectrum.

coefficients are respectively $A(2\pi)^{3/2}$, $B(2\pi)^{3/2}$ and $C(2\pi)^{3/2}$. The spectrum of the correlation matrix is shown in figure 4.10. The time-dependent amplitudes are shown in fig. 4.11. Corresponding POD modes are not shown. Like the Stuart solution above, POD yields the same result as a Fourier decomposition. For this purely convective flow, the spatial modes and temporal amplitudes appear in pairs of equal wave number, frequency and energy. This behaviour is reflected in the Galerkin system coefficients $q_{ijk}$, the only non zero terms are: $q_{102} = -q_{201} = 3$, $q_{304} = -q_{403} = 1$ and, $q_{506} = -q_{605} = 2$. These are the angular velocities of the temporal coefficients in fig. 4.11 (compare with fig. 4.4). Note that the pressure term vanishes because of the homogeneous boundary conditions (compare with eq. $(3.39)$).

For validation purposes, the projection of the diffusion term in the Navier–Stokes equation is also computed, although absent in the Euler equation. This term is a matrix with negative ones on its diagonal

$$l_{ij} = (\mathbf{u}_i, \triangle \mathbf{u}_j)_\Omega = -\delta_{i,i}, \tag{4.17}$$

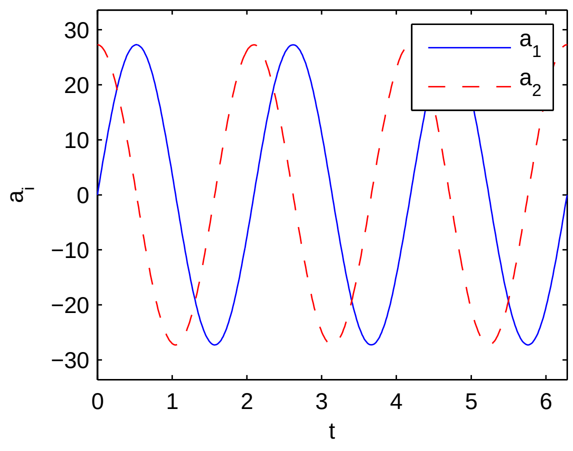
where the indices are $i = 1, \ldots, 6$ and $j = 0, \ldots, 6$.

## Example 2, mixed frequency flow

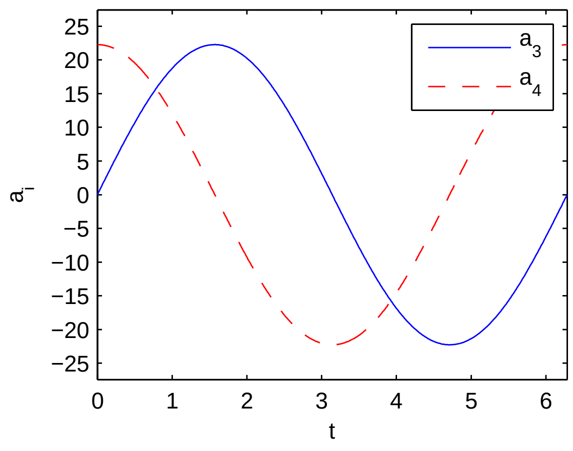
Consider the following values:

- $c_1 = 1$, $c_2 = \sqrt{2} \approx 1.4142$ and, $c_3 = \sqrt{3} \approx 1.7321$.

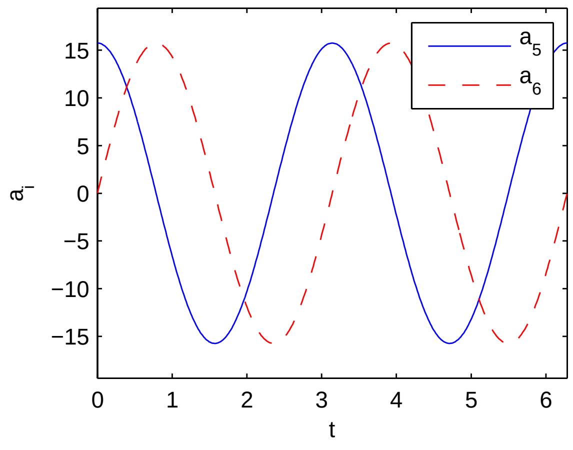
Substitution in $(4.16)$ yields a decomposition with incommensurable frequencies. The base mode for the POD in expression $(3.5)$ is defined as $\mathbf{u}_0 = [c_1, c_2, c_3]$. Since the POD is bi-orthogonal (by eq. $(3.9)$ and $(3.23)$) and the time interval is short, frequencies and wave numbers cannot



(a)



(b)



(c)

Figure 4.11: ABC flow example 1. Temporal evolution of the modal amplitudes. Each subfigure shows a pair of coefficients with the same frequency.

| | | | | | | |
|---|---|---|---|---|---|---|
| 0.0000 | 0.0000 | 1.4858 | 0.0084 | 0.0000 | 0.0000 | 0.0039 |
| 0.0000 | -1.4858 | 0.0000 | 0.0000 | -0.5683 | 0.0062 | -0.0000 |
| 0.0000 | -0.0084 | -0.0000 | 0.0000 | -1.1884 | 0.0050 | -0.0000 |
| 0.0000 | -0.0000 | 0.5683 | 1.1884 | 0.0000 | -0.0000 | -0.3437 |
| 0.0000 | -0.0000 | -0.0062 | -0.0050 | 0.0000 | 0.0000 | 1.3925 |
| 0.0000 | -0.0039 | 0.0000 | 0.0000 | 0.3437 | -1.3925 | 0.0000 |

Table 4.1: ABC flow example 2. The coefficient $q_{i0k}$ for $i = 1, \ldots, 6$, $k = 0, \ldots, 6$.

be clearly separated. This observation is confirmed by the weights $T'_{ij}$ for construction of the modes in expression (3.20), which are shown in fig. 4.12. The corresponding spectrum is shown in fig. 4.13. The modal amplitudes exhibit mixed frequencies, see fig. 4.14. The flow, as in example 1, is purely convective, which is reflected in the GS coefficients, only the convection term $q_{i0k}$ is non zero, see fig. 4.15. However, the frequencies mix up, most dominantly in the temporal coefficients of the second and third pair, see also table 4.1. This problem is due to the short time interval, which results in non-converged second order moments.

## Example 3, mixed frequency flow with artificial production

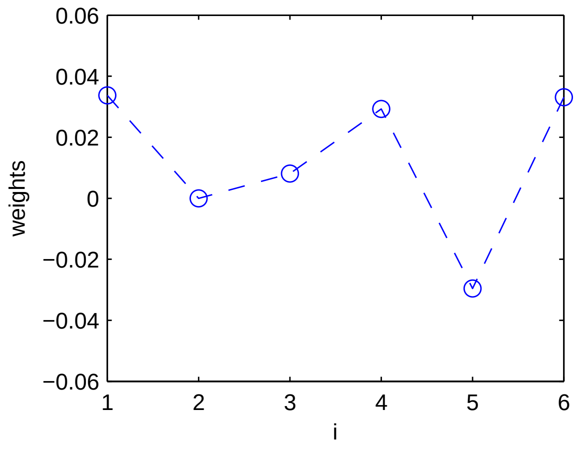
Consider the same values as in example 2:

- $c_1 = 1$, $c_2 = \sqrt{2} \approx 1.4142$ and, $c_3 = \sqrt{3} \approx 1.7321$.

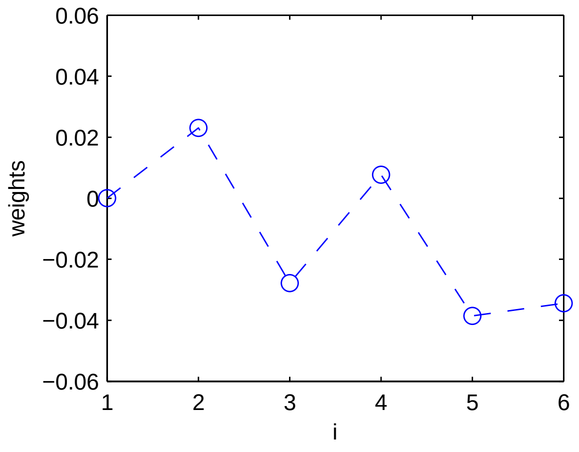
Substitution in (4.16) yields a decomposition with incommensurable frequencies. The base mode for the POD in expression (3.5) is now defined as the time averaged flow $\mathbf{u}_0 = \frac{1}{2\pi}\int_0^{2\pi} dt\, \mathbf{u}(\mathbf{x}, t)$. The mean flow not only comprises a constant part $[c_1, c_2, c_3]$, but also Fourier modes with weights $\sin(2\pi c_i)/(2\pi c_i)$ and $(\cos(2\pi c_i) - 1)/(2\pi c_i)$, where $i = 1, 2, 3$. This clearly illustrates the implication of a non-converged mean flow. In particular, this example was chosen to show the possible consequences of applying the POD, say the method of snapshots, to, for instance, a short CFD data ensemble. It should be emphasized at this point that the POD only yields physically sensible results for converged first and second order moments.

For this example, similar problems arise as in example 2: frequencies and wave numbers are mixed up. The spectrum of the correlation matrix is shown in fig. 4.16. This spectrum deviates more from the pairwise arrangement as in example 2 (see fig. 4.13). The temporal coefficients look similar to the ones in fig. 4.14 and are not shown. The most striking difference follows from the coefficients of the Galerkin system. For this case, not only the coefficients $q_{i0k}$ are non zero, but also $q_{ij0} = -(\mathbf{u}_i, (\mathbf{u}_j \cdot \nabla)\mathbf{u}_0)_\Omega$. This means that we "artificially" have introduced production in the system. See fig. 4.17 for the diagrams of $q_{i0k}$ and $q_{ij0}$.

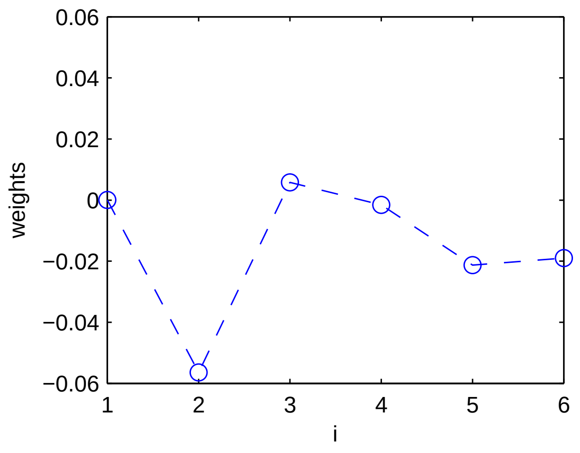
This example shows that POD must be carefully applied. However, it can also serve as a validation case for a numerical code. For this reason, the dissipation term is also computed in



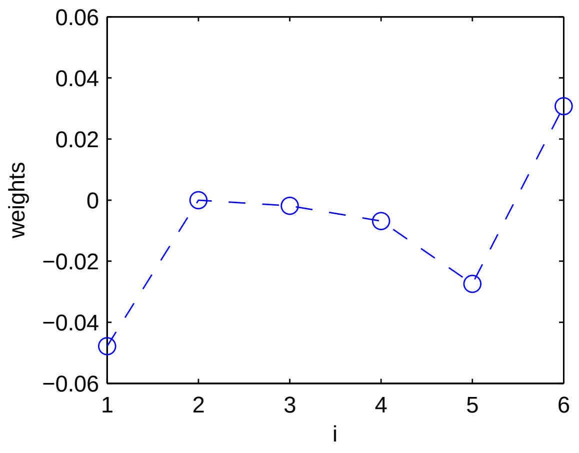
(a) $T_{1j}$



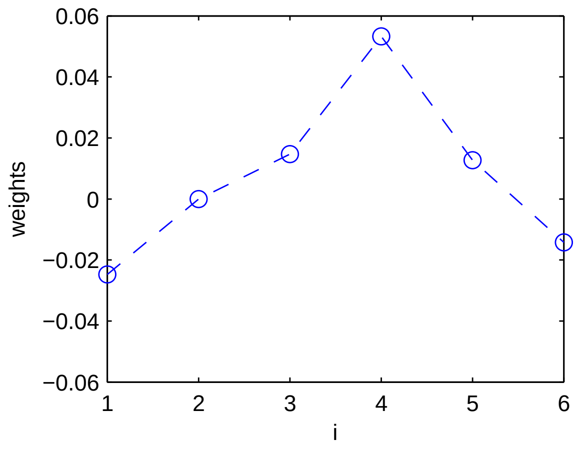
(b) $T_{2j}$



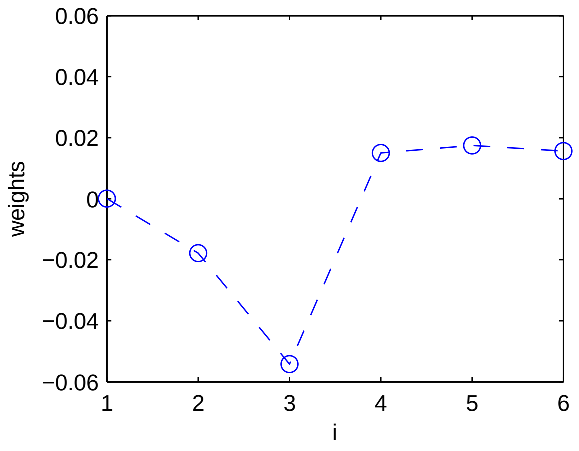
(c) $T_{3j}$



(d) $T_{4j}$



(e) $T_{5j}$



(f) $T_{6j}$

Figure 4.12: ABC flow example 2. The weights $T'_{ij}$ for construction of the POD modes.

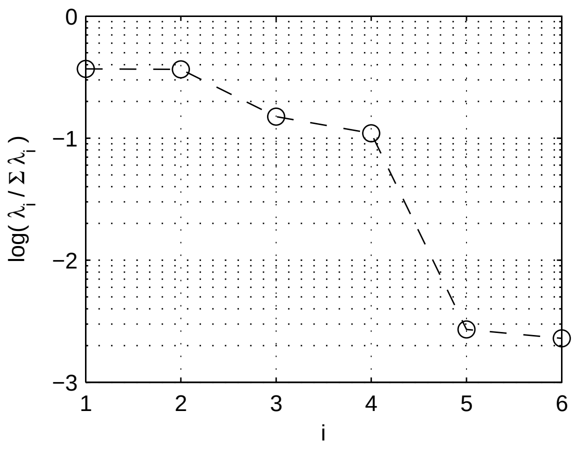
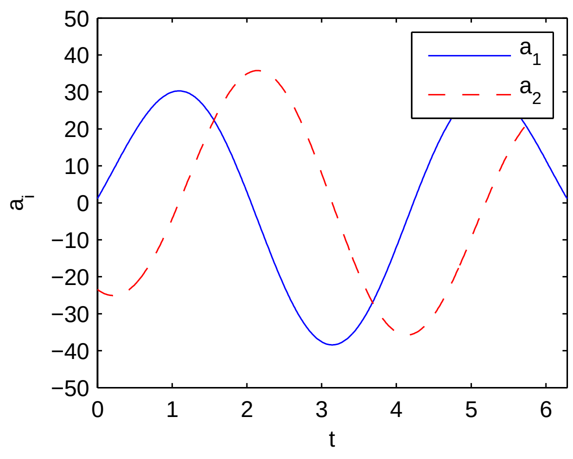
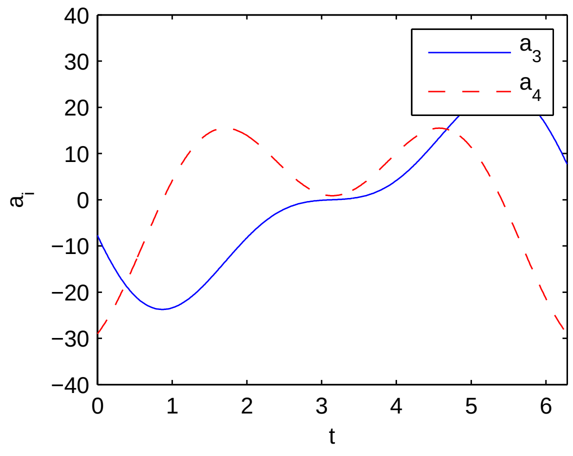


Figure 4.13: ABC flow example 2. The POD spectrum.

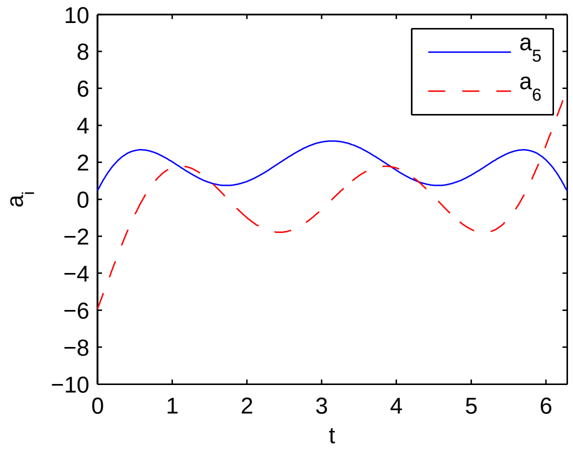
the MATLAB source code, see appendix B.



(a)



(b)



(c)

Figure 4.14: ABC flow example 2. Temporal evolution of the modal amplitudes. Each subfigure shows a pair of coefficients with similar energy content.

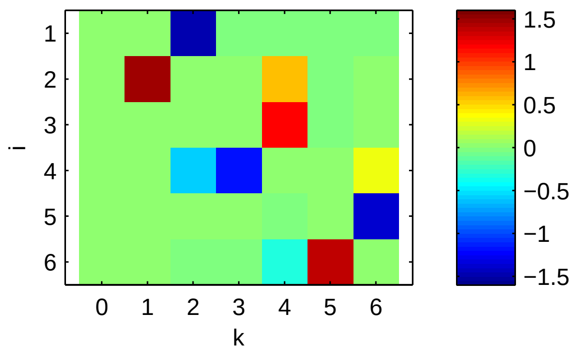


Figure 4.15: ABC flow example 2. The coefficient $q_{i0k}$ of the Galerkin system.

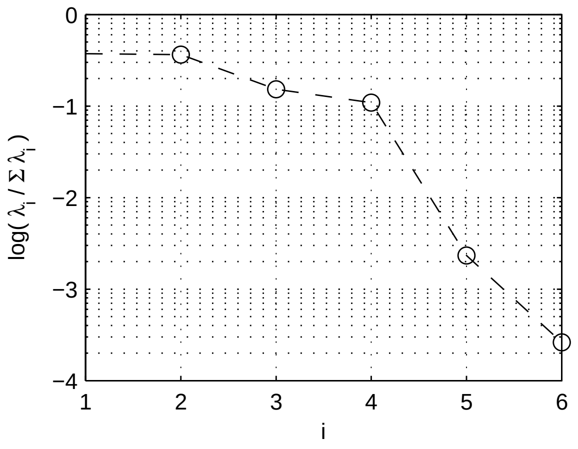
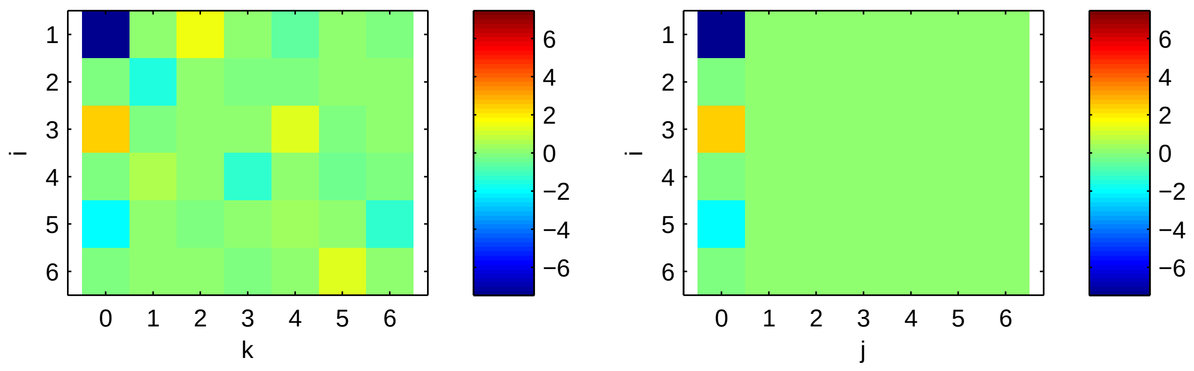


Figure 4.16: ABC flow example 3. The POD spectrum.



(a) $q_{i0k}$ (b) $q_{ij0}$

Figure 4.17: ABC flow example 3. The coefficients $q_{i0k}$ and $q_{ij0}$ of the Galerkin system.

# Appendix A. Galilean invariance

The substantial derivative $D/Dt = \partial/\partial t + \mathbf{u}\cdot\nabla$ is invariant under a Galilean transformation $(\mathbf{x}, t) \to (\mathbf{x}', t')$:

$$\mathbf{x} = \mathbf{x}' + \mathbf{c}t \tag{4.18a}$$
$$t = t'. \tag{4.18b}$$

**Proof**

The derivatives in the new coordinates $(\mathbf{x}', t')$ are given by

$$\nabla = \nabla' \tag{4.19a}$$
$$\frac{\partial}{\partial t} = \frac{\partial}{\partial t'} - \mathbf{c}\cdot\nabla'. \tag{4.19b}$$

Thus the total derivative for an observer with relative velocity $\mathbf{c}$ is

$$\frac{D}{Dt} = \frac{\partial}{\partial t'} + (\mathbf{u} - \mathbf{c})\cdot\nabla'. \tag{4.20}$$

Note from $(4.18\text{a})$ that

$$\mathbf{u} = \mathbf{u}' + \mathbf{c}, \tag{4.21}$$

where $\mathbf{u}' = \mathbf{u}(\mathbf{x}', t')$ [1]. In summary, the substantial derivative transforms to

$$\frac{D}{Dt} = \frac{\partial}{\partial t'} + \mathbf{u}'\cdot\nabla', \tag{4.22}$$

so the form of the convective derivative is invariant under transformation $(4.18)$.

Since the spatial gradient is invariant by $(4.19\text{a})$ and also the Laplace operator

$$\nabla^2 = \nabla'^2, \tag{4.23}$$

it holds that the complete Navier–Stokes equation is Galilean invariant. This property can be exploited to construct time-varying solutions (in a moving reference frame) from a stationary solution of the Navier–Stokes equation (in a fixed reference frame).

[1] A sketch of the coordinate vectors makes this clear.

# Appendix B. MATLAB code listing

## Convection diffusion equation

The following MATLAB code can be used to reproduce the results of §4.1.

### MATLAB file: `cde.m`

```
%% Header
%   @program name: cde.m
%   @dependency: MATLAB symbolic toolbox required, cde_gs.m
%   @task: Compute analytical POD / GM of a solution
%          of the 1D convection diffusion equation
%   @author: D.M. Luchtenburg
%   @see: Technical Report 01/2009 by DML, BRN and MS
%   @created: March 2009, DML

%%
clear all; close all; clc

%% Globals (defined for integration GS)
global Lmat

%% Symbols
syms x t alpha beta nu T w intg;

%% Parameters
nu = sym(0.1);     % 1 / Re
T  = sym(2*pi);  % period for time average
nModes = 2;      % number of modes
eps = 1e-6;      % tolerance for Runge Kutta GS integration

%% Init
v = zeros(nModes,1); v = sym(v);  % temporal coefficients
b = zeros(nModes,1); b = sym(b);  % spatial basis vectors
up = sym(0);                      % velocity field
Q = zeros(nModes,nModes); Q = sym(Q); % matrix needed for eigenprob.

%% Compute alpha, beta
alpha = (1/(4*nu))*( sqrt(  2+2*sqrt(1+16*nu^2) ) - 2 );
beta  = (1/(4*nu))*( sqrt( -2+2*sqrt(1+16*nu^2) ) );

%% Define basis functions
% v_1, v_2
v(1) = exp(-alpha*x)*cos(beta*x);
v(2) = exp(-alpha*x)*sin(beta*x);

%% Define temporal coefficients
% b_1, b_2
b(1) = cos(t);
b(2) = sin(t);
```

```
%% Construct velocity field
for i = 1:nModes
    up = up + b(i)*v(i);
end

%% Compute Q matrix
for i = 1:nModes
    for j = 1:nModes
        % sum over all modes
        for k = 1:nModes
            Q(i,j) = Q(i,j) + (1/T)*int(b(i)*b(k),t,0,T)*...
                int( v(k)*v(j), x,0,inf );
        end
    end
end

%% Compute eigenvalues
[eigenVectorArray,eigenValues] = eig(Q);
eigenValues = diag(eigenValues);
[eigenValuesTemp,I] = sort(eval(eigenValues),'descend');
eigenValues = eigenValues(I);
eigenVectorArray = eigenVectorArray(:,I);

%% Assign weights c, u_i = sum_j c_ji v_j
c = eigenVectorArray;

%% Compute POD modes
u = zeros(nModes,1); u = sym(u);
for i = 1:nModes
    % sum
    for j = 1:nModes
        u(i) = u(i) + c(j,i)*v(j);
    end
end
% Compute weights and normalized modes
for i = 1:nModes
    w = int( u(i)*u(i), x,0,inf );
    w = sqrt(w);
    u(i) = u(i) / w;
    c(:,i) = c(:,i) / w;
end

%% Compute Fourier coefficients
a = zeros(nModes,1); a = sym(a);
for i = 1:nModes
    a(i) = int( up*u(i), x,0,inf );
end

%% Compute Galerkin system
ug = sym(0);  % temp. storage
qij = zeros(nModes,nModes); qij = sym(qij);  % convection term
lij = zeros(nModes,nModes); lij = sym(lij);  % dissipation term
for i = 1:nModes
    for j = 1:nModes
        % Dissipation term: Int dx ( (d_xx u_j) * u_i )
        %  Compute 2nd derivative
        ug = diff(u(j),'x',2);
        %  integrand: (d_xx u_j) * u_i
        intg = ug*u(i); % note i-index: only POD modes!
        lij(i,j) = int( intg, x,0,inf );

        % Convection term: Int dx ( (d_x u_j) . u_i )
        ug = diff(u(j),'x',1);
        % integrand: (d_x u_j) . u_i
        intg = ug*u(i); % note i-index: only POD modes!
        qij(i,j) = -int( intg, x,0,inf );
    end
end
Lmat = double(simple(nu*lij+qij));

%% Integrate Galerkin system
a0 = zeros(nModes,1);
for i = 1:nModes
    a0(i) = double(subs(a(i),0)); % Get initial condition
end
% Integrate with Runge Kutta
options = odeset('RelTol',eps,'AbsTol',eps*ones(nModes,1));
[tInt,aInt] = ode45(@cde_gs,[0 40],a0,options);

%% Figures
close all

% Plot eigenvalues
figure
plot([1:nModes],eval(eigenValues./sum(eigenValues)),'--o')
grid on
xlabel('i'); ylabel('\lambda_i / \Sigma \lambda_i')
set(gca, 'XTick', [1:nModes]);
title('Eigenvalues')

% Plot Fourier coefficients
figure
p1 = ezplot(a(1),[0 20]);
set(p1,'Color','blue','LineStyle','-')
hold on
p2 = ezplot(a(2),[0 20]);
set(p2,'Color','red','LineStyle','--')
axis([0 20 -1.8 1.8])
title('')
xlabel('t'); ylabel('a_i');
legend('a_1','a_2')
title('Fourier coefficients')

% Plot POD modes
figure
p1 = ezplot(u(1),[0 40]);
set(p1,'Color','blue','LineStyle','-')
hold on
p2 = ezplot(u(2),[0 40]);
set(p2,'Color','red','LineStyle','--')
axis([0 40 -0.9 0.9])
```

```
title('')
xlabel('x'); ylabel('u_i');
legend('u_1','u_2')
title('POD modes')

% Plot comparison projected and integrated Fourier coefficients
figure
plot(tInt,aInt,'--x')
hold on
p1 = ezplot(a(1),tInt);
set(p1,'LineStyle','-')
p2 = ezplot(a(2),tInt);
set(p2,'LineStyle','-')
xlabel('t'); ylabel('a_i')
title('Comparison of projected a_i (-) and GS integrated a_i (--x)')
```

### MATLAB file: `cde_gs.m`

```
%% Header
%   @program name: cde_gs.m
%   @dependency: -
%   @task: Compute GS
%   @author: D.M. Luchtenburg
%   @see: Technical Report 01/2009 by DML, BRN and MS
%   @created: March 2009, DML

function [dadt] = cde_gs(t,a)

% Galerkin system of cde: dadt = L a

global Lmat
nModes = length(a);
dadt = zeros(nModes,1);

% Computed derivative
dadt = Lmat*a;
```

# Stuart solution

The following MATLAB code can be used to reproduce the results of §4.2.

### MATLAB file: `stuart.m`

```
%% Header
%   @program name: stuart.m
%   @dependency: stuart_q.m, stuart_gs.m
%   @task: Compute POD using the snapshot method and
%          GM of the convecting Stuart solution
%   @author: D.M. Luchtenburg
%   @see: Technical Report 01/2009 by DML, BRN and MS
%   @created: March 2009, DML
```

```
close all; clear all; clc;

%% Globals (defined for integration and integration GS)
global W q

%% Parameters
nModes = 6;   % number of computed modes

C = 1.1;   % C=1: laminar shear layer, C>1: cat eyes
c = 1;     % convection velocity

nx = 100;  % number of grid points in x-direction
ny =  50;  % number of grid points in y-direction
nt =  50;  % number of snapshots

eps1 = 1e-10;  % tolerance for orthonormality
eps2 = 1e-6;   % tolerance for Runge Kutta GS integration

% Figure properties
Quiver.Scale = 2;       % length of vector arrows for field plot
Quiver.LineWidth = 0.1; % thickness of vector arrows for field plot
XLim = [-2*pi 2*pi];    % x limits for field plot
YLim = [-3 3];          % y limits for field plot
qs = 2;                 % number of points skipped for vector field plot

%% Set up Grid
Lx = 4*pi; % two wave lengths
dx = Lx/(nx-1); x = [-Lx/2:dx:Lx/2];
Ly = 6;    % region with constant velocity at edges
dy = Ly/(ny-1); y = [-Ly/2:dy:Ly/2];
[X,Y] = meshgrid(x,y);

%% Create Snapshots
A = sqrt(C^2-1); e = A/C; % vortex strength
dt = 2*pi/(nt-1);
t = [0:dt:2*pi];

u = zeros(ny,nx,nt); v = zeros(ny,nx,nt); % init u,v-vel.
for i = 1:nt
    u(:,:,i) = c +     sinh(Y) ./ ( cosh(Y)+e*cos(X-c*t(i)) );
    v(:,:,i) = e*sin(X-c*t(i)) ./ ( cosh(Y)+e*cos(X-c*t(i)) );
end

%% Compute mean corrected snapshots: u' = u - <u>
uMean = mean(u,3); vMean = mean(v,3);
for i = 1:nt
    u(:,:,i) = u(:,:,i) - uMean;
    v(:,:,i) = v(:,:,i) - vMean;
end

%% Compute weighting matrix W for integration
% use simple midpoint quadrature rule
W = dx*dy*ones(ny,nx); % inner points
W(: , 1) = 0.5*dx*dy*ones(ny,1); % boundary
W(: ,nx) = 0.5*dx*dy*ones(ny,1); % boundary
W( 1, :) = 0.5*dx*dy*ones(1,nx); % boundary
```

```
W(ny, :) = 0.5*dx*dy*ones(1,nx); % boundary
W(1 , 1) = 0.25*dx*dy; % corner
W(ny, 1) = 0.25*dx*dy; % corner
W(ny,nx) = 0.25*dx*dy; % corner
W(1 ,nx) = 0.25*dx*dy; % corner

%% Compute correlation matrix C with snapshot method
%  C_ij = Int u'(x,t_i).u'(x,t_j) dx
C = zeros(nt,nt); % init
for i = 1:nt
    for j = i:nt
        C(i,j) = sum(sum(W.*( ...
            u(:,:,i).*u(:,:,j) + ...
            v(:,:,i).*v(:,:,j) )));
        C(j,i) = C(i,j);  % use symmetry
    end
end
C = C ./ nt;

%% Compute eigenvalues and eigenvectors of C
% Note that the returned eigenvectors are unit vectors,
% i.e. <a_i a_i> = 1
[V,D] = eig(C);
[D,I] = sort(diag(D),'descend');
V = V(:,I);

%% Scale Fourier Coefficients: <a_i a_i> = lambda_i
% a_i(t_j) = V(j,i)*sqrt(Nt*D(i))
FourCoef = zeros(nt,nModes);
for i = 1:nModes
    for j = 1:nt
        ModeFactor = sqrt(nt*D(i));
        FourCoef(j,i) = V(j,i)*ModeFactor;
    end
end

%% Compute POD modes
% u_i = (1/sqrt(Nt*D(i))) * Sum_{j=1}^Nt u'(x,t_j)*V(j,i)
uPOD = zeros(ny,nx,nModes); vPOD = zeros(ny,nx,nModes); % init
for i = 1:nModes
    for j = 1:nt
        uPOD(:,:,i) = uPOD(:,:,i) + V(j,i)*u(:,:,j);
        vPOD(:,:,i) = vPOD(:,:,i) + V(j,i)*v(:,:,j);
    end
    %   Normalize
    modeFactor = 1 ./ sqrt(nt*D(i));
    uPOD(:,:,i) = uPOD(:,:,i)*modeFactor;
    vPOD(:,:,i) = vPOD(:,:,i)*modeFactor;
end

%% Check normalization POD modes
Id = zeros(nModes,nModes);
for i=1:nModes
    for j = 1:nModes
        Id(i,j) = sum(sum(W.*( ...
            uPOD(:,:,i).*uPOD(:,:,j) + ...
            vPOD(:,:,i).*vPOD(:,:,j) )));
    end
end
if( abs(norm(eye(nModes)-Id)) > eps1 )
    Id
    disp('Error: POD modes not orthonormal')
end

%% Compute Fourier coefficients by projection (check)
FourCoefProj = zeros(nt,nModes);
for i = 1:nModes
    for j = 1:nt
        FourCoefProj(j,i) = sum(sum(W.*( ...
            u(:,:,j).*uPOD(:,:,i) + ...
            v(:,:,j).*vPOD(:,:,i) )));
    end
end

%% Compute Galerkin system coefficients
q = zeros(nModes,nModes+1,nModes+1);
q = stuart_q(X,Y,uMean,vMean,uPOD,vPOD);

%% Integrate Galerkin system
qijk=q;
aIC = FourCoefProj(1,:);  % initial conditions
options = odeset('RelTol',eps2,'AbsTol',eps2*ones(nModes,1));
[tInt,A] = ode45(@stuart_gs,[0 4*pi],aIC,options);

%% Figures
close all

% Plot eigenvalues
figure;
semilogy([1:nt],D./sum(D),'o-');
% axis([1 nt 1.e-3 1])
xlabel('i'); ylabel('\lambda_i / \Sigma \lambda_i')
grid on
title('Normalized eigenvalues')

% Plot comparison direct and projected Fourier coefficients
figure
plot(t,FourCoef(:,1),'r',t,FourCoefProj(:,1),'ro',...
    t,FourCoef(:,2),'g',t,FourCoefProj(:,2),'go')
xlabel('t'); ylabel('a_i')
title('Fourier coefficients (direct -, projection o)')

% Plot Fourier coefficients
for i = 1:2:nModes
    figure
    plot(t,FourCoef(:,i),'b-',...
         t,FourCoef(:,i+1),'r--')
    legend(['a_', num2str(i)],['a_', num2str(i+1)])
    xlabel('t'); ylabel('a_i');
    title('Fourier coefficients')
end
```

```
% Plot Phase portraits of Fourier coefficients
figure
for i = 1:nModes/2
    subplot(nModes/2,2,2*i-1)
    plot(FourCoef(:,2*i-1),FourCoef(:,2*i),'-.')
    xlabel(['a_',num2str(2*i-1)]); ylabel(['a_',num2str(2*i)]);
    title('Phase portraits');
    axis('equal')
end
for i = 1:nModes/2-1
    subplot(nModes/2,2,2*i+2)
    plot(FourCoef(:,1),FourCoef(:,2*i+2),'-.')
    xlabel('a_1'); ylabel(['a_',num2str(2*i+2)]);
    title('Phase portraits');
end

% Plot mean flow
figure;
%  vorticity
[curlz,cav]= curl(X,Y,uMean,vMean);
pcolor(X,Y,curlz); shading interp
% caxis([-1.12 0]); colorbar;
hold on;
%  vortor plot
Xq = X(1:qs:end, 1:qs:end);
Yq = Y(1:qs:end, 1:qs:end);
Uq = uPOD(1:qs:end, 1:qs:end,i);
Vq = vPOD(1:qs:end, 1:qs:end,i);
quiver(Xq, Yq, Uq, Vq, Quiver.Scale, 'k-', 'LineWidth', Quiver.LineWidth);
axis equal;
set(gca, 'XLim', XLim);
set(gca, 'YLim', YLim);
xlabel('x'); ylabel('y');
title(['mean flow'])

% Plot POD modes
for i = 1:nModes
    figure;
    % vorticity
    [curlz,cav]= curl(X,Y,uPOD(:,:,i),vPOD(:,:,i));
    pcolor(X,Y,curlz); shading interp
    caxis([-1.2 1.2]); % colorbar;
    hold on;
    % vortor plot
    Xq = X(1:qs:end, 1:qs:end);
    Yq = Y(1:qs:end, 1:qs:end);
    Uq = uPOD(1:qs:end, 1:qs:end,i);
    Vq = vPOD(1:qs:end, 1:qs:end,i);
    quiver(Xq, Yq, Uq, Vq, Quiver.Scale, 'k-', 'LineWidth', Quiver.LineWidth);
    axis equal;
    set(gca, 'XLim', XLim);
    set(gca, 'YLim', YLim);
    xlabel('x'); ylabel('y');
    title(['POD mode ',num2str(i)])
end
```

```
% Plot q_i0k (convection)
figure
image(squeeze(q(:,1,:)),'CDataMapping','scaled')
cm = max(abs(min(min(squeeze(q(:,1,:))))),max(max(squeeze(q(:,1,:)))));
caxis([-cm cm])
xlabel('k'); ylabel('i');
set(gca, 'XTickLabel', [0:nModes]);
colorbar
xlabel('k'); ylabel('i');
title('q_{i0k} (->convection)')

% Plot comparison projected and GS integrated Fourier coefficients
figure
plot(tInt,A,'--x')
hold on
plot(t,FourCoef,'-')
xlabel('t'); ylabel('a_i')
title('Comparison of projected a_i (-) and GS integrated a_i (--x)')
```

## MATLAB file: stuart_q.m

```
%% Header
%   @program name: stuart_q.m
%   @dependency: -
%   @task: Compute convection term q_ijk
%   @author: D.M. Luchtenburg
%   @see: Technical Report 01/2009 by DML, BRN and MS
%   @created: March 2009, DML

function [q] = stuart_q(X,Y,uMean,vMean,uPOD,vPOD)

% Compute convection term q_ijk for Stuart solution
%  q_ijk = - Int dxdy [ ( u_j*d/dx(u_k) + v_j*d/dy(u_k) )*u_i + ...
%                       ( u_j*d/dx(v_k) + v_j*d/dy(v_k) )*v_i ]

global W

nModes = size(uPOD,3);
nx = size(uMean,2);
ny = size(uMean,1);
dx = X(1,2)-X(1,1);
dy = Y(2,1)-Y(1,1);

% Init
q = zeros(nModes,nModes+1,nModes+1);
dukdx = zeros(ny,nx); dukdy = zeros(ny,nx);
dvkdx = zeros(ny,nx); dvkdy = zeros(ny,nx);

% Assemble mean and POD modes in one array
u = zeros(ny,nx,nModes+1); v = zeros(ny,nx,nModes+1);
u(:,:,1) = uMean; v(:,:,1) = vMean;
u(:,:,2:nModes+1) = uPOD; v(:,:,2:nModes+1) = vPOD;

% Compute q_ijk
for i = 1:nModes
```

```
    for j = 1:nModes+1
        for k = 1:nModes+1
            [dukdx dukdy] = gradient(u(:,:,k),dx,dy);
            [dvkdx dvkdy] = gradient(v(:,:,k),dx,dy);
            q(i,j,k) = -sum(sum(W.*(...
                (u(:,:,j).*dukdx + v(:,:,j).*dukdy).*u(:,:,i+1) + ...
                (u(:,:,j).*dvkdx + v(:,:,j).*dvkdy).*v(:,:,i+1) )));
        end
    end
end
```

### MATLAB file: `stuart_gs.m`

```
%% Header
%   @program name: stuart_gs.m
%   @dependency: -
%   @task: Compute GS
%   @author: D.M. Luchtenburg
%   @see: Technical Report 01/2009 by DML, BRN and MS
%   @created: March 2009, DML

function [dadt] = stuart_gs(t,a)

% Galerkin system of Stuart:
%  dadt = sum_{j,k=0}^N q_ijk * a_j*a_k

global q

c = [1; a]; % define: a_0 = 1
nModes = length(a);
dadt = zeros(nModes,1); % init

% Computed derivative
for iMode = 1:nModes
    for j = 1:(nModes+1)
        for k = 1:(nModes+1)
            dadt(iMode) = dadt(iMode) + q(iMode,j,k)*c(j)*c(k);
        end
    end
end
```

# ABC flow

The following MATLAB code can be used to reproduce the results of §4.3.

### MATLAB file: `abc_example1_2.m`

Use the following m-file for ABC flow example 1, 2.

```
%% Header
%   @program name: abc_example1_2.m
%   @dependency: MATLAB symbolic toolbox required, abc_gs.m
%   @task: Compute POD and GM of the convecting ABC flow
%   @author: D.M. Luchtenburg
%   @see: Technical Report 01/2009 by DML, BRN and MS
%   @created: April 2009, DML

close all; clear all; clc;

%% Globals (defined for integration GS)
global qijk

%% Symbols
syms x y z t c1 c2 c3 A B C T w intg;

%% Parameters
% coefficients ABC flow
A = sym(sqrt(3));
B = sym(sqrt(2));
C = sym(1);

% % example 1: convection velocities
% c1 = sym(1);
% c2 = sym(2);
% c3 = sym(3);
% example 2: convection velocities
c1 = sym(1);
c2 = sym(sqrt(2));
c3 = sym(sqrt(3));


T = sym(2*pi); % period for time average
nDim = 3;      % number of dimensions
nModes = 6;    % number of modes
eps = 1e-6;    % tolerance for Runge Kutta GS integration

%% Init
v = zeros(nDim,nModes); v = sym(v);  % temporal coefficients
b = zeros(nModes,1); b = sym(b);     % spatial basis vectors
u0 = zeros(nDim,1); u0 = sym(u0);    % mean flow
uc = zeros(nDim,1); uc = sym(uc);    % velocity field
up = zeros(nDim,1); up = sym(up);    % fluctuation part velocity field
Q = zeros(nModes,nModes); Q = sym(Q); % Matrix needed for eigenprob.

%% Define base velocity field
u0 = [c1; c2; c3];

%% Define basis vectors
% v_1, v_2
v(2,1) = sin(x); v(2,2) = -cos(x);
v(3,1) = cos(x); v(3,2) =  sin(x);
% v_3, v_4
v(1,3) = cos(y); v(1,4) =  sin(y);
v(3,3) = sin(y); v(3,4) = -cos(y);
% v_5, v_6
v(1,5) = sin(z); v(1,6) = -cos(z);
v(2,5) = cos(z); v(2,6) =  sin(z);
```

```
%% Define temporal coefficients
% b_1, b_2
b(1) = B*cos(c1*t);
b(2) = B*sin(c1*t);
b(3) = C*cos(c2*t);
b(4) = C*sin(c2*t);
b(5) = A*cos(c3*t);
b(6) = A*sin(c3*t);

%% Construct fluctuation and complete velocity field
% up(x,t)
for i = 1:nModes
    up = up + b(i)*v(:,i);
end
uc = u0 + up;

%% Compute Q matrix
for i = 1:nModes
    for j = 1:nModes
        % sum
        for k = 1:nModes
            for l = 1:nDim
                Q(i,j) = Q(i,j) + (1/T)*int(b(i)*b(k),t,0,2*pi)*...
                    int( int( int( v(l,k)*v(l,j), x,0,2*pi ), y,0,2*pi ), z,0,2*pi );
            end
        end
    end
end
Q = eval(Q);

%% Compute eigenvalues
[eigenVectorArray,eigenValues] = eig(Q);
eigenValues = diag(eigenValues);
[eigenValuesTemp,I] = sort(eigenValues,'descend');
eigenValues = eigenValues(I);
eigenVectorArray = eigenVectorArray(:,I);

%% Assign weights c, u_i = sum_j c_ji v_j
c = sym(eigenVectorArray);

%% Compute POD modes
u = zeros(nDim,nModes); u = sym(u);
for i = 1:nModes
    % sum
    for j = 1:nModes
        u(:,i) = u(:,i) + c(j,i)*v(:,j);
    end
end
% Compute weights and normalized modes
for i = 1:nModes
    w = int( int( int( u(:,i).'*u(:,i), x,0,2*pi ), y,0,2*pi ), z,0,2*pi );
    w = sqrt(w);
    u(:,i) = u(:,i) ./ w;
    c(:,i) = c(:,i) ./ w;
end
c = eval(c);
```

```
%% Compute Fourier coefficients
a = zeros(nModes,1); a = sym(a);
for i = 1:nModes
    a(i) = int( int( int( up.'*u(:,i), x,0,2*pi ), y,0,2*pi ), z,0,2*pi );
end

%% Compute Galerkin projection
umm = zeros(nDim,nModes+1); umm = sym(umm);
umm(:,1) = u0; umm(:,2:nModes+1) = u;  % store: mean, modes
ug = zeros(nDim,nDim); ug = sym(ug);   % temporary storage
qijk = zeros(nModes,nModes+1,nModes+1); qijk = sym(qijk); % convection term
lij = zeros(nModes,nModes+1); lij = sym(lij);             % dissipation term
lh =  ones(nDim,1); lh = sym(lh); % ones(3,1)
for i = 1:nModes
    for j = 1:nModes+1 % mean flow + modes
        % Dissipation term: Int dx ( (lapl u_j) * u_i )
        %  Compute 2nd derivatives
        for ig = 1:nDim
            ug(ig,1) = diff(umm(ig,j),'x',2);
            ug(ig,2) = diff(umm(ig,j),'y',2);
            ug(ig,3) = diff(umm(ig,j),'z',2);
        end
        %  integrand: lapl(u_j) . u_i
        intg = (ug*lh).'*umm(:,i+1); % note i-index: only POD modes!
        lij(i,j) = int( int( int( intg, x,0,2*pi ), y,0,2*pi ), z,0,2*pi);

        % convection term: Int dV ((grad u_k) u_j) . u_i
        for k = 1:nModes+1 % mean flow + modes
            % grad u_k
            for ig = 1:3
                ug(ig,1) = diff(umm(ig,k),'x');
                ug(ig,2) = diff(umm(ig,k),'y');
                ug(ig,3) = diff(umm(ig,k),'z');
            end
            % integrand: ((grad u_k) u_j) . u_i
            intg = (ug*umm(:,j)).'*umm(:,i+1); % note i-index: only POD modes!
            qijk(i,j,k) = -int( int( int( intg, x,0,2*pi ), y,0,2*pi ), z,0,2*pi );
        end
    end
end
% Numerical values
lij = eval(lij);
for k = 1:nModes+1 % mean flow + modes
    qijk(:,:,k) = eval(qijk(:,:,k));
end
% Pressure term
%  f_i = - Int grad p . u_i = Int D/Dt u . u_i
fp = zeros(nModes,1); fp = sym(fp);
dudt = zeros(nDim,1); dudt = sym(dudt);
for i = 1:nDim
    dudt(i) = diff(uc(i),'t');
    ug(i,1) = diff(uc(i),'x');
    ug(i,2) = diff(uc(i),'y');
    ug(i,3) = diff(uc(i),'z');
```

```
end
for i = 1:nModes
    % integrand: D/Dt u . u_i
    intg = (dudt + (ug*uc)).'*umm(:,i+1);
    fp(i) = int( int( int( intg, x,0,2*pi ), y,0,2*pi ), z,0,2*pi );
end

%% Integrate Galerkin system
a0 = zeros(nModes,1);
for i = 1:nModes
    a0(i) = double(subs(a(i),0)); % get initial condition
end
qijk = double(qijk);
lij = double(lij);
% Integrate with Runge Kutta
options = odeset('RelTol',eps,'AbsTol',eps*ones(nModes,1));
[tInt,aInt] = ode45(@abc_gs,[0 2*pi],a0,options);

%% Figures
close all

% Plot eigenvalues
figure
semilogy([1:nModes],(eigenValues./sum(eigenValues)),'--o')
grid on
xlabel('i'); ylabel('\lambda_i / \Sigma \lambda_i')
set(gca, 'XTick', [1:nModes]);
title('Eigenvalues')

% Plot weights
for i = 1:nModes
    figure
    plot([1:nModes],c(:,i),'--o')
    axis([1 nModes -0.07 0.07])
    xlabel('i'); ylabel('weights')
    set(gca, 'XTick', [1:nModes]);
    title(['Weights: T_', num2str(i),'j']);
end

% Plot Fourier coefficients
for i = 1:2:nModes
    figure
    p1 = ezplot(a(i),[0 eval(T)]);
    set(p1,'Color','blue','LineStyle','-')
    hold on
    p2 = ezplot(a(i+1),[0 eval(T)]);
    set(p2,'Color','red','LineStyle','--')
    title('')
    xlabel('t'); ylabel('a_i');
    legend(['a_', num2str(i)],['a_', num2str(i+1)])
    if( i==1 )
        axis([0 2*pi -50 50])
    elseif( i == 3 )
        axis([0 2*pi -40 40])
    else
        axis([0 2*pi -20 20])
    end
    title('Fourier coefficients')
end

% Plot dissipation term lij
figure
image(lij,'CDataMapping','scaled')
axis('equal')
xlabel('j'); ylabel('i');
set(gca, 'XTickLabel', [0:nModes]);
colorbar
title('dissipation term: l_{ij}')

% Plot q_i0k (convection)
figure
image(squeeze(double(qijk(:,1,:))),'CDataMapping','scaled')
axis('equal')
caxis([-1.6 1.6])
xlabel('k'); ylabel('i');
set(gca, 'XTickLabel', [0:nModes]);
colorbar
xlabel('k'); ylabel('i');
title('q_{i0k} (->convection)')

% Plot Comparison exact Fourier coefficients and integration GS
for i = 1:2:nModes
    figure
    hold on
    plot(tInt,aInt(:,i  ),'b.-')
    plot(tInt,aInt(:,i+1),'r.-')
    p1 = ezplot(a(i),[0 eval(T)]);
    set(p1,'Color','blue','LineStyle','-')
    p2 = ezplot(a(i+1),[0 eval(T)]);
    set(p2,'Color','red','LineStyle','--')
    legend('a_1','a_2')
    % axis
    if( i==1 )
        axis([0 2*pi -45 45])
    elseif( i == 3 )
        axis([0 2*pi -35 35])
    else
        axis([0 2*pi -20 20])
    end
    xlabel('t'); ylabel(['a_', num2str(i),', ','a_', num2str(i+1)]);
    title('Comparison exact (-) Fourier coefficient and integration GS (.-)')
end
```

## MATLAB file: abc_example3.m

Use the following m-file for ABC flow example 3.

```
%% Header
%   @program name: abc_example3.m
%   @dependency: MATLAB symbolic toolbox required, abc_gs.m
%   @task: Compute POD and GM of the convecting ABC flow
%   @author: D.M. Luchtenburg
%   @see: Technical Report 01/2009 by DML, BRN and MS
%   @created: April 2009, DML

close all; clear all; clc;

%% Globals (defined for integration GS)
global qijk

%% Symbols
syms x y z t c1 c2 c3 A B C T w intg;

%% Parameters
% coefficients ABC flow
A = sym(sqrt(3));
B = sym(sqrt(2));
C = sym(1);
% convection velocities
c1 = sym(1);
c2 = sym(sqrt(2));
c3 = sym(sqrt(3));

T = sym(2*pi); % period for time average
nDim = 3;      % number of dimensions
nModes = 6;    % number of modes
eps = 1e-6;    % tolerance for Runge Kutta GS integration
nu = 0;        % 1 / Re

%% Init
v = zeros(nDim,nModes); v = sym(v);  % temporal coefficients
b = zeros(nModes,1); b = sym(b);     % spatial basis vectors
ub = zeros(nDim,1); ub = sym(ub);    % base flow
u0 = zeros(nDim,1); u0 = sym(u0);    % mean flow
uc = zeros(nDim,1); uc = sym(uc);    % velocity field
up = zeros(nDim,1); up = sym(up);    % fluctuation part velocity field
Q = zeros(nModes,nModes); Q = sym(Q); % Matrix needed for eigenprob.

%% Define base velocity field
ub = [c1; c2; c3];

%% Define basis vectors
% v_1, v_2
v(2,1) = sin(x); v(2,2) = -cos(x);
v(3,1) = cos(x); v(3,2) =  sin(x);
% v_3, v_4
v(1,3) = cos(y); v(1,4) =  sin(y);
v(3,3) = sin(y); v(3,4) = -cos(y);
% v_5, v_6
v(1,5) = sin(z); v(1,6) = -cos(z);
v(2,5) = cos(z); v(2,6) =  sin(z);

%% Define temporal coefficients
% b_1, b_2
b(1) = B*(cos(c1*t)-sin(2*pi*c1)/(2*pi*c1));
b(2) = B*(sin(c1*t)+(cos(2*pi*c1)-1)/(2*pi*c1));
b(3) = C*(cos(c2*t)-sin(2*pi*c2)/(2*pi*c2));
```

```
b(4) = C*(sin(c2*t)+(cos(2*pi*c2)-1)/(2*pi*c2));
b(5) = A*(cos(c3*t)-sin(2*pi*c3)/(2*pi*c3));
b(6) = A*(sin(c3*t)+(cos(2*pi*c3)-1)/(2*pi*c3));

%% Construct fluctuation velocity field
for i = 1:nModes
    up = up + b(i)*v(:,i);
end

%% Mean velocity field
u0 = ub + B*sin(2*pi*c1)/(2*pi*c1)*v(:,1) + B*(1-cos(2*pi*c1))/(2*pi*c1)*v(:,2) + ...
          C*sin(2*pi*c2)/(2*pi*c2)*v(:,3) + C*(1-cos(2*pi*c2))/(2*pi*c2)*v(:,4) + ...
          A*sin(2*pi*c3)/(2*pi*c3)*v(:,5) + A*(1-cos(2*pi*c3))/(2*pi*c3)*v(:,6);

%% Compute Q matrix
for i = 1:nModes
    for j = 1:nModes
        % sum
        for k = 1:nModes
            for l = 1:nDim
                Q(i,j) = Q(i,j) + (1/T)*int(b(i)*b(k),t,0,2*pi)*...
                    int( int( int( v(l,k)*v(l,j), x,0,2*pi ), y,0,2*pi ), z,0,2*pi );
            end
        end
    end
end
Q = eval(Q);

%% Compute eigenvalues
[eigenVectorArray,eigenValues] = eig(Q);
eigenValues = diag(eigenValues);
[eigenValuesTemp,I] = sort(eigenValues,'descend');
eigenValues = eigenValues(I);
eigenVectorArray = eigenVectorArray(:,I);

%% Assign weights c, u_i = sum_j c_ji v_j
c = sym(eigenVectorArray);

%% Compute POD modes
u = zeros(nDim,nModes); u = sym(u);
for i = 1:nModes
    % sum
    for j = 1:nModes
        u(:,i) = u(:,i) + c(j,i)*v(:,j);
    end
end
% Compute weights and normalized modes
for i = 1:nModes
    w = int( int( int( u(:,i).'*u(:,i), x,0,2*pi ), y,0,2*pi ), z,0,2*pi );
    w = sqrt(w);
    u(:,i) = u(:,i) ./ w;
    c(:,i) = c(:,i) ./ w;
end
c = eval(c);
```

```
%% Compute Fourier coefficients
a = zeros(nModes,1); a = sym(a);
for i = 1:nModes
    a(i) = int( int( int( up.'*u(:,i), x,0,2*pi ), y,0,2*pi ), z,0,2*pi );
end

%% Compute Galerkin projection
umm = zeros(nDim,nModes+1); umm = sym(umm);
umm(:,1) = u0; umm(:,2:nModes+1) = u;  % store: mean, modes
ug = zeros(nDim,nDim); ug = sym(ug);   % temporary storage
qijk = zeros(nModes,nModes+1,nModes+1); qijk = sym(qijk); % convection term
lij = zeros(nModes,nModes+1); lij = sym(lij);             % dissipation term
lh =  ones(nDim,1); lh = sym(lh); % ones(3,1)
for i = 1:nModes
    for j = 1:nModes+1 % mean flow + modes
        % Dissipation term: Int dx ( (lapl u_j) * u_i )
        %  Compute 2nd derivatives
        for ig = 1:nDim
            ug(ig,1) = diff(umm(ig,j),'x',2);
            ug(ig,2) = diff(umm(ig,j),'y',2);
            ug(ig,3) = diff(umm(ig,j),'z',2);
        end
        %  integrand: lapl(u_j) . u_i
        intg = (ug*lh).'*umm(:,i+1); % note i-index: only POD modes!
        lij(i,j) = int( int( int( intg, x,0,2*pi ), y,0,2*pi ), z,0,2*pi);

        % convection term: Int dV ((grad u_k) u_j) . u_i
        for k = 1:nModes+1 % mean flow + modes
            % grad u_k
            for ig = 1:3
                ug(ig,1) = diff(umm(ig,k),'x');
                ug(ig,2) = diff(umm(ig,k),'y');
                ug(ig,3) = diff(umm(ig,k),'z');
            end
            % integrand: ((grad u_k) u_j) . u_i
            intg = (ug*umm(:,j)).'*umm(:,i+1); % note i-index: only POD modes!
            qijk(i,j,k) =  -int( int( int( intg, x,0,2*pi ), y,0,2*pi ), z,0,2*pi );
        end
    end
end
% Numerical values
lij = eval(lij);
for k = 1:nModes+1 % mean flow + modes
    qijk(:,:,k) = eval(qijk(:,:,k));
end
% Pressure term
%  f_i = - Int grad p . u_i = Int D/Dt u . u_i
fp = zeros(nModes,1); fp = sym(fp);
dudt = zeros(nDim,1); dudt = sym(dudt);
for i = 1:nDim
    dudt(i) = diff(uc(i),'t');
    ug(i,1) = diff(uc(i),'x');
    ug(i,2) = diff(uc(i),'y');
    ug(i,3) = diff(uc(i),'z');
end
for i = 1:nModes
```

```
    % integrand: D/Dt u . u_i
    intg = (dudt + (ug*uc)).'*umm(:,i+1);
    fp(i) = int( int( int( intg, x,0,2*pi ), y,0,2*pi ), z,0,2*pi );
end

%% Integrate Galerkin system
a0 = zeros(nModes,1);
for i = 1:nModes
    a0(i) = double(subs(a(i),0)); % get initial condition
end
qijk = double(qijk);
lij = double(lij);
% Integrate with Runge Kutta
options = odeset('RelTol',eps,'AbsTol',eps*ones(nModes,1));
[tInt,aInt] = ode45(@abc_gs,[0 2*pi],a0,options);

%% Figures
close all

% Plot eigenvalues
figure
semilogy([1:nModes],(eigenValues./sum(eigenValues)),'--o')
grid on
xlabel('i'); ylabel('\lambda_i / \Sigma \lambda_i')
set(gca, 'XTick', [1:nModes]);
title('Eigenvalues')

% Plot weights
for i = 1:nModes
    figure
    plot([1:nModes],c(:,i),'--o')
    axis([1 nModes -0.06 0.06])
    xlabel('i'); ylabel('weights')
    set(gca, 'XTick', [1:nModes]);
    title(['Weights: T_', num2str(i),'j']);
end

% Plot Fourier coefficients
for i = 1:2:nModes
    figure
    p1 = ezplot(a(i),[0 eval(T)]);
    set(p1,'Color','blue','LineStyle','-')
    hold on
    p2 = ezplot(a(i+1),[0 eval(T)]);
    set(p2,'Color','red','LineStyle','--')
    title('')
    xlabel('t'); ylabel('a_i');
    legend(['a_', num2str(i)],['a_', num2str(i+1)])
    if( i==1 )
        axis([0 2*pi -45 45])
    elseif( i == 3 )
        axis([0 2*pi -35 35])
    else
        axis([0 2*pi -10 10])
    end
    title('Fourier coefficients')
```

```
end

% Plot dissipation term lij
figure
image(lij,'CDataMapping','scaled')
axis('equal')
xlabel('j'); ylabel('i');
set(gca, 'XTickLabel', [0:nModes]);
colorbar
title('dissipation term: l_{ij}')

% Plot q_i0k (convection)
figure
image(squeeze(double(qijk(:,1,:))),'CDataMapping','scaled')
axis('equal')
cm = max(abs(min(min(squeeze(qijk(:,:,1))))),max(max(squeeze(qijk(:,:,1)))));
caxis([-cm cm])
xlabel('k'); ylabel('i');
set(gca, 'XTickLabel', [0:nModes]);
colorbar
xlabel('k'); ylabel('i');
title('q_{i0k} (->convection)')

% Plot q_ij0 (production)
figure
image(squeeze(qijk(:,:,1)),'CDataMapping','scaled')
axis('equal')
cm = max(abs(min(min(squeeze(qijk(:,:,1))))),max(max(squeeze(qijk(:,:,1)))));
caxis([-cm cm])
xlabel('j'); ylabel('i');
set(gca, 'XTickLabel', [0:nModes]);
colorbar
xlabel('j'); ylabel('i');
title('q_{ij0} (->production)')

% Plot Comparison exact Fourier coefficients and integration GS
for i = 1:2:nModes
    figure
    hold on
    plot(tInt,aInt(:,i  ),'b.-')
    plot(tInt,aInt(:,i+1),'r.-')
    p1 = ezplot(a(i),[0 eval(T)]);
    set(p1,'Color','blue','LineStyle','-')
    p2 = ezplot(a(i+1),[0 eval(T)]);
    set(p2,'Color','red','LineStyle','--')
    legend('a_1','a_2')
    % axis
    if( i==1 )
        axis([0 2*pi -45 45])
    elseif( i == 3 )
        axis([0 2*pi -35 35])
    else
        axis([0 2*pi -10 10])
    end
    title('Comparison exact (-) Fourier coefficient and integration GS (.-)')
    xlabel('t'); ylabel(['a_', num2str(i),', ','a_', num2str(i+1)]);
```

```
end
```

## MATLAB file: abc_gs.m

```
%% Header
%   @program name: abc_gs.m
%   @dependency: -
%   @task: Compute GS
%   @author: D.M. Luchtenburg
%   @see: Technical Report 01/2009 by DML, BRN and MS
%   @created: March 2009, DML

function [dadt] = abc_gs(t,a)

% compute GS: dadt = qijk*aj*ak

global qijk

nModes = length(a);
c = [1; a];             % define: a0 = 1
dadt = zeros(nModes,1);

% Sum q_ijk*a_j*a_k
for iMode = 1:nModes
    for j = 1:(nModes+1)
        for k = 1:(nModes+1)
            dadt(iMode) = dadt(iMode) + qijk(iMode,j,k)*c(j)*c(k);
        end
    end
end
```

# Bibliography

[Chatterjee 2000] Chatterjee, A. 2000 An introduction to the proper orthogonal decomposition. *Curr. Sci.* **7**, 808–817.

[Cordier & Bergmann 2003] Cordier, L. & Bergmann, M. 2003 *Proper Orthogonal Decomposition: An Overview*, Von Kármán Institut for Fluid Dynamics.

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

[Holmes *et al.* 1998] Holmes, P., Lumley, J.L. & Berkooz, G. 1998 *Turbulence, Coherent Structures, Dynamical Systems and Symmetry*, Cambridge University Press.

[Lumley 1967] Lumley, J.L. 1967 *The structure of inhomogeneous turbulent flows*, In A.M. Yaglom, & V.I. Tatarski *Atmospheric Turbulence & Wave Propagation*, 166–178.

[Noack 2006] Noack, B.R. 2006 *Niederdimensionale Galerkin-Modelle für laminare und transitionelle freie Scherströmungen*, Habilitationsschrift, Fakultät V für Verkehrs- und Maschinensysteme, Technische Universität Berlin.

[Noack *et al.* 2005] Noack, B.R., Papas, P. & Monkewitz, P.A. 2005 The need for a pressure-term representation in empirical Galerkin models of incompressible shear flows. *J. Fluid Mech.* **523**, 339–365.

[Remfer & Fasel 1994] Rempfer, D. & Fasel, F.H. 1994 Evolution of three-dimensional coherent structures in a flat-plate boundary-layer. *J. Fluid Mech.* **260**, 351–375.

[Sirovich 1987] Sirovich, L. 1987 Turbulence and the Dynamics of Coherent Structures. Part 1 : Coherent Structures. *Quarterly of Applied Mathematics* **45**(3), 561–571.

[Strang 1986] Strang, G. 1986 *Introduction to applied mathematics.*, Wellesly-Cambridge Press.

[Stuart 1967] Stuart, J.T. 1967 On finite amplitude oscillations in laminar mixing layers. *J. Fluid Mech.* **29**, 417–440.

[Wu *et al.* 2003] Wu, C.G., Liang, Y.C., Lin, W.Z., Lee, H.P. & Lim, S.P. 2005 A note on equivalence of proper orthogonal decomposition methods. *J. Sound Vibr.* **265**: 5, 1103–1110.